Lecture Notes in Computer Science 5951

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Mumtaz Siddiqui Thomas Fahringer

Grid Resource Management

On-demand Provisioning, Advance Reservation, and Capacity Planning of Grid Resources

Springer

Authors

Mumtaz Siddiqui
Institute for Computer Science, University of Innsbruck
Technikerstraße 21a, 6020 Innsbruck, Austria
E-mail: Mumtaz.Siddiqui@uibk.ac.at

Thomas Fahringer
Distributed and Parallel Systems Group
Institute of Computer Science, University of Innsbruck
Technikerstraße 21a, 6020 Innsbruck, Austria
E-mail: tf@dps.uibk.ac.at

Library of Congress Control Number: 2009942714

CR Subject Classification (1998): C.2, F.2, D.2, H.3, I.2.11, C.4, I.6, K.6.5

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-11578-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-11578-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12833689 06/3180 5 4 3 2 1 0

dedicated to our families
for their love, encouragement, and steadfastness

Preface

The Grid computing paradigm has evolved tremendously since the beginning of this millennium. It enables a flexible, secure and coordinated resource sharing among dynamic collections of individuals and institutions. The evolution has occured in various sectors – different Grid infrastructures such as computational Grid, data Grid and knowledge Grid are available to fulfill the diverse requirements of scientific communities. Today various application development and execution environments are present to create new Grid-enabled applications and to execute them onto the Grid. During this timeframe, the major focus of the research community remained on the development of an effective Grid middleware that could lead to a robust operating and management system. The middleware provides services for security, scheduling, information dissemination and resource provisioning.

Despite all these efforts, resource allocation remained a challenge for building an effective Grid resource management, as Grid applications are continuously evolving over time. In the Grid, resource management intends to make high-performance computational resources available on-demand to anyone from anywhere at anytime without undermining the resource autonomy. This is still an art due to non-dedicated heterogeneous resources distributed under multiple trust domains spanning across the Internet under the dynamic Grid environments.

In this monograph, we address the challenges of providing an effective Grid resource management by applying various techniques such as automatic brokerage, dynamic allocation, on-demand resource synthesis, advance reservation and capacity planning. In contrast to the conventional computing environments, applying these techniques in the Grid is all but straightforward. Nevertheless, in the Grid, application of such techniques is of paramount importance for resource management with effective provisioning, better utilization and an optimal allocation.

First of all, we identify the requirement of automatic resource brokerage in the Grid. The development of the Grid with diversified applications competing for scarce resources accentuates the need for an automatic resource

brokerage – this is required to enable the Grid to shield its middleware complexities and to lead toward an invisible, intuitive, and robust runtime environment. Conventional research in the area of Grid resource management mainly focuses on job scheduling with manual or semi-manual resource allocation where resources must be prepared in advance with required applications and system software. This results in non-portable applications hard-coded to a specific Grid environment. Furthermore, it undermines the possible availability of a wide range of logical resources such as software services and tools, thus restricting Grid users to a limited set of physical resources. However, the importance of logical resources is accentuating with the evolution of Grid applications. Some advanced Grid programming environments allow application developers to specify Grid application components at a higher level of abstraction, which then require a dynamic mapping between high-level resource descriptions and actual deployments.

Advance reservation has been largely ignored in the Grid. A Grid application is mostly distributed in nature and runs in a workflow paradigm where dynamic mappings of application components may require advance allocation of resources so that resources could be available as the workflow execution progresses. Advance reservation improves behavioral predictability and on-demand provisioning QoS. A service-level agreement is required to ensure terms and conditions to be agreed upon during negotiation. In the Grid, advance reservation of resources is a challenging task due to the dynamic behavior of the Grid, multi-constrained contending applications, under utilization concerns, and lack of support for agreement enforcement. The under-utilization of resources is a major issue with advance reservation. For a workflow application, resources must be available even under the worst possible conditions for a successful execution. As a result, resource allocation time has to be much longer than average execution time, resulting in computing power wastage.

Nowadays a huge collections of logical resources such as software services and tools are independently and freely available in the Grid, distributed among different physical resources. In order to utilize these logical resources a manual composition is required to build a Grid-enabled application. The manual composition is not only a time-consuming process but it also requires a domain-specific knowledge. Thus, a large collection of logical resources are left unused.

This monograph renders boundaries of Grid resource management, identifies research challenges, proposes new solutions and introduces new techniques with implementations in the form of a Grid resource management system called *GridARM*. GridARM is part of Askalon—a Grid application development and execution environment. The system is designed and developed as a scalable distributed resource manager that delivers resources on-demand, works for resource providers, and optimizes resource utilization with better load distribution and capacity-planning strategies. Optimal load distribution among resources is done according to their proportional share in the Grid.

Novel techniques are introduced for on-demand provisioning, advance reservation, and capacity planning. On-demand provisioning becomes possible with automatic deployment, resource synthesis, and advance reservation. In contrast to existing resource managers, *GridARM* covers logical resources as well by enabling advanced Grid programming environments to specify application components at higher level of abstractions and mapping them to actual deployments dynamically. Henceforth, it simplifies abstract descriptions and separates them from concrete deployments. We further exploit Semantic Web technologies for the Grid to specify explicit definitions and unambiguous machine-interpretable resource descriptions for intelligent resource matching and automatic resource synthesis capabilities.

We introduce a smart negotiation protocol to make optimal resource allocation for a better resource utilization. Advance reservation is supported with a practical solution for agreement enforcement. In order to address underutilization concerns, we introduce capacity planning with multi-constrained optimized resource allocations. We model resource allocation as an *on-line strip packing problem* and introduce a new mechanism that optimizes resource utilization and other QoS parameters while generating contention-free solutions.

Furthermore, this monograph introduces a new mechanism for automatic synthesis of logical resources by applying ontology rules. The synthesis process generates new compound resources that can be provisioned as new or alternative options for negotiation and advance reservation. This is a major advantage compared to other approaches that only focus on resource matching. The newly generated compound activities provide aggregated capabilities that otherwise may not be possible; this leads toward an automatic generation of complex workflow applications. In addition, we introduce semantics in capacity planning for improving optimization in resource allocation.

The newly introduced techniques and proposed solutions are already integrated in Askalon Grid runtime environment and deployed in the Austrian Grid. The book also demonstrates the effectiveness of the system through well-performed experiments.

June 2009

Mumtaz Siddiqui
Thomas Fahringer

Contents

1

Introduction

A world-wide communication system was developed in the early 1970*s* to make electronic messaging possible among scientists for the sharing of their research findings with each other. However, the communication protocols were complex and their applications were non-intuitive. Later in the early 1990*s*, world wide web was created to share not only information but (raw-)data as well in an intuitive way of publishing and sharing instead of messaging. The invention of world wide web has ubiquitized the Internet and enabled scientists to access (raw-)data published by fellow scientists for their own analysis. As a next step, an environment was required to process the huge amounts of data using computers in an efficient way; an environment, in which users and scientists could have communicated and shared not only information and data but resources as well, such as computers and scientific instruments.

In the last decade, a major step has been taken towards building an economical computing infrastructure composed of commodity computers and network components. As a consequence an effective and efficient utilization of widely distributed resources to fulfill the needs of a range of applications [144] is established. The low-cost computing resources for instantaneous sharing and processing of ideas, knowledge, and skills has made the collaborative work dramatically possible that paved a path towards rapid evolution of distributed computing. As soon as computers are interconnected and communicating, we have a distributed system; this raised issues in designing, building and deploying distributed computer systems which have been explored over many years [144].

Distributed computing is a way of scaling computation so that different parts of a program can run simultaneously on multiple computers interconnected over the Internet. The idea is further evolved to harness computing power of *'idle computers'* available across the world under different administrative domains. This is to eliminate the requirements of having expensive dedicated resources and to get aggregated power of inexpensive resources that collectively turnover the power of expensive supercomputers.

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 3–16, 2010.

The idea of CPU cycle scavenging evolved in the early history of distributed computing, however it has been popularized with the introduction of *volunteer computing*. Volunteer computing became widely known in 1999 through SETI@home [127] that harnesses the free computing cycles of idle personal computers interconnected and distributed all over the world. Distributed computing evolved in the same timeframe to target not only personal computers but all kind of heterogeneous computing resources distributed under multiple trust domains. This is referred to as *meta-computing* or *the Grid*.

The term *Grid* was coined in the early 1990*s* as a metaphor for introducing computing capability as a utility and making it as easy to access as an electric power Grid. Analogous to power Grid the computing Grid is perceived as a Grid of distributed computers that provides a transparent and pervasive computing infrastructure in which computing capability is delivered over the Internet and can be used as a utility.

Ian Foster, Carl Kesselman and Steve Tuecke, the so-called fathers of *the Grid* [22, 3], lead the efforts to provide *management* of computing resources for CPU cycle scavenging, distributed security, data transfer, monitoring, and provision of a Grid-enabled service development environment [76, 10]. In the beginning of 2004, developments were made towards latest standardization in *the Grid* for management of resources and configurations and mechanisms like negotiation, notification, event propagation, and information aggregation [191]. The standardization efforts are taking place under the umbrella of the Global Grid Forum (GGF) [81] that is transformed into the Open Grid Forum (OGF) [129]. The evolution of the Grid forum indicates the importance of the Grid, requirements for standardization, and above all the interest of the international research community and the industry for a reliable, robust and pervasive Grid infrastructure.

The Grid enables flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources [144, 74]. It is a distributed computational environment that is composed of non-dedicated diverse resources spanning the entire Internet under multiple trust domains. It intends to make high performance computational resources available on-demand to anyone from anywhere at anytime.

An effective *Resource Management* is required for the provisioning and sharing of resources without undermining the autonomy of their environments and independence of geographical locations. In contrast to the resource management in conventional systems, the *resource management in the Grid* has to balance global resource sharing with local autonomy by dealing with issues of heterogeneity of resources and multiple administrative domains. This emphasizes the importance of information aspects, essential for resource description, discovery, selection and brokerage.

Apart from matchmaking of physical resources, Grid resource management has to deal also with on-demand provisioning of logical resources, such as software components. On-demand provisioning of resources needs to address issues of resource matching and selection, automatic deployment, and

capacity planning. Existing resource managers concentrate mostly on physical resources. However, some advanced Grid programming environments allow application developers to specify Grid application components at a higher level of abstraction which then require an effective mapping between high level description and concrete deployments. For this purpose, a mechanism is required that can be used to enable and build Grid applications. Resource allocation with advance reservation is an important aspect that plays a key role in enabling the Grid resource management to deliver resources on-demand with a significantly improved quality-of-service (QoS).

Recently, Grid researchers have begun to take a step further, from information to knowledge [144]. To strengthen this vision, we introduce semantics in the Grid using state-of-the-art Semantic Web technologies. It simplifies descriptions and representations of both physical and logical resources. Semantics gives the Grid resources clear and machine interpretable meanings so that they can be registered and discovered unambiguously and as a result become available on-demand. This is a step towards enablement of potential applications for *the Grid* and shielding the Grid middleware complexities and low-level details from the Grid users and application developers.

1.1 Motivation

To benefit from the Grid, an application needs to be compute and/or data-intensive, and should be decomposable into smaller problems. The distinguishing aspect of *the Grid* is provisioning of an abstraction layer over resources that allows homogeneous access to and better usability of heterogeneous distributed resources. Access is provided through uniform operating and management system. Computational performance is gained at the cost of increased latency due to the additional abstraction layer. The uniform interfacing mechanism provided by the abstraction layer refers to as *virtualization* and the logical grouping and sharing arrangement in the Grid is referred to as *Virtual Organization (VO)* [74]. Based on such realizations the Grid can be described as:

> a virtualized distributed computing infrastructure in which the computing power is transparently delivered on-demand using open standards in a coordinated and shared way by aggregating capabilities of low-cost off-the-shelf heterogeneous computing devices dispersed under multiple trust domains.

However, *the Grid* has a long way to go in order to qualify as a virtualized single computer delivering computing power as a utility. Transparency and pervasiveness is possible only by providing a virtualized, uniform and transient access to the heterogeneous resources, whereas intuitive usability of the Grid is possible only through effective *resource management*.

The main focus of our research described in this book is the *Grid resource management* enabled to deliver resources on-demand and to ensure and maintain a certain level of service quality. Both physical resources such as computers and logical resources such as software components are covered. In this book, we investigate techniques and strategies for automatic resource discovery and selection, automatic deployment of legacy scientific applications, and Grid capacity planning and management. We try to answer which techniques and strategies are better and how the provisioning quality of service can be improved. While investigating on-demand provisioning and capacity planning for the Grid, we also identify how existing Grid and Web technologies can be exploited for the enforcement of agreements, improvement in resource utilization, and betterment in on-demand resource provisioning. One of the main goals was to develop a smart and robust resource management for the Grid that delivers Grid resources on-demand with improved quality of service, better capacity planning, and the Grid enablement for potential applications.

Some of the challenges and motivations that have been driving the research about resource management in the Grid are given below. These challenges are addressed with innovations in our research work.

1.1.1 Collaboration Instead of Isolation

Many scientific and engineering problems today require widely dispersed resources to be operated and uniformly accessed as systems. Networking, distributed computing, and parallel computational research have matured to make it possible for distributed systems to support high-performance applications. However, resources are dispersed, connectivity among them is fluctuating, and dedicated access is impossible. This adds a real challenge to virtualized access that is important for the manageability of resources in a Grid.

Widely distributed resources in the Grid are heterogeneous, shared, and federated. Therefore it is important to address issues such as connectivity, performance, interoperability and manageability of these resources. A uniform, location independent, and transient access to these resources is the vision of the Grid. Resources like scientific instruments that facilitate the solution of large-scale, complex, multi-institutional, and multi-disciplinary data and computational problems must be accessible through problem solving environments that are appropriate for the target user community.

1.1.2 Discovery and Selection

In dynamic environments such as *the Grid*, where resources may join and leave at any time, discovery and selection plays a key role in an effective Grid-level resource management. In the currently available Grid operating environments [10, 70, 68], resources are published in a hierarchical information systems where published information are propagated from resources to an

index service [39]. The index service is an aggregator and hub of the collected information and thus becomes a bottleneck. Such a discovery system should be reliable and fault-tolerant.

Once resources have been discovered, the resource that fits best to the user goal has to be selected. It is important to make resources available on-demand no matter where they reside or who owns them. It is also very crucial that while making a resource selection for a resource requester, the utility of one stake holder should not be compromised at the cost of other stake holder. That means, the load distribution should not become unfair just to satisfy a certain user, or resource utilization should not be maximized at the cost of user goals.

Currently, resource selection is manual or semi-manual in the Grid. For the evolving Grid infrastructure in which number of resources is increasing along with number of competing applications, provision of an automatic resource selection and brokerage is required.

1.1.3 Lifecycle Management

The Grid can also be visioned as a pool of idle resources. These resources can be re-purposed by re-configuring their environments. Automatic lifecycle management can play a significant role in re-purposing these resources on-demand. This mechanism not only improves resource utilization but also enables a resource manager to generate more options to offer on request.

However, lifecycle management of Grid resources is hard as resources are distributed under different administrative domains, their availability is unpredictable, and the resource requesters and providers have conflicting goals and policies. In such an environment, dealing with lifecycle management is a real challenge.

The modernization and virtualization of resources are a few encouraging and motivating factors. Physical resources possess great potential to be managed throughout their lifespan using virtualization technologies. Similarly, lifecycle management of logical resources is also possible by considering automatic deployment, undeployment, exposure and/or shielding from Grid users.

1.1.4 On-Demand Provisioning

On-demand provisioning has been gaining momentum in various fields including resource management for the Grid. Gaining control over the lifecycle management of underlying resources can help to handle dynamic environment resulting in improved delivering of resources on-demand.

The Grid environment is dynamic in which resources join and leave the Grid and prediction about their behavior is very hard. However, by applying sophisticated monitoring and prediction means it might be possible to foresee the expected demand and to supply the underlying resources. For instance, it

is more likely that during peak times less resources become available. Similarly, during weekends and holidays more resources become available than an average. In contrast, during the working hours, demand increases than supply.

In the light of this dynamic scenario, enabling on-demand delivery of resources at runtime is more economical, efficient, and transparent than hard-coding a set of resources statically for an application before runtime.

1.1.5 Role of Planning

On-demand provisioning of resources can be counter productive without proper capacity planning and management. As mentioned earlier, it is possible to deliver resources on-demand. One simple form of planning could be to allocate resources with low utilization to Grid applications.

However, resources are scarce and applications compete for these resources. In this context, planning becomes a forward looking process that can be achieved with proper capacity planning of Grid resources expected to be available sometime in the future. This problem has similarities to the yield management [123] that is offered, for example, in airline and hotel reservation systems. In the yield management, resource leasing/selling is planned well in time so that as a result utilization of available resources is optimized and profit is maximized, and resource capacity does not wasted with passage of time.

Similarly, resource allocation needs to be planned in such a way that resource utilization is optimized by allocating either under-utilized resources or by making allocation in future when resources are expected to be available. This approach improves provisioning quality by offering later delivery of resources instead of not delivering at all.

1.1.6 Service-Level Agreement

Looking towards the future is important for planning, however, protecting resources for potential applications and clients, complements the capacity planning in the Grid. It is important to protect a resource offered in future. This emphasizes the need for service-level agreement (SLA).

However, in the dynamic Grid environment, an agreement enforcement is a real challenge. It is very hard for the resource management system to make a promise that an unpredictable resource will be available in future. To overcome this problem, a solution is required that works in dynamic and heterogeneous environments. This book addresses this challenge along with capacity planning.

An important aspect is fairness in resource allocation. Protecting an appropriate share for a class of users, so that other users could not consume the entire capacity available in the Grid, could lead towards fair distribution. This kind of protection is possible with planning using allocation with advance reservation.

1.1.7 Optimized Resource Allocation

Allocation of scarce resources to contending applications is an NP-complete problem [152, 38] and legacy heuristics are not sufficient to provide an efficient and optimal solution because of the dynamic nature of the Grid. Resource allocation with advance reservation may result in under-utilization of resource capabilities. In order to overcome this concern, allocation needs to be made with optimization by using appropriate capacity planning strategies.

Capacity planning and advance reservation is therefore of paramount importance for the Grid in order to agree on negotiated SLAs. Proper planning ensures a stable and powerful Grid that can grow to meet future needs and preserve client-centric SLAs while optimizing often contradicting requesters and providers goals. Optimization of QoS parameters is an increasingly important approach to manage Grid resource capacity as the sophisticated and distributed applications are evolved.

Since a resource manager works for resource providers, it is important for a resource manager to protect resource provider's interests while fulfilling requirements of clients. That means new algorithms are required to make optimal resource allocations without compromising client goals while dealing with the dynamic nature of the Grid environment.

1.1.8 Synthesis and Aggregation

On-demand synthesis of application components can be used in automatic workflow composition and in improving quality of the resource provisioning. However, synthesis of activities has been largely ignored due to the limited expressiveness of the representation of resource capabilities and the lack of adapted resource management means to take advantage of such resource synthesis. The synthesis of resources combines multiple primitive resources to form new compound resources.

The synthesized resources can be provisioned as new or alternative options for negotiation as well as advance reservation. Furthermore, the new synthesized resources can provide aggregated capabilities that otherwise may not be possible, leading towards an automatic generation of large-scale application workflows as a virtually single compound activity. This is a major advantage compared to existing approaches that only focus on resource matching and brokerage.

1.1.9 Grid Enablement

To treat Grid resources as commodities, an application should be Grid-enabled so that it can exploit the environment of the Grid in order to get benefit from it.

An application refers to as Grid-enabled if it is ubiquitous, resource aware, and adaptive: the ubiquity enables an application to interface to the system at

any point and leverage whatever is available at that point in time, the resource awareness enables an application to be capable of managing heterogeneity of available resources, and finally, the adaptability enables an applications to tailor its behavior dynamically so that it gets maximum performance benefits from services and resources at hand [95].

However, enabling the legacy scientific applications for the Grid is very hard since it requires re-implementation of applications according to the Grid environment. Re-implementation requires vigorous testing and quality assurance. On the other hand, a fully Grid enabled application cannot be executed in a non-Grid environment without modification.

1.1.10 Portability

Grid applications are portable if they can be mapped dynamically to the Grid environment unless the architecture of the target machine is inconsistent with the application's supported architecture. Dynamic mapping is possible by separating abstract or functional description of application components from concrete deployment descriptions and then providing a mechanism to map abstract descriptions to concrete deployments.

However, clear separation between abstractions and concrete deployment representations is not an easy task especially for legacy applications. Legacy applications are unobtrusively defined in a way that their abstract descriptions or interface definitions are not well defined.

1.1.11 Semantics in the Grid

Semantic technologies like ontologies provide vocabularies with explicitly defined, unambiguously understandable and automatically machine-interpretable meanings that enable automatic resource brokerage. As a resource description model, it is proposed to replace the classic attribute-based symmetric resource description model with an extensible ontology-based asymmetric model. The proposed model provides foundation to a flexible and extensible discovery and resource matching mechanism.

Different types of resources can provide similar capabilities but with varying degrees of QoS. This highlights that resource capabilities are required to be presented in such a way that consumers can easily discover resources matching their requirements, by following some sophisticated patterns of resource discovery, matching, and negotiation.

A powerful discovery mechanism can be built based on expressive description mechanisms. This means, it is necessary to explicitly, precisely, and unambiguously describe Grid resources and specify various constraints over resource descriptions. The description should be automatically interpretable and understandable by machines. Major contributions of semantics in the Grid are possible in the area of on-demand provisioning, optimized resource allocation and synthesis of resources leading towards introduction of new capabilities.

1.2 Research Goals

Under the motivations outlined in the previous section, in this book we address the mentioned problems and provide a novel resource management system called *GridARM* [153, 154] that is developed as part of the Askalon Grid application development and execution environment [63, 62].

Our goal is to explore theoretical and practical aspects of resource management for the Grid in order to provide automatic resource matching and selection, automatic deployment of logical resources i.e. applications and software components, advance reservation and co-allocation, capacity management and planning for resources currently available and expected to be available in future. Additional goals include adaptation of state-of-the-art Grid and Web technologies and the proposed service-oriented architecture, introduction of semantics in the Grid, and last but not least to take a step towards shielding the Grid complexities and enabling it for potential applications. The major goals of this book are described as follows:

1.2.1 Automatic Resource Brokerage

We propose an automatic resource selection and brokerage mechanism as a part of the *resource manager*. The main task of the resource manager is to optimize resource allocations for all contending applications. Most of the currently available Grid operating environments [10, 70, 68] provide tools and services to support resource brokerage. Nevertheless, resource brokerage in existing systems is manual. The Grid environment is getting more and more mature and its applications contending for scarce resources are evolving. This highlights the need for a mechanism for automatic resource brokerage that does resource matching with user goals while distributing resources in a fair and optimal manner. This book intends to provide a resource manager that is capable to do automatic resource brokerage by allocating resources based on some criteria such as fairness or optimization without undermining interests of requesters/providers. Some of the important features of the proposed resource brokerage are:

- the resource management system is to be **distributed in a service-oriented fashion** and multiple instances of the resource manager coordinate with each other in order to share their underlying resources in a superpeer model [169]-based distributed infrastructure;
- the resources, matching the user requirements, should be automatically selectable and a candidate set needs to be generated in an order according to a user-defined **candidate selection criteria**;
- the final selection should be made in accordance with a fairness policy; resources need to be **allocated according to the proportional share** of resources in the Grid;

- the resource manager can be **customizable** in order to set up a dedicated experimental environment and a coarse-grained access control policy for underlying resources.

1.2.2 Dynamic Registration and Automatic Deployment

A key concerns of resource management is to implement an effective Grid middleware that shields application developers from low level details. Existing resource managers concentrate mostly on physical resources. However, some advanced Grid programming environments allow application developers to specify Grid application components at high level of abstraction which then requires an effective mapping between high level application descriptions and actual deployed software components. In this book we introduce a framework that provides dynamic registration, automatic deployment and on-demand provision of application components that can be used to build Grid applications. This framework is called *GLARE* and is implemented as an extension of *GridARM*. Here are the main features of *GLARE* which

- separates and simplifies abstract and concrete descriptions and representations of resources so that they can be advertised and located unambiguously to be **delivered on-demand**;
- provides **automatic deployment** of applications on the selected node (Definition 11), the deployment procedure can be provided as part of the abstract descriptions of application components by the providers, and executed on the target node by *GLARE*;
- **un-deploys automatically** once applications are no longer required;
- enables **registration and un-registration** of deployed applications in order to expose or hide them to the Grid users.

1.2.3 Advance Reservation and Co-allocation

Advance reservation plays a significant role in providing a smart and robust resource management for the Grid in order to have a better control over Grid resources for capacity planning, fair load distribution, and optimal resource utilization.

In this book we propose a mechanism for advance reservation of Grid resources with better planning for resource allocation and a practical solution for enforcement of agreements. The distributed resource allocation system enables a client to negotiate for required resources in order to reach on a better compromise between application requirements and resource capabilities. This mechanism contributes not only for better planning but also for improvement in predictability. A set of features of distributed advance reservation system has been identified to be implemented as part of the *GridARM* which

- introduces different algorithms to perform advance reservation by different types of **allocation strategies** including **fairshare** and **optimization** in resource utilization;
- introduces a 3-layered **negotiation protocol** between resource requesters and providers in order to reach and seal an **agreement**;
- introduces a **practical solution** for the **agreement enforcement** based on the off-the-shelf Grid technologies;
- introduces the idea of **open reservations** in order to deal with the dynamic behavior of the Grid. An open reservation is a promise by the system that a resource will be available at sometime in the future but the resource binding is deferred to be decided later at runtime. This scenario represents a priority provision; the next available resource that fulfills user goals is allocated at runtime;
- demonstrates the effectiveness of advance reservation for planning and predictability.

1.2.4 Capacity Management and Planning

In this book we introduce a new mechanism for capacity management and planning that exploits advance reservation of Grid resources. In *the Grid*, capacity planning and management has been ignored due to the dynamic Grid behavior, multi-constrained contending applications, lack of support for advance reservation and its associated challenges like under utilization and agreement enforcement concerns.

These issues force a Grid resource manager to allocate resources at runtime with reduced quality of service (QoS). The proposed Grid capacity planning and management is performed with the help of advance reservation and **multi-constrained allocation optimization**. It models resource allocation as an **on-line strip packing** problem and introduces a new mechanism that optimizes resource utilization and other QoS parameters while generating contention-free solutions. Our proposed solution

- provides a **forward looking process** in which allocations are made along a planning horizon;
- exploits advance reservation for **optimized resource allocation** with service-level agreement (SLA);
- provides a mechanism to plugin different **allocation offer generation** algorithms;
- provides a set of offer generation algorithms that can be used according to the policy of resource providers;
- generates multiple options to be offered to the client. The options are generated as **alternative offers** based on **multi-constrained optimization** of **resource utilization**;
- generates allocation offers in such a way that resource capacity is optimally utilized and capacity **wastage is minimized**.

1.2.5 Semantics in the Grid

We introduce an ontology-based resource description, discovery and selection mechanism. For the resource description model the book proposes to replace the classic attribute-based symmetric resource description model with an extensible ontology-based asymmetric model. This model provides foundation to a flexible and extensible discovery and correlation mechanism.

Furthermore, this book exploits semantics in the Grid and introduces automatic synthesis of resources and software components in the Grid by applying ontology rules. Rule-based synthesis combines multiple primitive resources to form new compound resources. The main goals of the book in the context of semantics are the following:

- **asymmetric resource description** with ontologies so that resource requesters and providers don't have to agree on certain terms and their agreed upon values;
- **subsumption**-based resource selection that allows to propose alternative options if exact match does not exist;
- **synthesis** generates multiple resources that can be provisioned as new or alternative options; the newly generated synthesized resources provide aggregated capabilities that otherwise may not be possible;
- synthesis enables **automatic workflow generation**.

1.2.6 Standard Adaptation

GridARM follows the paradigm of service-oriented architecture (SOA) in which services are loosely coupled and coordinate with each other [74, 76]. Following the SOA vision, the Open Grid Forum is actively working on the standardization of various aspect of the Grid. *GridARM* intends to adapt proposed standards in the area of:

- Grid Resource Allocation Agreement Protocol (GRAAP) that proposes WS-Agreement specification [87], we use this proposed standard for negotiation and agreement management;
- Job Specification Description Language (JSDL) [100] that supports a rich set of constructs for constraints specification. We use JSDL to specify multiple constraints;
- Configuration Description, Deployment, and Lifecycle Management (CDDLM) [31] is a standard for the management, deployment, and configuration of Grid resources. We propose to use CDDLM in configuration and lifecycle management of resources.

Last but not least, this book is a step towards an invisible Grid. A smart resource management enables Grid middleware infrastructure to deliver seamless resource management capabilities by shielding the Grid users and the

application developers from low level details and the middleware complexities. *The Grid* has been visioned as a virtualized single computer system, and one of the goals of resource management for this virtualized computers is to make it work as if the entire Grid is a single computer that transparently delivers computing capabilities to its clients. Following are the main features that contributed towards invisible Grid:

- **automatic resource brokerage** that performs resource matching with users requirements;
- Grid-independent **abstract description** of resources that are mapped to concrete deployments dynamically at runtime, thus hides low-level details and complexities of the Grid middleware.

We address the major challenges described in Section 1.1 in a systematic way in order to achieve the goals described in Section 1.2. This book is organized in five parts as described in the following sub-section.

1.3 Organization

This book is subdivided in five parts and appendices, each part consists of two chapters with the exception of the last part that concludes the book with a single chapter.

1.3.1 Part 1: Overview

The first part provides an overview of the book that includes introduction (this chapter) and model (next chapter). Chapter 2 describes model of the Grid, its components, characteristics, middleware by covering both Grid operating and runtime environments. Furthermore, it describes resource management model by defining various concepts and terminologies used in the following chapters. This chapter also gives overview of a Grid operating environment called Globus Toolkit [10], a Grid runtime environment called Askalon [61], Semantic Grid, and various manageability models.

1.3.2 Part 2: Brokerage

This part of the book covers resource brokerage. Chapter 3 describes *GridARM* architecture in general and discovery, selection, and brokerage of physical resources in particular. It describes in detail, resource discovery and selection mechanisms, candidate set generation, and a mechanism for proportional share-based optimal load distribution. The chapter also proposes a super-peer model-based distributed decentralized infrastructure. Finally, this chapter demonstrates and analyzes results.

Chapter 4 covers resource brokerage of logical resources i.e. applications and application components. It introduces a Grid-level application registration, deployment, and provisioning framework in which application components can be registered dynamically, deployed automatically, and provisioned on-demand. The chapter also demonstrates effectiveness of our approach with experiments.

1.3.3 Part 3: Planning

Part three covers allocation and capacity planning in the Grid. Chapter 5 covers resource allocation with negotiation-based advance reservation and a practical solution for agreement enforcement. It defines various concepts related to allocation, advance reservation, and agreement that are used for negotiation and contract representations.

Chapter 6 describes capacity planning and management for optimization of multi-constrained allocations. It proposes a 3-layer negotiation protocol and a new algorithm for allocation offer generation that generates multiple allocation offers in order to improve resource utilization. Finally, the chapter demonstrates effectiveness of the capacity planning approach with experiments.

1.3.4 Part 4: Semantics

Chapter 7 gives an overview of the semantic Grid technologies and proposes an ontology-based semantics description and matching mechanism for Grid resources. It proposes Grid resource ontologies in the form of ontological classes and concepts for describing resources so that they can be unambiguously interpreted and automatically understood by the management and brokerage system.

Chapter 8 introduces a mechanism for automatic synthesis of resources by applying ontology rules. In particular, it covers application components. Rule-based synthesis combines multiple primitive resources to form new compound resources. The newly generated compound resources provide aggregated capabilities that otherwise may not be possible. The chapter also demonstrates advantages of semantic-based automatic synthesis of Grid activities.

1.3.5 Part 5: Conclusion

The final part concludes the book by highlighting contributions and future research foci.

1.3.6 Appendices

Appendix A gives a table of all notations and mathematical symbols we have used in this book.

2

Model

This chapter provides an overview of *the Grid*, its components and discourses with special focus on resource management. It defines important aspects and the technological and architectural advances that have led to the evolution of the Grid thereby setting the foundation for this book.

2.1 The Grid

Initially, the Grid computing infrastructure was visioned as a *metacomputer* formed by connecting supercomputers that could be remotely controlled and managed. Nevertheless, the idea evolved to cover not only supercomputers but almost all kind of computing devices ranging from commodity computers to scientific instruments.

In the context of distributed computing, the term *'Grid'* was coined in the early 1990s as a metaphor for introducing computing as a utility and making it as easy to access as an electric power Grid. Analogous to power Grid the computing Grid is perceived as a Grid of distributed computing power generators i.e. computers. *The Grid* is a transparent and pervasive computing infrastructure in which computing power can be used as a utility that is to be delivered on the Internet. It enables resource sharing and coordinated problem-solving across computers and humans in a distributed and heterogeneous environment [144].

Today there are many definitions that define the Grid from different perspectives [75, 47, 72, 95], for instance as a conceptual framework like the World Wide Web (WWW), as an utility infrastructure like power grid, or as a single virtual computer. In general the idea behind the Grid is to solve challenging problems by using low-cost off-the-shelf computing devices. The solutions to these problems is otherwise considered very hard and expensive if not impossible. In this perspective, the Grid is an emerging computing model that enables virtualization, sharing and transparent provisioning of heterogeneous resources distributed across multiple administrative domains using open

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 17–44, 2010.

standards to model a virtual computer architecture. It distributes computing power across a networked infrastructure to offer a better quality of service (QoS).

> *The Grid* is a virtualized distributed computing infrastructure in which computing power is transparently delivered on-demand using open standards in a coordinated and shared way by aggregating capabilities of low-cost off-the-shelf heterogeneous computing devices dispersed across multiple trust domains. Formally, the Grid $\mathcal{G}$ is an aggregation of heterogeneous resources and consists of a set of nodes such that $\mathcal{G} = \sum_{i=1}^{n} \mathtt{g}_i | n \in \mathbb{N}$. Each *node* $\mathtt{g}_i \in \mathcal{G}$ is a computer as described in Section 2.2.3 that is logically connected with other geographically dispersed nodes $\in \mathcal{G}$ and they may join or leave $\mathcal{G}$ at any time.

A major discourse of the Grid is to harness unused power of idle computers in the Internet for solving problems too intensive for any stand-alone machine. The unused computing power is wasted otherwise. A practical way to use this computing power is to perform computations on idle computers by remote program execution. However, this is very hard to achieve as these idle computers can be widely distributed across the globe while being managed by different individuals or organizations, that is, they are under multiple administrative domains. Another issue deals with the problem that computers perform actual computations that might not be entirely trustworthy. Thus the model of the system must introduce measures to prevent malfunctions or malicious participants from producing false, misleading, or erroneous results, and from using the system as an attack vector.

On of the initial works in the dimension of harnessing unused idle cycles is the SETI (Search for Extraterrestrial Intelligence)@Home [127] project, in which personal computers distributed across the world donate unused processing cycles to help finding signs of extraterrestrial life by analyzing signals coming from outer space. The project relies on individual volunteers to allow the project to harness the unused processing power of the volunteer's computer. This method saves both money and resources for the project.

Since computers are distributed without any central control, it is hard to guarantee that the state of the computers will not change. It might be possible that some computers leave whereas others join the Grid. The impact of trust and availability on performance and development difficulty can influence the selection of a specific computer.

2.1.1 Characteristics

Computers distributed across multiple administrative domains are usually heterogeneous with different operating systems, hardware architectures and

several languages. This underlines the need for standardization and virtualization. A generalized Grid can be defined in terms of four characterizing aspects that are *heterogeneity*, *adaptability*, *scalability* and *autonomy* [144, 26].

- **Heterogeneity:** The Grid is a collection of heterogeneous resources with heterogeneous architectures, platforms, operating environments and other technological aspects.
- **Adaptability:** The Grid environment is dynamic by nature. Resources join and leave the Grid at any time without intimation. In such an environment, the probability of resource failures is high and the operating environment (middleware) has to deal with such situations.
- **Scalability:** Since resources join and leave the Grid, it grows and shrinks and may consists of upto millions of resources. Further, Grid applications are also increasing in number and diversity. This emphasize the requirement of highly scalable middleware and Grid-enabled applications.
- **Autonomy:** The second prominent characteristic of the Grid is that the underlying resources are autonomous and they are administered under different trust domains. It is required that the autonomy of the resources is not compromised while being used in the Grid environment.

2.1.2 Layers

The major components of the Grid as depicted in Figure 2.1 are infrastructure fabric, middleware, and applications.

- **Infrastructure Fabric:** The underlying physical resources distributed across the globe and connected through high-speed Internet form the base of the Grid. These resources include but are not limited to computers, clusters, storage and network devices and scientific instruments. The front-end resources, which are directly accessible, are equipped with low-level *middleware*-aware local resource managers such as SGE (Sun Grid Engine), PBS (Portable batch System) and LSF (Load Sharing Facility) etc.
- **Middleware:** The middleware is the brain of the Grid and offers important generic services required for a functional Grid. Various middlewares have been developed to allow the scientific and commercial community to harness the computing power and form a Grid [10, 68, 91, 96, 27, 61]. The middleware can be categorized in Grid operating environment and Grid runtime environment.
  Core Middleware: The core middleware, also called *Grid operating environment* constitutes lower part of the middleware. It provides a fundamental framework for interaction with physical infrastructure fabric. This includes security infrastructure, data transferring facility, local execution management etc. This layer does not see structure of a complete Grid application but only it components without their relation to the full application.

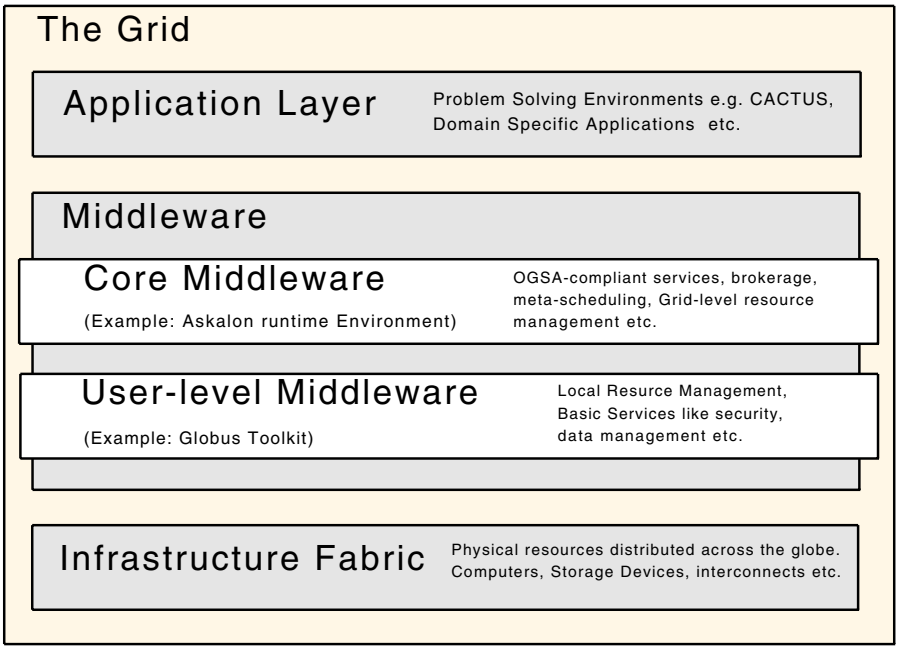


Fig. 2.1. The Grid Layers.

Examples of such kind of low-level middleware includes Globus [10], gLite [68], Unicore [70] etc.

User-level Middleware: The user-level middleware, also called *Grid runtime environment*, constitutes the upper part of the middleware and provides high-level OGSA-compliant services for the Grid applications. They include job scheduling, job enactment, monitoring, meta-scheduling, resource management etc. In contrast to the operating environment, it usually is aware of an entire Grid application (e.g. workflows). It understands all components of a workflow and their inter-dependencies.

Examples include Askalon [61], GridBus [27], myGrid [96], Pegasus [58], P-Grade [131], Kepler [4], ICENI [112] etc. The focus of this book is the resource management in Askalon (see Section 2.4) Grid runtime environment.

Introduction of layered middleware is intended to shield low-level Grid complexities from its clients. The type of the middleware contribute in defining architectures of the Grid.

- **Applications:** The Grid applications constitute the high-level layer of the Grid and are developed with or without the Grid in mind. The *Grid-enabled applications* are those which may exploit the Grid to its full potential. For instance, *Cactus* [1] is a Grid-enabled open-source application that

[1] http://cactuscode.org

provides a problem solving environment designed for scientists and engineers. Its modular structure easily enables parallel computation across different resources in the Grid. Montage [28] is used to generate complex astronomy workflows. LIGO Data Grid [6] makes use of workflow technologies for *gravitational wave* data analysis. The workflows for e-science [62] includes several interesting examples: workflows in *Pulsar Astronomy* operate on the output of the signal of a radio telescope to detect the characteristic signals of pulsars. These signals are much weaker and computation-intensive distributed algorithms are applied using the Grid to extensify these signals. SCEC CyberShake Workflows is used in automating probabilistic seismic hazard analysis calculations. The *biomedical informatics* research network works for the *telescience project* and *ecological niche modeling* using *Kepler* [4] examines the details of a specific analysis within Kepler to illustrate the challenges, workflow solutions, and future needs of *biodiversity* analysis. Applications developed for clusters or suppercomputers can be ported to the Grid.

2.1.3 Architectures

The Grid has been developed with different perspectives, including *Computational Grid*, *Data Grid*, and *Knowledge Grid*. We define it as follows:

Definition 1. *A* Computational Grid $\mathcal{G}^c \subseteq \mathcal{G}$ *is a form of the Grid in which each node* $g_i \in \mathcal{G}^c$ *provides computational capabilities. It was originally defined as a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [72]. Over time, the Grid concept has been refined and better formulated, e.g., as a persistent infrastructure that supports computation-intensive and data-intensive collaborative activities that spawn across multiple Virtual Organizations (VO) [136].*

The underlying infrastructure for a computational Grid is the Internet that is a worldwide network of computer-networks. The publicly accessible interconnected computers in the Internet transmit data by packet switching using the standard Internet Protocol (IP). Millions of smaller public/private networks are participating in the Internet, which together provide various kinds of information and services, such as electronic mail, file transfer, and the interlinked Web pages and other documents of the World Wide Web (WWW).

An important class of data is the raw and meta data that is generated and gathered by the scientific instruments and experimentation, business, games and e-learning applications, and monitoring infrastructures. This kind of raw and/or meta data is processed by a data-centric Grid called the *Data Grid*. Along with business applications and distributed games, scientific applications, such as Montage [28], Wien2k [21], CERN's HEP applications [135] and Invmod [138] use and produce a huge amount of data (terabytes or petabytes)

distributed across laboratories. The size and collections of data is growing rapidly that requires a distributed infrastructure that is scalable and better organize and process huge amounts of distributed data. Hence the need of the data Grid.

Definition 2. *A* Data Grid $\mathcal{G}^d \subseteq \mathcal{G}$ *is a form of the Grid in which each node* $g_i \in \mathcal{G}^d$ *(See Section 2.2.3) possesses data processing and/or storage capabilities and optionally with computational capabilities. It deals with controlled sharing and management of huge amounts of data distributed across organizations and databases. It provides an intensive computation and analysis of shared data. The* Data Grid *is mostly integrated with the* Computational Grid *(Definition 1) that process the data.*

Several projects have been started to build large scale *Data Grid*. These projects include Southern California Earthquake Center (SCEC) [120], Biomedical Informatics Research Network (BIRN) [20], Real-time Observatories, and Applications, and Data management Network (ROADNet) [141]. SDSC Storage Resource Broker [165] provides an *operating environment* for the *Data Grid*. The Linked Environments for Atmospheric Discovery (LEAD) [104] provides forecast models and analysis and visualization tools for interactively and dynamically explore the meteorological data. It provides a convenient access point for all the necessary resources including the high-performance computing systems.

In many scientific and commercial applications, it is necessary to perform the analysis of huge data sets, maintained over geographically distributed nodes $\in \mathcal{G}^d$, by using the computational power of distributed computers $\in \mathcal{G}^c$. Discovering required information, useful patterns, models and trends in large volumes of data employs a variety of software systems and tools (Section 2.2). This is collectively called *data mining* or *knowledge discovery* [2].

Data mining has been investigated in the domain of parallel and distributed knowledge discovery. *The Grid* may play a significant role in providing an effective computational support for data mining applications. The Grid built for this purpose is referred as *Knowledge Grid*.

The *Knowledge Grid* $\mathcal{G}^k$ has been evolved from both *Computational Grid* $\mathcal{G}^c$ and *Data Grid* $\mathcal{G}^d$. It introduces a set of new services to employ a distributed *knowledge discovery* on globally connected computers [29]. The *Knowledge Grid* enables the collaboration of scientists for mining of data stored in different research centers as well as analysts that must use a knowledge management system operating on several data warehouses located in different autonomous establishments [30].

Definition 3. *A* knowledge Grid $\mathcal{G}^k \subseteq \mathcal{G}^d$ *is a special form of* Data Grid *in which volumes of data clustered around data-centric resources* $\in \mathcal{G}^d$ *is annotated, discovered, and provisioned semantically and data sets are analyzed to find specific patterns or models by exploiting computational resources* $\in \mathcal{G}^c$.

[2] Grid Computing Lab: http://grid.deis.unical.it/kgrid

Examples of *Knowledge Grids* include Adaptive Services Grid (ASG) [2], KWfGrid [5], OntoGrid [7] and IntelliGrid [193].

2.2 Resources and Applications

The Grid consists of a set of heterogeneous *resources* $\mathcal{R}$.

Definition 4. *A* Grid resource $r \in \mathcal{R}$ *is an entity associated with a node* $g \in \mathcal{G}$ *that contributes or facilitates to contribute some* capability *to the Grid and is logically available and accessible through a* reference $ref(r)$. *A* reference $ref(r)$ *of a resource* $r \in \mathcal{R}$ *is referred to as a unique address in the Internet that can be used to access the resource* r *remotely. Examples of references include, IP address, domain name, URL (Universal Resource Locator) and URI (Universal Resource Identifier).*

The capability *is the ability of a resource* $r \in \mathcal{R}$ *to perform some (useful) actions. Generally, a* capability *is the synthesis of* expertise *and* capacity*, where* expertise *represents qualitative properties such as the* functionality, manageability *etc. and* capacity *represents quantitative properties such as number of* total processors*,* available memory *etc.*

There are different kinds of resources but at the higher level they are categorized as *logical and physical resources. Logical resource*s include software components, configurations, policy files, workflow applications along with basic building blocks of a workflow i.e. workflow tasks or activities (Section 2.2.1). The set of Grid resources $\mathcal{R}$ includes a set of physical resources $\mathcal{PR}$ and a set of logical resources $\mathcal{LR}$, that is $\mathcal{R} = \mathcal{LR} \cup \mathcal{PR}$.

Definition 5. *A* physical resource $pr \in \mathcal{PR} \subseteq \mathcal{R}$ *is a hardware device operating directly or indirectly at the* network-layer *and may be enabled with the Grid operating environment (Section 2.3). The* network-layer *is the third-lowest layer of the* OSI Reference Model*.*

A set of physical resources $\mathcal{PR}$ include computers, interconnects, storage elements, scientific instruments etc. A computational Grid $\mathcal{G}^c$ is often centered around computers with high performing computational capabilities, such as cluster of computers, parallel computers, high-end PCs etc. These resources are considered as main sources of computing power associated with the Grid. A physical resource $\mathtt{pr} \in \mathcal{PR}$ is normally accessible with the help of a resource reference $\mathtt{ref}(\mathtt{pr})$.

An *Infrastructure Fabric* is a collection of physical (hardware) resources $\mathcal{PR}$. Each resource $\mathtt{pr} \in \mathcal{PR}$ is logically connected with other resources $\in \mathcal{PR}$ mostly through high performance interconnects. Enabled with a *Grid operating environment* the resources $\in \mathcal{PR}$ dynamically join the Grid $\mathcal{G}$, contribute their capabilities to the Grid and may leave afterward.

A *Grid Application* is a software that is *Grid-enabled* and may exploit its potential. An application is said to be *Grid-enabled* when it can be executed by a Grid. Fully exploiting the grid, however, means taking advantage of the virtualized grid infrastructure to accelerate processing time or to increase collaboration [102].

A set of *logical resources* $\mathcal{LR} \subseteq \mathcal{R}$ mostly consists of software components which include legacy software programs, libraries, software components, services (e.g. Web services), workflow applications etc. Similar to physical resources, logical resource are also accessible through an addressing mechanism, for instance as Web service is accessed through Universal Resource Identifier (URI).

An application is referred to as a collection of software components called *activities*.

2.2.1 Activities

A set $\mathcal{A} \subset \mathcal{LR} \subset \mathcal{R}$ includes a collection of activities available on the Grid. An *Activity* $\mathtt{a} \in \mathcal{A}$ is a high level abstraction that refers to a single self-contained *computational task* that corresponds to an execution unit, initiated for instance by an executable program or a service deployed on a Grid node $\in \mathcal{G}$.

Definition 6. *An* Activity $\mathtt{a} \in \mathcal{A}$ *is an abstraction of a logical Grid resource that contributes some capability to the Grid. This capability is utilized or accessed directly or indirectly through well defined interfaces. It can be modelled as* $\mathtt{a} = \{\mathcal{I}_\mathtt{a}, \mathcal{O}_\mathtt{a}\}$ *where* $\mathcal{I}_\mathtt{a} \subset \mathcal{I}$ *is a set of input arguments and* $\mathcal{O}_\mathtt{a} \subset \mathcal{O}$ *is a set of output arguments that belongs to the activity* $\mathtt{a}$.

> An argument $arg \in \{\mathcal{I} \cup \mathcal{O}\}$ is referred to as a logical entity that is to be passed to an *activity* as an input or generated by an activity as an output. It may be an *integer*, *double*, *file*, *uri* etc.

Activities $\in \mathcal{A}$ are organized in abstract and concrete descriptions [158] (Chapter 4). Abstraction of a resource contributes towards virtualization whereas concretization is used to access the resources. The term *virtualization* refers to the abstraction of computer resources. It is a technique of hiding the non-standard characteristics of a resource from the standard way in which other resources or end users interact with those resources. This includes making a single physical resource appear to work as multiple logical resources or it can include making multiple physical resources appear as a single logical resource. According to this, the Grid is also visioned as a single *virtual computer* made out of multiple physical and logical resources.

An activity is further described in terms of an *activity type* and an *activity deployment*. In the Grid $\mathcal{G}$, there is a set of activity types $\mathcal{E}$ and a set of activity deployments $\mathcal{D}$ such that for each activity $\mathtt{a}_i \in \mathcal{A}$ there is an activity type $\mathtt{at}_i \in \mathcal{E}$, and for every activity there can be multiple *activity deployments*.

Definition 7. *An* activity type $\mathtt{at}_i \in \mathcal{E}$ *is referred to as a description of semantics or functional behavior of an activity* $\mathtt{a}_i \in \mathcal{A}$ *that may be used to look-up a set of* activity deployment*s. For every activity type* $\mathtt{at}_i \in \mathcal{E}$ *there is a set* $\mathcal{D}_i = \{\mathtt{ad}_{i,1}, ..., \mathtt{ad}_{i,n_i}\}$ *of* activity deployment*s that implements the functionality described by* activity type $\mathtt{at}_i$*, where* n_i *may be different for different activity types* $\in \mathcal{E}$*.*

Definition 8. *An* activity deployment $\mathtt{ad}_i \in \mathcal{D}$ *is a realization of an activity* $\mathtt{a}_i \in \mathcal{A}$ *that implements the capability described by* activity type $\mathtt{at}_i \in \mathcal{E}$*. An activity deployment is referred to as an executable program or a service and provides accessibility information such as a reference (address)* $\mathtt{ref}(\mathtt{a}_i)$ *of the activity* $\mathtt{a}_i$*.*

Following the vision of *virtualization*, a resource that delivers some capability directly to the Grid user is virtualized as a service using some open source virtualization technology. A *service* is a resource that is referred as a self-contained autonomous entity that delivers a discrete capability with a well-defined interface. According to OASIS:

> A *service* is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description [125].

In the context of the Grid, a *service* is a form of an *activity deployment* that can be defined as:

Definition 9. *A* service $\mathtt{s} \in \mathcal{D}$ *is an activity deployment whose described capability is made accessible through a standard set of interfaces and is exercised consistent with terms and conditions defined by the service* $\mathtt{s}$*. It provides observable set of behaviors accessible via prescribed interfaces.*

2.2.2 Workflows

A *workflow* $\mathcal{W}$ is a well-defined and possibly repeatable pattern or systematic organization of activities designed to achieve a certain transformation of data [136]. A single workflow $\mathcal{W}$ can be defined in term of three basic components: a set of inputs denoted by $\mathcal{I}^w \in \mathcal{I}$, a set of outputs denoted by $\mathcal{O}^w \in \mathcal{O}$, and a set of activities denoted by $\mathcal{A}^w \in \mathcal{A}$.

Definition 10. *A* workflow (application) *is modelled as* $\mathcal{W} = (\mathcal{I}^w, \mathcal{O}^w, \mathcal{A}^w, \mathcal{V})$ *The activities* $\in \mathcal{A}^w$ *are executed in a well defined order. The simplest workflow is one with* $\mathcal{A}^w = \oslash$ *(empty set) that is,* $\mathcal{W} = (\mathcal{I}^w, \mathcal{O}^w, \oslash, \oslash)$*. This represents a workflow (itself) as a single activity.*

The activities in a workflow are interlinked as a graph with $\mathcal{V}$ *edges (dependencies).*

Let $\mathtt{succ}(\mathtt{a})$ *denotes the set of* successors *of an activity* $\mathtt{a} \in \mathcal{A}^w$*:*

$$\mathtt{a}_s \in \mathit{succ}(\mathtt{a}) \iff \exists\ (\mathtt{a}, \mathtt{a}_s) \in \mathcal{V}.$$

Similarly, let $\mathit{pred}(\mathtt{a})$ *denote the set of* predecessors *of an activity* $\mathtt{a} \in \mathcal{A}^w$*:*

$$\mathtt{a}_p \in \mathit{pred}(\mathtt{a}) \iff \exists\ (\mathtt{a}_p, \mathtt{a}) \in \mathcal{V}.$$

If $\mathit{pred}(\mathtt{a}) = \oslash$ *then* $\mathtt{a}$ *is a* start activity*:* $\mathtt{a} \in \mathcal{A}^w_{start}$*. Similarly, if* $\mathit{succ}(\mathtt{a}) = \oslash$ *then* $\mathtt{a}$ *is an* end activity*:* $\mathtt{a} \in \mathcal{A}^w_{end}$*. Additionally, the set of predecessors and successors of rank* p *of an activity* $\mathtt{a}$ *are referred as:*

$$\mathit{pred}^p(\mathtt{a}) = \mathit{pred}(\ldots \mathit{pred}(\mathtt{a})),$$

with p *recursive invocations of* pred*. Similarly:*

$$\mathit{succ}^p(\mathtt{a}) = \mathit{succ}(\ldots \mathit{succ}(\mathtt{a}))$$

with p *recursive invocations of* succ*. Two activities* $\mathtt{a}_1$ *and* $\mathtt{a}_2$ *are* independent *iff* $\nexists\ p$ *such that* $\mathtt{a}_1 \in \mathit{pred}^p(\mathtt{a}_2) \ \vee\ \mathtt{a}_1 \in \mathit{succ}^p(\mathtt{a}_2)$ *[136].*

Let $\mathcal{I}_a$ *denote the set of* input arguments *and* $\mathcal{O}_a$ *denote the set of* output arguments *of an activity* $\mathtt{a} \in \mathcal{A}^w$*:*

$$\mathcal{I}_a \subseteq \mathcal{I}^w \iff \mathit{pred}(\mathtt{a}) = \oslash$$

Similarly:

$$\mathcal{O}_a \subseteq \mathcal{O}^w \iff \mathit{succ}(\mathtt{a}) = \oslash$$

2.2.3 Grid Node

A *Grid node* (a.k.a. *Grid Site*) is a combination of both physical and logical resources. It is a *'farm'* of resources (computers, processors, services, applications) that is accessible through a unique address. The more prominent resources belonging to a node $\mathtt{g}_i \in \mathcal{G}$ are a set of *processors* $\mathcal{P}_i \in \mathcal{P}$ and a set of activities $\mathcal{A}_i \in \mathcal{A}$. where $\mathcal{P}$ is a set of all processors in the Grid $\mathcal{G}$, that is $\mathcal{P} = \sum_{\mathtt{g}_i \in \mathcal{G}} \mathcal{P}_i$. A minimum operating environment (middleware) is required for its computing power to be advertised and utilized. Formally, a *Grid node* can be defined as:

Definition 11. *A* Grid node (site) $g_i \in \mathcal{G}$ *is a combination of physical resources* $\mathcal{PR}_i \subset \mathcal{PR}$ *and logical resources* $\mathcal{LR}_i \subset \mathcal{LR}$*, that is,* $g_i \equiv \mathcal{R}_{g_i} = \mathcal{PR}_i + \mathcal{LR}_i$*. The resources* $\in \mathcal{R}_{g_i}$ *share same local security, interconnects, and resource management policies. Each resource* $\mathtt{r} \in \mathcal{R}_{g_i}$ *is managed under a single hosting environment accessible directly or indirectly through a unique reference* $\mathit{ref}(g_i)$*. In a* hosting environment *resources are administrated under a single trust domain and they are advertised and utilized through a single access point called the front-end resource (*Gatekeeper*)* $\in \mathcal{PR}_{g_i}$ *of the node* g_i*. Reducing to processors and activities,* $g \equiv \mathcal{R}_{g_i} = \mathcal{P}_i \cup \mathcal{A}_i$*.*

Note that the work presented in this book uses terms like *node* $\mathtt{g} \in \mathcal{G}$, *activity* $\mathtt{a} \in \mathcal{A}$, *activity type* $\mathtt{at}$, *activity deployment* $\mathtt{ad}$, and *workflow* $\mathcal{W}$. These terms are referred to as defined in this chapter unless stated otherwise.

2.3 Grid Operating Environment

The low-level *middleware* of the Grid constitutes the *Grid operating environment*. It refers to the security infrastructure, local resource management and provisioning, data access and movement, instrumentation and monitoring, policy and access control, accounting, and other services required for proper functioning of the Grid. Besides these services a gluing mechanism that binds these services together is also part of the Grid middleware [76]. Currently several middleware infrastructures have been developed such as Globus [10], gLite [68], Unicore [70] etc. However, Globus Toolkit is most widely used as it provides not only a basic set of services but a framework for developing new high-level Grid-enabled services as well.

2.3.1 Open Grid Services Architecture

Open Grid Forum [129] (OGF) is a standardization body for the Grid middleware and high-level services and their interaction mechanisms. The fundamental work is the formalization of *Open Grid Service Architecture (OGSA)* a.k.a. *anatomy of the Grid* [71]. OGSA defines a service-oriented architecture for the Grid that formalizes interaction and computation mechanism assuring interoperability on heterogeneous systems so that different types of resources can communicate and share capabilities.

> *OGSA realizes the* middleware *in terms of services, the interfaces these services expose, the individual and collective state of resources belonging to these services, and the interaction between these services within a service-oriented architecture (SOA)* [71].

The OGSA services framework is shown in Figure 2.2 (source OGF-OGSA [71]) in which services are built on *Web service standards*, with semantics, additions, extensions and modifications that are relevant to the Grid. OGSA introduces the following:

- *Security Infrastructure* is required for controlled access to services through robust security protocols and according to provided security policies. It includes an *authentication* mechanisms that establishes the identity of individuals and services, an *authorization* mechanism that accommodates various access control models, a *credential and policy delegation* that supports inter-services and inter-domain interactions, and mechanisms for ensuring the integrity of resources;
- *Resource Management and Provisioning* deals with resource management and virtualization, application deployment and configuration, optimized allocation and provisioning;
- *Execution Management* introduces a set of services that are required for workflow planning, application scheduling and mapping, execution control and monitoring;

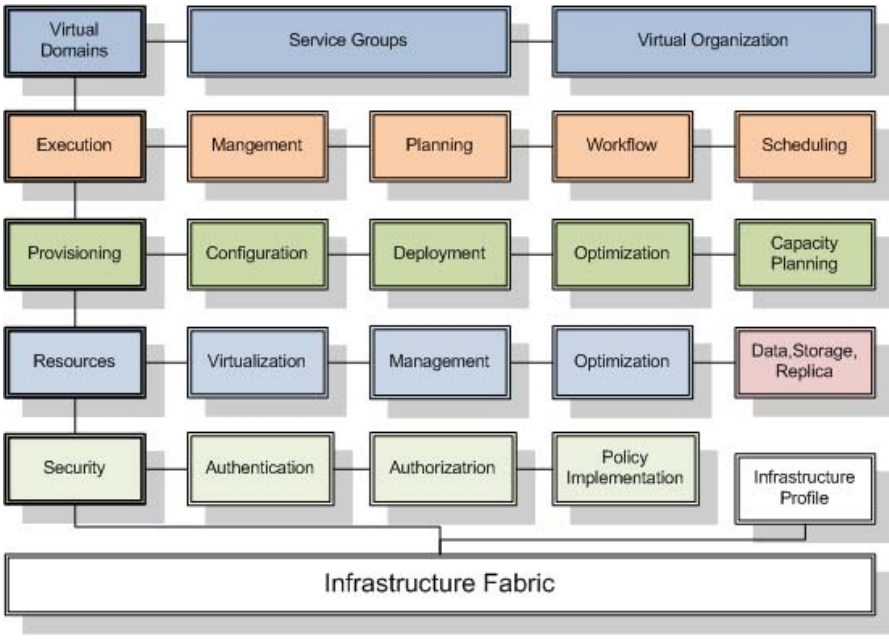


Fig. 2.2. Open Grid Services Architecture.

- *Data Management* introduces services required for storage management, replica management and data transportation;
- The virtualized resources (services) may participate in a virtual collection called *Virtual Domain* or *Virtual Organization (VO)*. This shares a set of collective capabilities and manageability framework;

  A Virtual Organization (VO) *is a conceptual group of individuals, services and/or institutions in which a well defined defined and highly controlled sharing of capabilities takes place. The sharing is clearly and carefully defined, just what is shared, who is allowed to share, and the conditions under which sharing occurs* [74].
- *Infrastructure Fabric* is a physical environment that may include well-known physical resources and interconnects such as computing hardware and networks, and physical equipment such as scientific instruments.

Currently available OGSA-compliant Grid middleware implementations are for instance the *Globus Toolkit* (Section 2.3.3) and *Askalon* (See Section 2.4), built on top of *Web Services* standards such as *WSRF*, *Web Services Description Language (WSDL)* and *Simple Object Access Protocol (SOAP)*.

2.3.2 WS-Resource Framework

the *Web Service Resource Framework (WSRF)* [191, 94] is a standard set of specifications for web services defined and approved by OASIS [125].

A *web service* is *stateless*: that means it retains no data between invocations or it cannot access data that is not part of the invocation messages. This limits the possibilities that can be done with web services in a generalized way. Nevertheless, the *statefulness* plays an important role in service-orientation. A conventional service implements a series of operations such that the result of one operation depends on a prior operation and/or prepares for a subsequent operation.

WSRF defines a standard mechanism for adding state in a web service in order to make it a stateful service. A service that acts upon stateful resources provides access to, or manipulates a set of logical stateful resources (documents) based on messages it sends and receives. WSRF provides a set of operations that web services may implement to become stateful, A web service client communicates with the service which allows data to be stored and retrieved. Clients includes the identifier of the specific resource (state or document) as part of service invocations request, encapsulated within the WS-Addressing [184] endpoint reference. The encapsulated address may be a simple URI address or a complex XML document that helps to identify or even fully describe the specific resource. Such a kind of web services is called WS-Resource [94] and the address of WS-Resource is referred as *Endpoint Reference*.

Alongside the notion of an explicit endpoint reference of a resource, a standardized set of web service operations to get/set resource properties and register/notify for any change in the state properties.

Definition 12. WS-Resource $wr \in \mathcal{A}$ *is a stateful web service that implements a standard set of operations to access and/or manipulate its state in a service-oriented fashion (Section 2.3.1). A special implied resource pattern [94] is used to describe a specific kind of relationship between a Web service and one or more stateful resources.* WS-Addressing *[184] standardizes the relationship with an endpoint reference construct.*

2.3.3 Globus Toolkit

The *Globus Toolkit 4 (GT4)* is a WSRF-compliant low-level Grid middleware that provides Grid operating environment in a service-oriented fashion. It is considered the first reference implementation of service oriented architecture proposed for the Grid in the OGSA. It provides a set of fundamental components that can be used either independently or in a combination to develop new Grid-enabled applications [10].

As shown in Figure 2.3, the toolkit includes a set of services for Grid security infrastructure (GSI) [78], information, data management, execution management, and libraries for common runtime. It supports virtualization, as its core services, interfaces and protocols allow users to access remote resources as if they were local while simultaneously preserving resource integrity and autonomy.

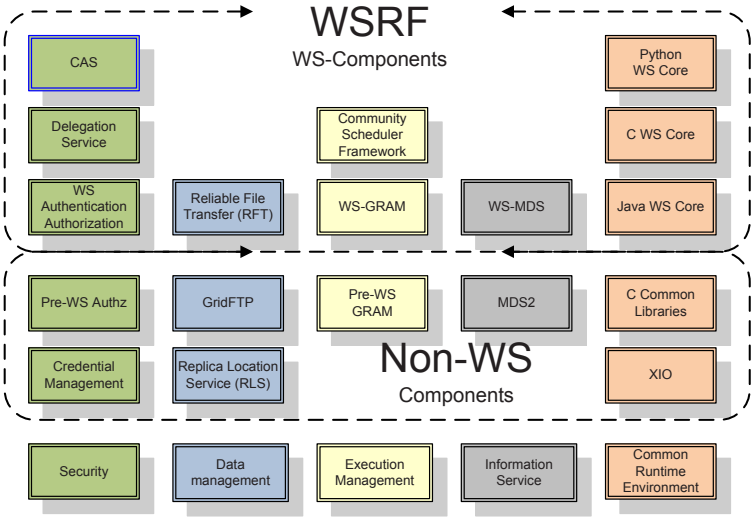


Fig. 2.3. Globus Toolkit 4 services.

The common runtime components provide GT4 Web services with a set of libraries and tools that allows these services to be platform independent, to build on various abstraction layers and to leverage functionality lower in the Web services stack. Execution management is done through a set of service components collectively referred to as the Grid Resource Allocation and Management (GRAM) [41]. The latest version of globus toolkit (GT4) [76] consists of three sets of components:

1. The *basic infrastructure service*s that includes execution management (GRAM) [41], data access and movement (GridFTP and RTF) [9], Replica Management (RLS), discovery and monitory (WS-MDS), and credential management (MyProxy [124], CAS i.e. a Community Authorization Service, Delegation Service) etc.
2. An extended version of open source *Web Service container* implemented in *C, Java* and *Python* as reference implementation of *hosting environment*s supporting open source Web Service (WS) technologies including WS Resource Framework (WSRF) [191], WS-Notification and WS-Security. These container provides basic set of infrastructure services and implementation that are required for building new high-level Grid-enabled applications and services for runtime environment.
3. A set of *client libraries* implemented in *C, Java* and *Python*. These libraries are used to access core and user developed services and components.

The toolkit includes two types of service implementations: the pre-WS components offer capabilities with a non-standard proprietary interfaces, whereas WS components use standardized technologies and interfaces (i.e. WSRF).

Execution Management

The *Execution Management (GRAM)* [41, 76] simplifies the use of remote systems by providing a single standard interface for requesting and using remote system resources for the execution of jobs. The most common use of *GRAM* is remote job submission and control. It comes with both WS and pre-WS favors. The *WS-GRAM* is a Web Services Resource Framework (WSRF) [191]-based GRAM implementation along with some additional features: it works with GT4 authorization framework and can be configured to work with a chained authorization by configuring multiple policy decision points (PDPs). It works as entry point and gatekeeper to a Grid node (site) abstracting out the underlying functionality of off-the-shelf local resource managers (LRM) such as Sun Grid Engine (SGE) [168] and Load Sharing Facility (LSF) [132] etc. It provides a common Web service interface for initiating, monitoring, and managing execution of arbitrary computations on a Grid node.

Each execution by the GRAM is managed as a newly spawned limited-time service for monitory and controlling its execution. *WS-GRAM* works with an authorization framework that is customizable.

Information Services

The *Monitoring and Discovery Service (MDS)* [39] is a distributed service that works in a hierarchical fashion. MDS2 (pre-WS part of GT4) is based on LDAP (Lightweight Directory Access Protocol) information model [103] whereas *MDS4 (WS-MDS: WS part of GT4)* works based on WSRF-based GT4 resource aggregation framework [76]. WSRF provides a mechanism for notification and associating properties with resources as their state in XML format. Services can be enabled with state and notification mechanisms and can register with their containers for sharing their state to other services. Container then can register in other containers thus forming a hierarchical structure. The information model used by the both version of MDS (WS and Pre-WS) largely use Grid resource information model described by *GLUE schema* [85].

Grid Security Infrastructure

In accordance with OGSA (Section 2.3.1), the *Globus Toolkit* provides a *Grid Security Infrastructure (GSI)* [78] that has been accepted as the de-facto standard by the Grid community for authentication and secure communication across the applications and the services over the Internet. It works with *Public Key Cryptography* that is based on public/private key pair and used as fundamental technology for encrypting and decrypting messages. GSI uses *X.509 Certificates* for representing the identity of each client (Grid user) that is required for authentication. *Mutual Authentication* ensures that the two parties involved in communication trust each other certificate authorities.

It provides a *Single Sign-On* mechanism that restricts the user authentication to one single password specification during a working session. This is done with a *proxy* (a short-term credential) that is created as a new key pair digitally signed by the user's (semi-permanent) certificate. The *proxy* temporarily represents the user Grid identity. This allows the true private key of the user be decrypted for a minimum amount of time, until the signed proxy is generated. Furthermore, a *delegation* of proxy allows remote services to act on behalf of the client through the creation of remote proxies that impersonate the user.

The GSI cryptography can be applied at two levels in the Grid: *network layer* and *message layer*. Security at the message layer is more powerful than the security at the network layer due to the data encryption at a higher level of abstraction.

2.4 Askalon: A Grid Runtime Environment

The high-level middleware i.e. the *Grid runtime environment* provides services on top of Grid operating environment such as workflow planning, scheduling, resource selection and mapping, application deployment, capacity planning, workflow enactment, monitoring and performance analysis etc. Examples include Askalon [61] a Grid application development and execution environment, ICENI [116], Pegasus [58], GridBus [27], P-Grade [131], Triana [174], Taverna [171], and Kepler [4].

Most existing Grid application development environments provide the application developer with a nontransparent Grid. Commonly, application developers are explicitly involved in tedious tasks such as selecting activities deployed on specific nodes, mapping applications onto the Grid, or selecting appropriate computers for their applications. Moreover, many programming interfaces are either implementation-technology-specific (e.g., based on Web services [14]) or force the application developer to program at a low-level middleware abstraction (e.g., start task, transfer data [112]). While a variety of graphical workflow composition tools are currently being proposed, none of them is based on standard modeling techniques such as Unified Modeling Language (UML).

Askalon is a high-level middleware for Grid application development and execution (Figure 2.4) [62, 63]. Its ultimate goal is to provide an invisible Grid to the application developers. In Askalon, the user composes Grid workflow applications graphically using a UML-based workflow composition and modeling service. Additionally, the user can programmatically describe workflows using the XML-based *Abstract Grid Workflow Language* (AGWL) [64], designed at a high level of abstraction that does not comprise any Grid technology details. The AGWL representation of a workflow is then given to the Askalon WSRF-based middleware services (runtime system) for scheduling and reliable execution on Grid infrastructures.

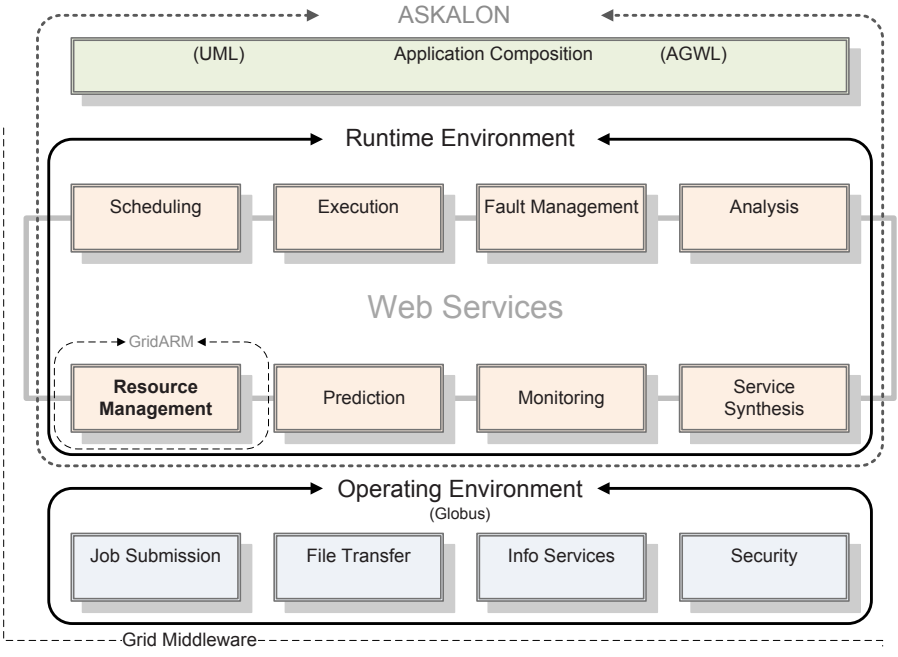


Fig. 2.4. The Askalon architecture.

Askalon provides a variety of services that support the composition and execution of scientific workflows on the Grid.

2.4.1 Workflow Composition

Askalon offers two interfaces for generating large-scale scientific workflows in a compact and intuitive form. Firstly, it offers to the end user the privilege of composing workflows through a graphical modeling tool based on the UML standard that combines Activity Diagram modeling elements in a hierarchical fashion. Secondly, it offers an XML-based workflow language called *Abstract Grid Workflow Language (AGWL)* that enables the composition of workflow applications from *activities* interconnected through control-flow and data-flow dependencies. The control-flow constructs include sequences, *directed acyclic graphs (DAG)*, `for`, `forEach`, `while` and `do-while` loops, and `if` and `switch` constructs, as well as more advanced constructs such as `parallel` activities, `parallelFor` and `parallelForEach` loops, and collection iterators [139]. In contrast to most existing work, AGWL is not bound to any implementation technology such as Web services.

2.4.2 Resource Management

Resource Management (called *GridARM*) [154, 158, 156], the topic of this book, is an integral part of Askalon. It renders the boundaries of Grid resource management and provides resource discovery, selection and provisioning, negotiation-based advanced reservation, and capacity planning for optimized resource allocation. Furthermore, *GridARM* is extended to cover logical resources (activities) and provides a dynamic registration and automatic deployment framework for activities [158]. This extension is referred to as *GLARE* (Chapter 4).

In combination with the AGWL, *GridARM* shields Grid users from the low-level middleware complexities. The main client of the *GridARM* is the Askalon *Scheduler*.

2.4.3 Workflow Scheduling

The Scheduler [187, 62] service prepares a workflow for execution on the Grid. It processes the workflow specification described in AGWL, converts it to an executable form, and maps it to available Grid resources. It is a best-effort Grid workflow scheduler, adapted to apply different algorithms which can be used by the service as interchangeable plug-ins.

The scheduling process consists of three main phases: refinement, mapping, and rescheduling upon important events triggered by the event generator component part of the Askalon (See Figure 2.4).

The refinement process resolves all ambiguities and refines sophisticated workflow graphs into simple DAGs on which existing graph-scheduling algorithms can be applied.

The mapping of a refined workflow onto the Grid is done based on a modular architecture, where different DAG-based scheduling heuristics can be used interchangeably. Currently it incorporates three scheduling algorithms: Heterogeneous Earliest Finish Time (HEFT), a genetic algorithm, and a myopic just-in-time algorithm [187].

After the initial scheduling, the workflow execution is started based on the current mapping until the execution finishes or any interrupting event occurs. The *event generator* module uses the *Monitoring Service* to watch the workflow execution and detect whether any of the initial assumptions, also called execution contracts, have been violated. The execution contracts that we currently monitor include structural assumptions made by the workflow converter, external load on processors, processors no longer available, congested interconnection networks, or new Grid node available. In case of a contract violation, the *Scheduler* sends a rescheduling event to the Enactor (Section 2.4.4), which generates and returns a new workflow based on the current execution status (by excluding the completed activities and including the ones that need to be re-executed) [187].

Scheduling Function

Precisely, the *scheduling function* determines effective mappings (schedule) of workflows onto the Grid $\mathcal{G}$ using graph-based heuristics and optimization algorithms that benefit from performance prediction and resource manager services. More specifically, it assigns different activities in a workflow to different *activity deployments*.

Definition 13. *Given a workflow* $\mathcal{W} = \{\mathcal{I}^w, \mathcal{O}^w, \mathcal{A}^w, \mathcal{V}\}$ *consisting of* n *activities* $\mathtt{a}_i \in \mathcal{A}^w, 1 \leq i \leq n$, *a* schedule *is defined by the scheduling function* $\mathtt{sched}^w : \mathcal{A}^w \mapsto \mathcal{D}$ *where* $\mathtt{sched}^w$ *assigns to each activity* $\mathtt{a}_i \in \mathcal{A}^w$ *an activity deployment* $\mathtt{ad}_i \in \mathcal{D}_i$. *The* $\mathtt{sched}^w$ *is a total function.*

2.4.4 Workflow Enactment

The *Enactor (Execution Engine)* targets reliable and fault-tolerant execution of workflows through techniques such as checkpointing, migration, restart, retry, and replication. It is responsible for controlling the execution of a workflow application based on the Grid mapping decided by the *scheduler*. The main tasks performed by the *enactor* are to coordinate the workflow execution according to the control-flow constructs and to effectively resolve the data-flow dependencies specified by the application developer in *AGWL* [52, 62].

2.4.5 Performance Prediction and Analysis

Performance Analysis [121, 137] supports automatic instrumentation and bottleneck detection (e.g., excessive synchronization, communication, load imbalance, inefficiency, or nonscalability) within Grid workflow executions. Furthermore, Askalon analysis comprises service-level negotiation and agreement on a variety of nonfunctional parameters.

A *Performance Prediction* [118] service estimates execution times of workflow activities through a training phase and statistical methods using the Performance Analysis service.

2.5 Semantic Grid

One of the key challenges in today's Grids is the need to deal with knowledge and data resources that are distributed, heterogeneous, and dynamic, and an effective elicitation of implicit knowledge in the system. In such systems, a complete global understanding is impossible to achieve. It is therefore needed to go beyond centralized knowledge elicitation and develop an effective, open standard, and distributed solutions.

The Semantic Grid aims to overcome this problem by adding meaning to the Grid in general and the underlying resources in particular. In this

way, the Semantic Grid not only provides a general semantic-based computational network infrastructure, but a rich, seamless collection of intelligent, knowledge-based services for enabling the management and sharing of complex resources and reasoning mechanisms. In the Semantic Grid, knowledge and semantics are deployed explicitly for Grid applications and for the development of innovative Grid infrastructures [43, 46, 36]. This knowledge-oriented semantics-based approach to the Grid goes hand-in-hand with the exploitation of techniques and methodologies from intelligent software agents and web services representing various components of the virtual organizations and often interacting in a P2P way.

> The *Semantic Grid* $\mathcal{G}^s \subseteq \mathcal{G}$ is an extension of the current Grid in which information and services are given well-defined meaning through machine-processable descriptions which maximize the potential for sharing and reuse. It is believed that this approach is essential to achieve the full richness of the Grid vision, with a high degree of easy-to-use and seamless automation enabling flexible collaborations and computations on a global scale [88].

One way of achieving human understandable and machine processable semantic descriptions of concepts is possible through the use of *ontologies*.

2.5.1 Ontology

An ontology is a specification of a conceptualization that provides vocabularies with explicitly defined and machine understandable meanings.

> An *Ontology* represents an explicit conceptual model with formal logic-based semantics. Its descriptions may be queried with abstract goals, may foresight required capabilities, or may be checked to avoid inconsistency in the declarations. Rules-based management of Grid middleware builds a rigorous approach towards giving the declarative descriptions of components and services a well-defined meaning by specifying ontological foundations and by showing how such foundations may be realized in practice [90].

An ontology model also refers to as *T-Box*. An information base developed by using an ontology model is also referred to as *knowledge-base* or *A-Box*.

2.5.2 Web Ontology Language

The *Web Ontology Language (OWL)* [146] is a formal standard language for representing ontologies in the Semantic Web. In OWL, an ontology is a set of definitions of classes and properties and the constraints to be employed on them. The OWL has three variants: OWL-Lite, OWL-DL, and OWL-full, each with different levels of expressiveness.

In order to provide a powerful expressiveness and fact stating ability, the OWL inherits features from RDF [34] and RDF Schema [42] and extends them by providing new and powerful constructs. It can declare and organize classes in a subsumption hierarchy, and the classes can be expressed as a logical combination of other classes. The properties in OWL can also be organized in a sub-property hierarchy. The OWL also provides different kinds of *restrictions* on classes and properties, which are considered as specialized concepts. The concepts (classes and properties) are represented as fully-qualified names with URIs.

In the domain of the Semantic Web, ontologies play an important role in automating processes to access semantics information. They provide structured and extensible vocabularies that demonstrate the relationships between different terms allowing intelligent agents to flexibly and unambiguously interpret their meanings. In the Description Logics, the fundamental reasoning of *concept expression* is subsumption [109] which checks whether or not a concept *is-a* subset (or superset) of an other concept.

2.5.3 Ontology Query Language

Simple Protocol and RDF Query Language (SPARQL) [182] is a candidate recommendation as an RDF query language by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium [164]. A query in SPARQL may consists of triple patterns, conjunctions, disjunctions, and optional patterns. Variables are outlined through the "?" prefix. The query processor searches for all hits that match the patterns defined as RDF-triples. SPARQL is property-orientation, that means that concepts matches can be conducted solely through class-attributes or properties. Since OWL can be represented in RDF format therefore SPARQL can be used for querying OWL model as well. OWL-QL [66] is a formal language and protocol for a query-answering dialog between intelligent agents using knowledge represented by the OWL Knowledge Base. It precisely specifies the semantic relationships among a query, a query answer, and the *knowledge base* used to produce the answer. An OWL-QL query can specify which of the URIs referred to in the query pattern are to be interpreted as variables. Variables come in three forms: *must-bind*, *may-bind*, and *don't-bind*. Answers are required to provide bindings for all the must-bind variables, may provide bindings for any of the may-bind variables, and are not to provide bindings for any of the don't-bind variable. OWL-QL uses the standard notion of logical entailment: query answers can be seen as logically entailed sentences (OWL facts and Axioms) of the queried knowledge base.

2.6 Resource Management

Conventionally, resource management is a way of delivering available resources when they are needed in an effective and efficient way. In Askalon Grid run-

time environment, resource management (a.k.a *GridARM*) covers on-demand *provisioning*, optimized resource allocation, negotiation-based advance reservation and *capacity planning*. The main *client* of the *GridARM* is the *scheduler* as explained in Section 2.4.3

The *provisioning* is referred to as delivering a set available resources $\mathcal{R}^a \subseteq \mathcal{R}$ to a *client* $\mathtt{c} \in \mathcal{C}$. Available resources $\mathcal{R}^a \subset \mathcal{R}$ are discovered, selected according to the request $\mathtt{q} \in \mathcal{Q}$ made by a client $\mathtt{c} \in \mathcal{C}$ and finally the selected resources $\mathcal{R}^s \subseteq \mathcal{R}^a$ are offered to the same client $\mathtt{c}$.

Definition 14. *A* client $\mathtt{c} \in \mathcal{C} | \mathcal{C} = \{\mathtt{c}_1, ..., \mathtt{c}_m\}, m \in \mathbb{N}$ *is a Grid user or a software component that requests for a resource* $\mathtt{r} \in \mathcal{R}$ *for* allocation *and utilizes its capability for a certain time. Alternatively, a* client *is also referred to as resource* consumer *or resource* requester*. . Each client* $\mathtt{c}_i \in \mathcal{C}$ *joins the Grid with a resource request* $\mathtt{q}_i \in \mathcal{Q}$*.*

Definition 15. *A* resource request $\mathtt{q}_i \in \mathcal{Q} | \mathcal{Q} = \{\mathtt{q}_1, ..., \mathtt{q}_m\}, m \in \mathbb{N}$ *is a query made by a client* $\mathtt{c}_i \in \mathcal{C}$ *for allocation of a set of resources. It consists of a set of resource constraints (terms and conditions)* $\mathcal{T}_i \subseteq \mathcal{T}$ *that needs to be matched with offered capabilities of available resources* $\mathcal{R}^a \subseteq \mathcal{R}$*. The matched resources are selected and offered to the client. For each request* $\mathtt{q}_i \in \mathcal{Q}$ *there is a set* $\mathcal{T}_i = \{\mathtt{t}_i 1, ..., \mathtt{t}_i n_i\}$ *of constraints that is part of the* $\mathtt{q}_i$*, where* n_i *may be different for different* $\mathtt{q}_i \in \mathcal{Q}$*.*

Definition 16. *A resource* constraint $\mathtt{t} \in \mathcal{T}$ *defines boundary values of a resource capability that is requested by a* client *or offered by a resource* provider*. For instance, in case of* `totalCPUs`*, maximum value, minimum value, exact value, degree of importance etc. defines a constraint over* `totalCPUs`*.*

In terms of resources, *Grid nodes* $\in \mathcal{G}$, physical constituents especially *CPUs* or *processors*, and logical constituents especially *activity deployments* are referred to as provisionable, allocatable or consumeable resources. Abstract description of resources such as *activity types* and *abstract workflows* are used by a *client* $\mathtt{c} \in \mathcal{C}$ to look up consumeable resources. A Grid user who provides a consumable resource is referred to as *resource provider*(or simply *provider*).

Beside provisioning, an important aspect of resource management is to maintain capabilities of the underlying resources. The principle is to invest in resources as stored capabilities, then unleash the capabilities as demanded. This is also referred to as resource leveling [195]. However, resource leveling does not fit in resource provisioning scenario defined by this book as in on-demand provisioning the resource leveling may reduce *utility*. An other dimension of resource management is to allocate available resources in a combination so that the combination provides either aggregated capability or a new capability all together. Broadly, a *resource manager* is responsible to maintain the natural integrity of the Grid. OGF [129] defines *management*, *resource management*, and *resource manager* as:

Management is the process of monitoring an entity, controlling it, maintaining it in its environment, and responding appropriately to any changes of internal or external conditions.
Resource management is a generic term for several forms of management as they are applied to resources.
A *resource manager* is a manager that implements one or more resource management functions.

Based on this explanation, resource management for the Grid can be defined as:

Definition 17. Resource management *describes the process of allocating resources* $\in \mathcal{G}$ *on-demand in an effective and efficient way while optimizing resource* utilization *and maintaining resource capabilities without undermining the natural* integrity *of the Grid* $\mathcal{G}$ *and* utility *of the underlying resources* $\in \mathcal{G}$. *The process of* resource management *works as an intermediary between resources and their clients, and thus it is also referred to as* resource brokerage.

The *utilization* is referred to as the proportion of the resource *capability* (See Definition 4) which is used by the client $\mathsf{c} \in \mathcal{C}$. Lower utilization represents inactivity of a resource. Maintaining capabilities referred to as advertising available capabilities and representing exact state of resources $\mathcal{R}$ so that they can be selected correctly for clients $\in \mathcal{C}$. The *integrity* referred to as the resource autonomy and (usage) policies associated with the resource.

Definition 18. *The* utility *is a degree of felicity, contentment or preference that is measured relative to ideal value. Given this degree, one may explain* capability *in terms of attempts to increase one's utility. The utility is measured by a utility function. A* utility function $\mu : \mathcal{G} \rightarrow \mathbb{R}$ *ranks each resource* $\mathsf{r}_i \in \mathcal{G}$. *If* $\mu(\mathsf{r}_i) \geq \mu(\mathsf{r}_j)$ *then the client* $\in \mathcal{C}$ *prefers* r_i *to* r_j. *Also, a* utility function $\mu : \mathcal{G} \rightarrow \mathbb{R}$ *rationalizes a preference relation* $\preceq$ *on* $\mathcal{G}$ *such that:*

$$\forall \mathsf{r}_i, \mathsf{r}_j \in \mathcal{G} | i \neq j, \mu(\mathsf{r}_i) \leq \mu(\mathsf{r}_j) \iff \mathsf{r}_i \preceq \mathsf{r}_j \ .$$

Example 1 (Utility).
Suppose a client discovers a set $\mathcal{G} = \{\oslash, \mathsf{g}_1, \mathsf{g}_2, \mathsf{g}_3, \mathsf{g}_4, \mathsf{g}_5\}$ of nodes in the Grid as candidates for its application, where each candidate possesses a number of CPUs and the architecture $\in \{32b, 64b, 128b\}$ as: $\mathsf{g}_1 = \{2cpu, 32b\}$, $\mathsf{g}_2 = \{1cpu, 64b\}$, $\mathsf{g}_3 = \{2cpu, 64b\}$, $\mathsf{g}_4 = \{3cpu, 32b\}$, and $\mathsf{g}_5 = \{1cpu, 128b\}$ with its utility functions as $\mu(\oslash) = 0$, $\mu(\mathsf{g}_1) = 1$, $\mu(\mathsf{g}_2) = 2$, $\mu(\mathsf{g}_3) = 4$, $\mu(\mathsf{g}_4) = 2$ and $\mu(\mathsf{g}_5) = 3$. Then the client will prefer a candidate with $64b$ architecture and $1cpu$ (i.e. g_2) to a candidate with $32b$ architecture and $2cpu$ (i.e. g_1), but will prefer a candidate with $64b$ architecture and $2cpu$ (i.e. g_3) to a candidate with $128b$ architecture and $1cpu$ (i.e. g_5).

Here the use of multiple utility functions for each pair is for exemplar purpose. In practice, a single utility function is used for different combinations of available options.

2.6.1 Provisioning

Resource provisioning in the Grid consists of two complement processes: resource discovery and resource selection. Finding resources capable of doing something useful in the Grid is referred to as *resource discovery* whereas matching or correlating discovered resources to the requirements of a client so that they can be offered is called *resource selection* [153].

Definition 19. *The process of offering allocations of discovered and selected resources to the client is referred to as resource provisioning or simply* provisioning.

Example 2 (Provisioning).
Consider an example Grid $\mathcal{G}$ with nodes $\mathtt{g}_1 - \mathtt{g}_5$ discovered from different information services as shown in Example 1. If a client requests a node with 64*b* or higher architecture, then the filtered candidate set will be $\{\mathtt{g}_2, \mathtt{g}_3, \mathtt{g}_5\}$. If the client has no further constraints then according to the *utility function* (Definition 18) node $\mathtt{g}_3$ will be selected as a best candidate among the available options and thus will be offered for allocation to the client for a certain timeframe.

2.6.2 Allocation Negotiation

A process of assigning available resources to contending clients. The substance of this process is called *allocation*. It can be defined as:

Definition 20. *An* allocation $\mathtt{alloc} \in \mathcal{L} | \mathcal{L} = \mathcal{P}_i \times \mathcal{A}_i$ *is an assignment of a proportion of the capability* ($\mathcal{P}_i \times \mathcal{A}_i$) *of a node* $\mathtt{g}_i \in \mathcal{G}$ *to a resource request* $\mathtt{q} \in \mathcal{Q}$ *made by a client* $\mathtt{c} \in \mathcal{C}$ *through an allocation function* $\mathtt{alloc} : \mathcal{G} \times \mathcal{Q} \mapsto \mathcal{L}$, *such that* $\mathtt{alloc}(\mathtt{g}_i, \mathtt{q}) = \mathtt{alloc} \equiv (\mathtt{p}, \mathtt{a}) | \mathtt{p} \in \mathcal{P}_i, \mathtt{a} \in \mathcal{A}_i$, *where request constraints* $\in \mathtt{q}$ *are matched to the offered constraints* $\in \mathtt{g}_i$.

The allocation process has two parts: Firstly, a basic decision is made in which suitable resources are assigned to the clients. Secondly, a contingency mechanisms is prepared. A priority ranking of resources can be made based on a selection criterion, for instance the $\mu(\mathtt{g})$ of a resource $\mathtt{g} \in \mathcal{R}$.

An advance reservation is a special form of allocation or a priority provisioning in which a resource allocation is made sometime in future and ensured later on that allocated resource remains available during the agreed upon timeframe.

Definition 21. *An* advanced reservation *is an allocation* $\in \mathcal{L}$ *in which a possibly limited or restricted delegation of a particular resource capability is made available sometime in future to a client* $\mathtt{c} \in \mathcal{C}$ *on a request* $\mathtt{q} \in \mathcal{Q}$ *through a negotiation process [156]. An advance reservation is defined by an* allocation function $\mathtt{alloc}_{ad} : \mathcal{G} \times \mathcal{Q} \mapsto \mathcal{L} | \mathcal{P} \times \mathcal{A} \times T$, T *is a* 3^{rd} *dimension that represents a time horizon.*

The process of *negotiation* is an interaction of Grid resource management with its clients to influence for better resource utilization. This interaction includes the process of resolving contentions among competing applications, agreeing upon or bargaining for individual or aggregated capabilities with better service quality provided by resources. Thus, negotiation can be considered as a process that generates multiple options for the same task.

Definition 22. Negotiation *is a process in which alternative allocation offers* $\mathcal{L}_{\mathtt{q}}^{o} \subseteq \mathcal{L}$ *are generated for a resource request* $\mathtt{q} \in \mathcal{Q}$ *by a client* $\mathtt{c} \in \mathcal{C}$*. The client may opt for an offer* $\mathtt{alloc} \in \mathcal{L}_{\mathtt{q}}^{o}$ *if* $\mathtt{alloc}(\mathtt{q}, \mathtt{r}) = \mathtt{alloc} \wedge \mu(\mathtt{r}) = \top$ *(*$\top$ *means maximum).*

Negotiation also involves multiple interactions, nevertheless, in the Grid, multiple interactions is a time consuming process that can be reduced by generating smart options as alternative allocation offers. Therefore, the focus of this book is the alternative offer generation rather than multiple interactions as part of *negotiation* process.

Negotiation for resource allocation in the Grid refers to how the parties (requester and provider) negotiate, the context of the negotiations, the parties to the negotiations, the relationships among these parties, the communication between these parties, the tactics used by the parties to agree upon certain terms and conditions, and the sequence and stages in which all of them reel off.

The main negotiators in the Grid are *resource management* and *scheduler*. Resource management system may use a variety of algorithms ranging from a simple attentive allocation to a more complex capacity planning strategy in order to improve *utility* $\mu\mathtt{r}$ of the resource $\mathtt{r} \in \mathcal{R}$. On the other hand, a scheduler works for clients and uses various strategies to improve client or application utility [188].

The substance of the *negotiation* is an *allocation* (or *advance reservation*) that is represented in the form of an agreement. An agreement includes not only time constraints but other quality of service parameters as well.

2.6.3 Capacity Planning

Resource management is visioned as a custodian of resources. It tries to optimize resource utilization [156]. The process in which the resource utilization can be improved by considering various constraints while allocating available resources $\in \mathcal{R}$ among the interested parties is referred to as *capacity management*. Capacity management along a planning horizon, for instance the *time*, is referred to as *capacity planning* [156]. It involves multi-constraint optimization. The constraints $\in \mathcal{T}$ include *cost*, *processors*, $\mathcal{P}$, *memory*, *start time* (`startt`), *end time* (`endt`), and *duration* (`duration`).

> *Capacity management* is a strategic process that focuses on the present, whereas *capacity planning* is a forward looking strategic activity of monitoring, understanding, and reacting to the clients' behavior in order to maximize the global utility [156].

Definition 23. Capacity planning *in the Grid is a forward looking strategic process of making allocations* $\in \mathcal{L}$ *of available nodes* $\in \mathcal{G}$ *to the contending applications in such a way that overall resource utility* $\mu(\mathcal{G}) \in \mathbb{R}$ *is maximized without compromising over the requested terms and conditions (constraints).*

2.6.4 Manageability Models

Manageability is an ability of a resource to be managed through well defined interfaces and interaction patterns. Several manageability models have been proposed for the Grid including: *centralized*, *decentralized*, and *hierarchical*. *Centralized management* of resources is a traditional approach that does not work with large-scale distributed resources. Instead, decentralized and hierarchical models are considered suitable for the Grid [26]. The modern approach for addressing manageability challenges is a set of new decentralized management models that have been studied for the resource management in the Grid. These models include: *peer-to-peer*, *service-orientation* and *superpeer*.

Peer-to-Peer Model

In a *Peer-to-Peer model (P2P)* [130] an aggregation of equivalent resources (called peers) dispersed across the Internet is formed in which peers share part of their capabilities (e.g., processing power, storage capacity, network link bandwidth etc.) directly with each other through diverse connectivity between peers in a network without passing through intermediate entities.

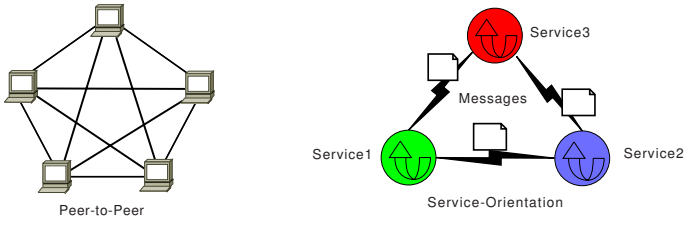


Fig. 2.5. Peep-to-Peer and Service-Oriented Models.

In contract to client/server model where communication is usually to and from a central server, in a peer-to-peer model only equal peers that simultaneously function as both 'clients' and 'servers' to other nodes on the network. This is depicted in Figure 2.5.

Service-Oriented Model

Service-oriented model relies on service-orientation as its fundamental design principle in which the model uses loosely coupled services to support the

requirements for the manageability of underlying resources. Service-oriented architecture is getting increasingly popular because of its similarities with the real world. In a community, people provide services to each other but without having close bindings or dependencies to each other. Similarly, the service-orientation in the Grid, resources are abstracted out as services and coordinate with each other through messages without having any hard bindings. This is depicted in Figure 2.5. If a resource is not available for a certain activity, the client may refer to an alternative option.

> A *service-oriented model* is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations [125].

The strength of peer-to-peer architectures is the high-degree of scalability and fault tolerance [136], whereas a service-oriented model is portable since it relies on self contained loosely coupled services. Services can be maintained and migrated independently without overall system downtime.

Superpeer Model

For the Grid resource management, this book introduces a new approach that is a combination of both peer-to-peer and service-oriented models. In this model, nodes $\in \mathcal{G}$ are organized in groups. As depicted in Figure 2.6, within a group service-oriented interaction model is used whereas inter group communication is provided by applying a peer-to-peer model. Each group selects a representative node, and all selected representative nodes interact with each other in a peer-to-peer fashion [158]. The groups are formed based on resource capabilities rather than their geographical locations or similarities with each other.

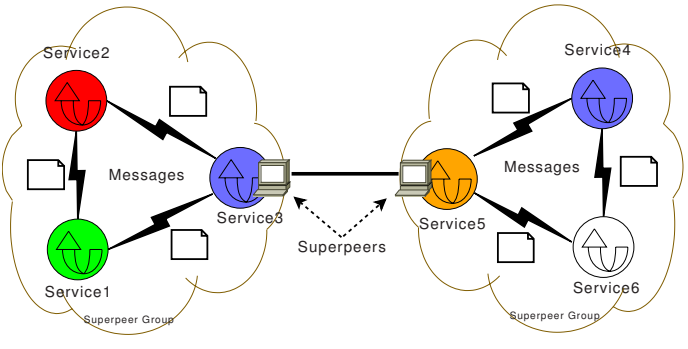


Fig. 2.6. Superpeer model.

> A *superpeer model* is a distributed resource management infrastructure in which resources are organized in groups where each group contains heterogeneous resources with diverse capabilities. Intra-group interaction is provided in a service-oriented fashion whereas peer-to-peer model is applied for inter-group interaction.

The superpeer model is described in more detail in Chapter 3.

2.7 Summary

This chapter presents a model of *the Grid*. It starts with general description and definition of the Grid, its characteristics, components and different architectures. Then it gives a formal overview of infrastructure fabric and applications along with definitions of Grid resources including the physical and logical resource, Grid node, activity, and workflow. It describes the Grid middleware by covering both Grid operating and runtime environments. A general overview of OGSA (Open Grid Services Architecture) that is a reference service-oriented architecture for the Grid followed by *WS Resource Framework*, Globus Toolkit, and the Askalon runtime environment is presented. Finally, a resource management model is presented that is topic of this book. Various terms are defined and explained which are used in the following chapters.

The next chapter starts with architecture of the *resource management (GridARM)* and describes resource selection, on-demand provisioning and load distribution problems to be handled by a resource manager.

3

Grid Resource Management and Brokerage System

The emergence of the Grid and ever evolving applications competing for scarce resources has accentuated the need for an adaptable, scalable and extensible resource selection and brokerage mechanism. This chapter presents the Askalon resource management system *called* GridARM *that delivers resources automatically on-demand. It is designed and developed as a scalable and extensible resource manager for the Grid. It works for resource providers and optimizes Grid utility with fair load distribution among resources. It renders boundaries of resource discovery, selection, brokerage, advance reservation, service-level agreement and capacity planning. This chapter starts with a general overview of the entire* GridARM *system architecture followed by a detailed description of resource selection model and brokerage implementation. The main focus of this chapter is brokerage of physical resources in the Grid.*

3.1 Introduction

In conventional computing systems, an effective resource management is rather straightforward since a resource manager has complete control over the underlying resources. Nevertheless, in the Grid, the resource management has to deal with heterogeneous, shared, and variant resources distributed under multiple trust domains. A resource manager for the Grid has to:

- balance global resource sharing with their local autonomy
- address issues of multiple layers of schedulers
- work with contending system participants having inconsistent performance goals and assorted local and global policies.

There is no central control over the heterogeneous resources distributed under multiple administrative environments. Each resource may have multiple schedulers and queuing systems mostly covered under the high-level Grid schedulers or meta-schedulers. In case of scarce resources, concurrent users with similar

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 47–78, 2010.

requirements may have to compete for similar resources. These challenges emphasize a very important and crucial role of resource management in the Grid.

In the Grid computing literature, *job scheduling* is represented as a part of the *resource management* [77, 149, 145, 150, 8, 44], therefore in most of the existing Grid infrastructures job scheduling is integrated with resource provisioning and the terms *scheduling* and *resource management* are used interchangeably. Nevertheless, in advanced Grids which support enhanced quality of service (QoS) and where negotiation between resource requester and provider is necessary, separation between *scheduler* and *resource manager* is important because both have conflicting goals. This is the main driving force of Askalon architecture. Furthermore, the Grid has been evolved enough that *job scheduling* and *resource management* can be addressed separately in a service-oriented fashion as defined by OGSA (Section 2.3.1). In the *Askalon Grid runtime environment* (Section 2.4), the very two components i.e. *scheduling* and *management* are developed not only as self-contained building blocks but they also work for two classes of users having conflicting goals. As depicted in Figure 3.1, the *scheduler* works for clients (requesters/consumers, see Definition 14), whereas the *resource manager* works for resource providers. Both components negotiate with each other.

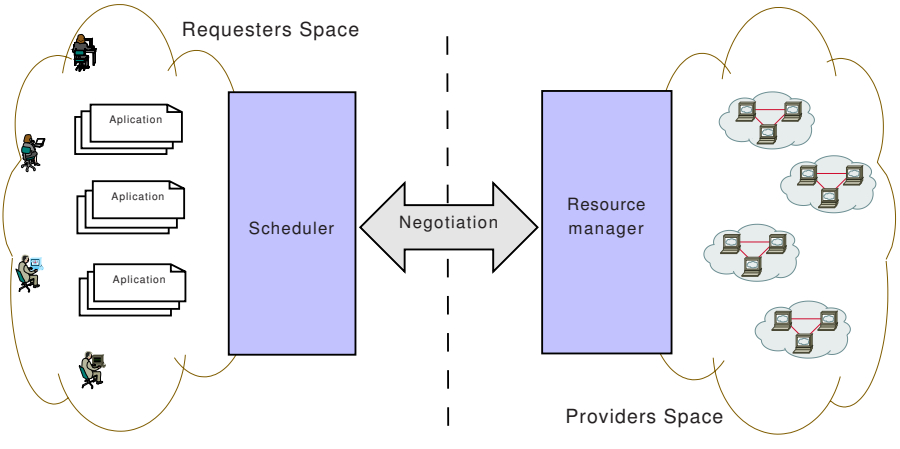


Fig. 3.1. Grid Scheduler and Resource manager Negotiation.

The *Resource discovery, selection and provisioning* is referred to as *resource management* and brokerage (Definition 17). It is an integral part of a Grid that is manual or semi manual in existing systems such as [91, 119, 97]. However, an automatic resource brokerage is very important for a scalable and adaptable Grid. The resource management must provide *resource selection* with automatic discovery of resource capabilities, automatic resource matching, and *capability and integrity checking* based on various static and dynamic resource

information advertised by *providers* and quality of service (QoS)-constraints set by resource requesters i.e. *clients* (Definition 14).

This chapter presents an architectural overview of *GridARM* along with the design and implementation of a scalable and extensible mechanism for resource selection and brokerage in the Grid. The selection is done based on client requirements or goals whereas extensibility is provided with a flexible mechanism to plug-in new algorithms for resource selection and new drivers for information discovery and retrieval from various Grid information services (GIS) [39, 10]. A distributed infrastructure is provided that scales well with an evolving Grid.

Definition 24. Scalability *is a quality of service that indicates the ability of a system to maintain its performance with increasing or decreasing number of resources and clients. In the Grid, it is an ability to preserve processing speed when both problem size and node size (machine size) increase.*

The distributed infrastructure of *GridARM* has been designed and implemented based on the superpeer model [169] with the support for self management and fault tolerance (Section 3.4.2). The system remains available and functional even if some of the resources (Grid nodes or services) stop working.

The automatic selection for the Grid is necessary not only because of its usability, efficiency and low cost but also because domain-specific users neither have enough time to make manual selections among various alternative options nor they possess enough knowledge about quantity and quality of huge number of resources. The Grid resources and contending applications are evolving so rapidly that it is increasingly difficult to make an optimal selection manually. Besides this, an automatic and scalable selection reduces involvement of node administrators as well.

As stated earlier, the *resource manager* works as a *resource broker* therefore it is its responsibility to make a fairly optimal allocation (Definition 20) of currently available resources to clients. It is necessary that available resources are distributed among competing applications or clients according to the proportional share (Section 3.3.4) of resources that are contributed to the Grid. For instance, if a resource contributes more computing power in terms of higher number of CPUs with better processor clock speed, then that resource shares more and thus is a candidate for a relatively higher client proportion. A proportional share-based resource allocation results in an optimal resource utilization and the steadiness in the system.

Currently, there is no widely deployed single resource management for the Grid that supports these functionalities all together. *GridARM* is developed based on off-the-shelf technologies while being capable to adapt new emerging technologies.

3.2 Architectural Overview

This Section gives a general overview of *GridARM*. Physical resource management and brokerage is described in detail in the following sections, whereas management of logical resources is covered in the following chapters.

GridARM is a WSRF (Section 2.3.2) compliant distributed service-oriented management system for the Askalon Grid environment (Section 2.4). As depicted in Figure 3.2, it consists of three loosely coupled distributed components. These components are responsible for (physical) resource management or brokerage (Section 3.3), agreement management (Chapter 5), and activity management (Chapter 4) respectively.

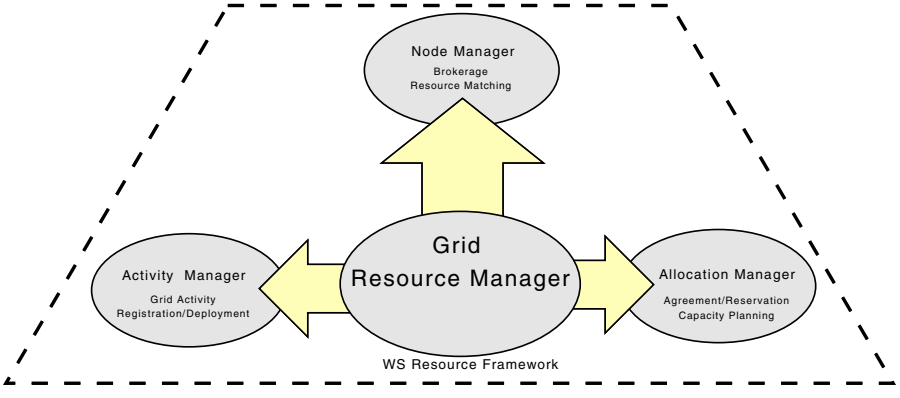


Fig. 3.2. GridARM System Architecture.

3.2.1 Resource Management

The *resource manager* is a main front-end service that is assisted by services for node, activity and agreement management as illustrated in Figure 3.2. It makes an efficient and smart use of other services, which can be registered and managed dynamically. It also works as a *Co-allocation manager* that can perform advance reservation of multiple resources on request. High level interfaces are provided for resource discovery, selection, allocation and management. Selection is performed by interacting with *node manager* that results in a selection of a resource ensemble based on a resource request, whereas allocation is performed by interacting with allocation manager. The input provided by a client (scheduler) is examined and an appropriate operation is invoked.

If a *resource manager* can not find suitable resources, it can interact recursively to *resource manager*s deployed in other part of the Grid in order to retrieve required resources (Section 3.4.2).

3.2.2 Node Management

Node manager is a Grid service that assists the front-end Grid resource manager for the discovery, selection, and allocation of physical resources, particularly nodes (Definition 11), available in the Grid.

3.2.3 Activity Management

An activity management framework (a.k.a *GLARE*) is provided as part of *GridARM* for dynamic registration, automatic deployment and on-demand provision of Grid activities.

In contrast to most of the existing Grid resource management systems which focus on physical resources, *GridARM* covers logical resources i.e. activities as well. It represents application components in terms of abstract or semantic descriptions in the form of *activity types* and concrete descriptions in the form of *activity deployments*. By separating activity types from deployments, it shields the Grid details from end users (clients), and performs automatic provisioning of deployed components by mapping activity types to a set of activity deployments. For instance, this allows a user to specify a workflow in terms of activity types, independent of the underlying hosting platform, implementation details of the activities, and the state of the Grid. *GLARE* is described in detail in Chapter 4.

3.2.4 Allocation Management

Allocation management improves on-demand provisioning of Grid resources by providing optimal allocations with advance reservations and capacity planning. The *GridARM allocator (allocation manager)* introduces a flexible mechanism to plugin new algorithms for allocation offer generation with a fair load distribution or an optimal resource utilization [156] (Chapter 6). In the *allocation offer generation* process, the *allocator* optimizes or fairly distributes resource capacity without compromising over client requirements. Free slots which are in accordance with the allocation policies are offered on request. The advance reservation with allocation algorithms and a practical solution for agreement enforcement is presented in Chapter 5 whereas optimized resource utilization with capacity planning and management is discussed in Chapter 6.

3.3 System Model

This section describes the *GridARM* model and internal components which deal with automatic brokerage of Grid resources, particularly Grid *nodes* $\in \mathcal{G}$ (Definition 11). *GridARM* consists of a *discoverer*, a *candidate set generator*

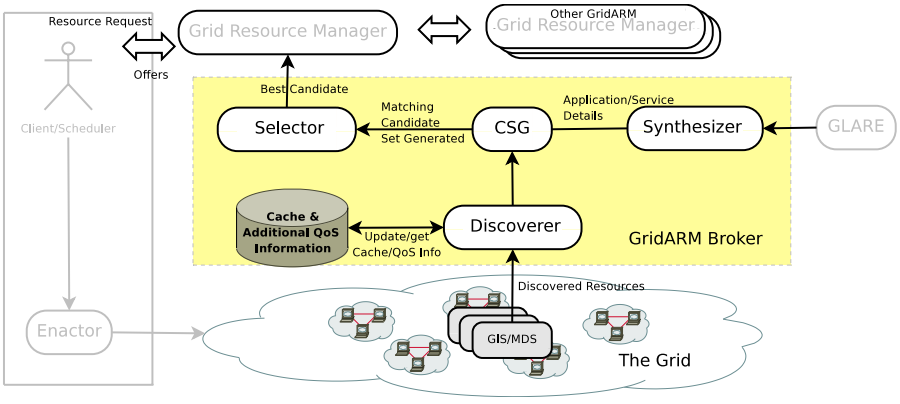


Fig. 3.3. *GridARM* system architecture.

(CSG), a *synthesizer* and a *selector* service, as depicted in Figure 3.3. The *discoverer* collects and congregates resources from different information services, *CSG* generates a set of candidates based on client requirements, the *synthesizer* updates candidates (specifications) with required logical resources, and the *selector* performs final selection of a best candidate. Following sub-sections describe these components in detail.

3.3.1 Resource Discoverer

A *discoverer* interacts with different types of Grid Information Services (GIS), for instance Globus monitoring and discovery service (MDS2 [39], and WS-MDS [10]), and Network Weather Service (NWS) [190], and collects information about available resources (nodes) in the Grid. Different information services support different representations and look-up mechanisms, and a client has to deal with them separately in a non-trivial time-consuming way. To overcome this problem, a generalized interface is introduced for heterogeneous information services with a specific driver (plug-in). The driver interacts with the service through service-specific interfaces, collects resource information in service-specific representation, and then transforms to a standard information representation model, that is the GLUE schema [85]. A set of information services $\mathcal{IS}$ is available in the Grid.

A client can make a query for resources based on both static and dynamic attributes, for instance, *total CPUs*, *operating system name/type* are static information whereas *free CPUs*, *average load*, *available memory* etc. represent dynamic information that is updated and discovered at runtime. The set $\mathcal{IT}$ contains all *information types* that may be associated with a node $\in \mathcal{G}$ to describe its capabilities. There is a set $\mathcal{IT}_i = \{\mathtt{it}_{i1}, ..., \mathtt{it}_{ik_i}\}$ of information types that is provided by $\mathtt{is}_i$, where k_i may be different for different $\mathtt{is}_i$. An information type $\mathtt{it} \in \mathcal{IT}$ is a class of information that covers related

attributes. For instance, *operating system* $\in \mathcal{IT}$ covers *os name*, *os type*, *os version* and *os release* attributes.

It is likely that different information services $\in \mathcal{IS}$ provide different kinds of resource information. For instance, MDS provides static or semi-dynamic information such as *OS* and *Processor* details whereas NWS provides more dynamic information such as *free CPUs*, *free memory* etc. Once discovered, all kind of information are consolidated (merged) in a resource description and represented in a unique format.

Definition 25. *The* resource discovery *is a process of finding information types* $\in \mathcal{IT}$ *from different information services* $\in \mathcal{IS}$*, consolidating information in the form of resource descriptions and finally making resources available for selections. Let* $\texttt{lookup}(\texttt{is}_i)$ *be a function that finds information type* $\mathcal{IT}_i \in \mathcal{IT}$ *available from an information service* $\texttt{is}_i \in \mathcal{IS}$ *such that* $\mathcal{IT} = \sum_{i=1}^{m} \mathcal{IT}_i \equiv \sum_{i=1}^{m} \texttt{lookup}(\texttt{is}_i)$*, then the* discovery function δ *is defined as* $\delta : \mathcal{IT} \mapsto \mathcal{G}$*, that consolidates (merges) all looked up information in resources* $\in \mathcal{G}$*.*

Algorithm 1 describes discovery algorithm along with steps taken for discovering and consolidating resource descriptions. As depicted, the *discoverer* iteratively looks up resource descriptions with different information types from all registered information services. Finally it consolidates all kind of information $\in \mathcal{IT}$ in the form of resource descriptions. Example 3 describes an example for discovery function δ.

Algorithm 1 Resource discovery and description consolidation Algorithm.

discover: $\delta()$
Input: $\mathcal{IS}$ // *A set of Grid information services (configurations)*
Output: $\mathcal{G}$ // *A set of Grid nodes (descriptions) available in the Grid*
$\mathcal{G} := \{\oslash\}$; // *Initially empty set of nodes*
for all $gis_i \in \mathcal{IS}$ **do**
  $\mathcal{IT}_i := lookup(gis_i)$; // *discover resources* $\mathcal{G}_{gis}$ *from gis*
  $\mathcal{G}_i := consolidate(gis_i)$; // *Consolidate information and make* $\mathcal{G}_i$ *found in* $\texttt{is}_i$
  for all $\mathtt{g} \in \mathcal{G}_i$ **do**
    if $\{\mathtt{g} \in \mathcal{G}_i\}$ **then**
      // $\mathtt{g}$ *is already discovered and exists in* $\mathcal{G}$
      $\delta^{'} := desc(\mathcal{G}_i, \mathtt{g})$; // *new information found from* $\texttt{is}_i$ *for* $\mathtt{g}$
      $\delta^{''} := desc(\mathcal{G}, \mathtt{g})$; // *Previously discovered information*
      $\mathtt{g} := consolidate(\delta^{'}, \delta^{''})$; // *node with updated (merged) information*
    end if
    $\mathcal{G} := \mathcal{G} + \{\mathtt{g}\}$; // *Add in* $\mathcal{G}$
  end for
end for
cache($\mathcal{G}$); // *Cache* $\mathcal{G}$ *for faster selection*
return $\mathcal{G}$;// *Return discovered resources* $\mathcal{G}$

Example 3 (Resource discovery and consolidatation).
Let two information services $\mathcal{IS} = \{\mathtt{is}_1, \mathtt{is}_2\}$, as depicted in Figure 3.4, gives different set of resources (nodes) as well as different types of information for same nodes. $\mathtt{is}_1 \in \mathcal{IS}$ provides 3 nodes $\{B, C, D\} \subset \mathcal{G}$ with additional information about CPUs $\in \mathcal{IT}$ whereas $\mathtt{is}_2 \in \mathcal{IS}$ provides 3 nodes $\{A, B, D\} \subset \mathcal{G}$ with additional information about OS$\in \mathcal{IT}$. After discovery and consolidation (δ), according to the Algorithm 1, a complete set of nodes $\mathcal{G} = \{A, B, C, D\}$ as well as consolidated information, that is, information about *CPUs* and *OS*, originally provided by different services, are made available as if they are from the same service. □

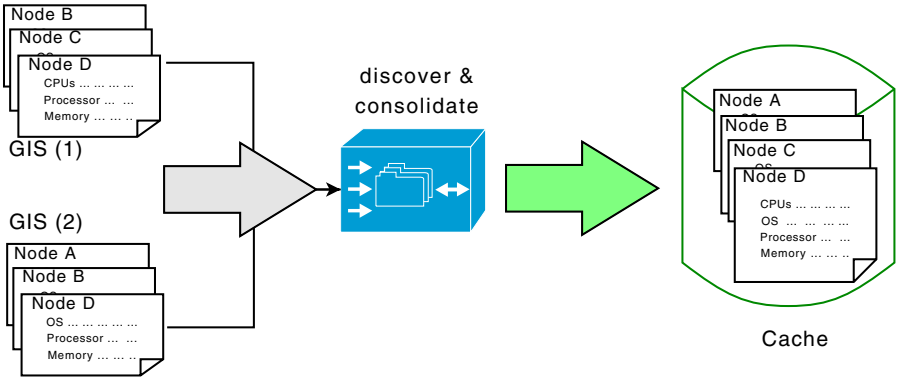


Fig. 3.4. Resource discovery and consolidation.

Besides off-the-shelf information services, *GridARM* introduces a *superpeer model*-based information service for resource discovery. The *superpeer model* is described in Section 3.4.2.

Resource Caching

The *discoverer* keeps a cache of discovered resources and then periodically refreshes the cache. Since a reasonable set of information (such as OS and processor details) is static and there is no need to `lookup` again and again, therefore the dynamic information such as *free CPUs* or *available memory* is refreshed in the cache more frequently than the static information. This improves response time and resource availability.

3.3.2 Candidate Set Generator

The *candidate set generator (CSG)* performs resource matching with client requirements. The solicited resource descriptions, congregated by the *discoverer* from the underlying information services $\in \mathcal{IS}$, are used. Once resources

(nodes) $\in \mathcal{G}$ are discovered, the *candidate set generator* filters out resources which do not match with client requirements, and generates a set of candidates that is offered to the client. The user requirements are represented in the form of a set of constraints $\in \mathcal{T}$ as part of the *resource request* $\mathtt{q} \in \mathcal{Q}$.

Definition 26. *A* candidate $\mathtt{b}_i \in \mathcal{B} | \mathcal{B} \subseteq \mathcal{G}$ *is a* node $\mathtt{g}_i \in \mathcal{G}$ *(Definition 11) that is available for* allocation *to a request* $\mathtt{q}_i \in \mathcal{Q}$ *with constraints (terms)* $\mathcal{T}_i \subseteq \mathcal{T}$ *such that all constraints* $\in \mathcal{T}_i$ *are fulfilled. For each request* $\mathtt{q}_i$ *there is a set* $\mathcal{B}_i = \{\mathtt{b}_{i,1}, ..., \mathtt{b}_{i,n_i}\}$ *of candidates generated by CSG, where* n_i *may be different for different* $\mathtt{q}_i$*.*

The candidates $\in \mathcal{B}_i$ are ordered and the ordering is performed based on a specific *selection criteria*. A selection criteria is an expression that is provided either by a client as part of its request, or system (*GridARM*) as part of its (default) configurations. For this purpose, a candidate selection language is used. This language is proposed by the OGSA-Resource Selection Service working group (RSS-wg) at Open Grid Forum [81]. According to the RSS proposed schema, a client can specify an expression that is evaluated and used by the *CSG* for sorting out candidates. The *best candidate* is always available at top of the sorted list.

Example 4 (CSG Selection criteria $(1 + freeCPUs \times clockSpeed)$*).*

```
1 <sum>
2   <constant>1</constant>
3   <product>
4     <select xpath="candidate/info/freeCPUs"/>
5     <select xpath="candidate/processor/clockSpeed"/>
6   </product>
7 </sum>
```

□

A sample *candidate selection criteria* could be expressed as shown in Example 4. The expression shows that the generated set of candidates should be sorted out according to the *free CPUs* and processor *clock speed*. Let $\mathtt{tcpu}(\mathtt{g}_i)$, $\mathtt{fcpu}(\mathtt{g}_i)$ and $\mathtt{clock}(\mathtt{g}_i)$ be the functions that specifies total CPUs, free CPUs, and *clock speed* of $\mathtt{g}_i$ respectively, then a candidate $\mathtt{b}_i$ is a *best candidate* if

$$(1 + \mathtt{fcpu}(\mathtt{b}_i) \times \mathtt{clock}(\mathtt{b}_i) = \top).$$

However, the selection criteria is configurable and a client can specify its own expression for selection criteria or a provider can modify the default one.

Definition 27. *A candidate set generation function* $\varsigma : \mathcal{G} \mapsto \mathcal{B}_i$ *is a partial function that generates an ordered list of candidates* $\mathcal{B}_i$ *such that each* $\mathtt{b} \in \mathcal{B}_i$ *fulfills constraints* $\mathcal{T}_i \subseteq \mathcal{T}$ *of a request* $\mathtt{q}_i \in \mathcal{Q}$ *and the first candidate is the* best candidate *according to the selection criteria. Let* $\mathtt{rank}(\mathtt{b}_i)$ *define the*

candidate selection criteria *of a candidate* $\mathtt{b}_i \in \mathcal{B}_i$ *then* $\mathtt{b}_i$ *is a* best candidate *if and only if* $\mathtt{rank}(\mathtt{b}_i)$ *is highest among all candidates* $\in \mathcal{B}_i$*, that is*

$$\mathtt{rank}(\mathtt{b}_i) = \top \iff \forall_{\mathtt{b}_j \in \mathcal{B}_i} \mathtt{rank}(\mathtt{b}_j) \not> \mathtt{rank}(\mathtt{b}_i), j \neq i$$

Algorithm 2 gives the pseudo code of *candidate set generation* ς that is explained in Example 6.

Algorithm 2 The Candidate Set Generation Algorithm.

generateCandidateSet: $\varsigma()$
Input: $\mathcal{G}, \mathtt{q}$ // *A set of Grid nodes* $\mathcal{G}$ *and a resource request* $\mathtt{q} \in \mathcal{Q}$
Output: $\mathcal{B}$ // *An ordered set of candidates* $\mathcal{B}$
$\mathcal{B} := \{\oslash\}$; // *Initially empty set of candidates*
$\mathcal{T}$:= set of constraints in request $\mathtt{q}$
for all $\mathtt{g} \in \mathcal{G}$ **do**
  if $\forall_{\mathtt{t} \in \mathcal{T}}$ $\mathtt{t}$ matches to $\mathtt{g}$ **then**
    // *node* $\mathtt{g}$ *does qualify constraint* $\mathcal{T}$
    $\mathcal{B} := \mathcal{B} + \{\mathtt{g}\}$; // $\mathtt{g}$ *is a candidate*
  end if
end for
$\mathcal{B} :=$ **concretize**$(\mathcal{B})$;// *add activity deployments for requested activity type*
$\mathcal{B} :=$ **sort**$(\mathcal{B})$; // *sort according to the candidate selection criteria*
return $\mathcal{B}$;// *Return generated candidate set* $\mathcal{B}$

Example 5 (Resource Request (rr)).

```
1  <ResourceRequest name="rr">
2    <CPUArchitecture>
3      <CPUArchitectureName>x86_64</CPUArchitectureName>
4    </CPUArchitecture>
5    <TotalCPUCount>
6      <LowerBound>1</LowerBound>
7    </TotalCPUCount>
8    <Processor>
9      <ClockSpeed>
10       <UpperBound>4.0 </UpperBound>
11       <LowerBound>2.0</LowerBound>
12     </ClockSpeed>
13   </Processor>
14   <OperatingSystem>
15     <OperatingSystemType>Linux</OperatingSystemType>
16   </OperatingSystem>
17 </ResourceRequest>
```

□

Example 6 (Candidate set generation and selection).
In the *Resource Request* ($\mathcal{Q} = \{rr\}$) listed in Example 5 in which a client requests for node with *x86_64* architecture, *Linux* operating system, at least 1 CPU, and processor clock speed between $2.0 - 4.0GHz$. Let $\mathcal{G} = \{A, B, C, D, E, F\}$ be the currently discovered state of the Grid, as shown in Table 3.1 then by applying Algorithm 2, that is $\varsigma(\mathcal{G}, rr)$, the generated candidate set $\mathcal{B}_{rr} = \{D, C, F\}$. The node D is the *best candidate* according to the selection criteria (`rank`) given in Example 4 as $\texttt{rank}(D) = 180 > \texttt{rank}(C) \wedge \texttt{rank}(D) > \texttt{rank}(F)$. The ranks are computed according to the selection criteria given in Example 4. □

Table 3.1. A sample Grid with a set of nodes and associated attributes.

Node	A	B	C	D	E	F
tcpu	32	132	8	64	128	12
fcpu	16	39	7	49	96	5
clock	3.9	1.4	3.0	2.8	3.2	2.0
ostype	Linux	Linux	Linux	Linux	Solaris	Linux
arch	ia64	x86_64	x86_64	x86_64	x86_64	x86_64
rank	-	-	25	180	-	25

3.3.3 Resource Synthesizer

The *synthesizer* further filters-out generated candidates according to the requested activity deployments (Definition 8) and adds details of the deployment description in the filtered candidates. For instance, candidates set generated by the *candidate set generator (CSG)* fulfills all constraints associated with physical resources such as *free CPUs*, *operating system*, *free memory*, *reliability* etc. whereas *synthesizer* checks with the help of *GLARE* (Chapter 4) whether or not a client's required *activity* $\in \mathcal{A}$ has *activity deployments* installed/deployed on the candidate node $\in \mathcal{G}$ and if deployed then consolidate deployment descriptions with the candidate descriptions. The synthesize algorithm is discussed in Chapter 4.

3.3.4 Resource Selector

The *resource selector* is the third component of *GridARM* that filters out the generated set of candidates and makes an optimal selection. The resource selection has two perspectives: a client's perspective and a resource perspective.

Client Perspective

For the client's perspective, the selection function $\xi : \mathcal{Q} \mapsto \mathcal{B}_{best}$ selects a *best candidate* $\mathtt{b}_i \in \mathcal{B}_i$ for each request $\mathtt{q}_i \in \mathcal{Q}$. $\mathcal{B}_{best}$ is produced with a

reduction operator Ψ such that $\mathcal{B}_{best} = \Psi(\mathcal{B}_1, ..., \mathcal{B}_m) = (\mathtt{b}_1, ..., \mathtt{b}_m), \mathtt{b}_i \in \mathcal{B}_i, m \in \mathbb{N}$, where $\mathcal{B}_i$ is an ordered set of candidates for request $\mathtt{q}_i$ and $\Psi(\mathcal{B}_i) = \mathtt{b}_i | \mathtt{rank}(\mathtt{b}_i) = \top$, that is, it gives highest ranked candidate.

Resource Perspective

With resource perspective, the *resource selector* filters out the generated set of candidates according to their proportional shares. For this purpose, a utility function (Definition 18) is introduced (Section 2.6) that is used to optimize proportional share-based resource allocation. Since the generated set of candidates $\mathcal{B}_i$ fulfills all constraints of the request $\mathtt{q}_i$, the *selector* may offer additional benefits to the client by making a fair allocation. For instance, if a client is offered a better *clock speed* or *free CPUs* than requested, then it may get benefit of additional capacity as long as there is no further request for the same resource by any other client.

The functions $\mathtt{tcpu}(\mathtt{g}_i), \mathtt{fcpu}(\mathtt{g}_i)$ and $\mathtt{clock}(\mathtt{g}_i)$ can be used to drive *total CPUs*, *free CPUs*, and *processor clock speed* respectively of a node $\mathtt{g}_i$. These functions are used in selection function ξ to evaluate the proportional share of a node that it offers to a client in addition to minimum requirements of the client. The selection function ξ can be redefined as follows:

Definition 28. *The* resource selection *is a process of allocating qualified candidates $\mathcal{B}$ to $\mathcal{Q}$ according to their proportional share. The selection function $\xi : \mathcal{S} \mapsto \mathcal{Q}$ assigns a selected candidate $\mathtt{b}_i \in \mathcal{B}_i$ to a request $\mathtt{q}_i \in \mathcal{Q}$, where $\mathcal{S} = \Psi(\mathcal{B}_1, ..., \mathcal{B}_m) = (\mathtt{b}_1, ..., \mathtt{b}_m)$ such that, $\Psi(\mathcal{B}_i) = \mathtt{b}_i \iff \mathcal{U}(\mathtt{b}_i) = \top$.*
Here Ψ is a set reduction operator whose domain is the set of all candidate sets and codomain is a set of all selected candidates. It selects a candidate $\mathtt{b}_i \in \mathcal{B}_i$ that has maximum utility*, that means $\forall_{\mathtt{b} \in \mathcal{B}_i} \mu(\mathtt{b}) \not\succ \mu(\mathtt{b}_i)$.*

Proportional Share

As described in the previous section, the proportional selection is done based on an *objective function*, that is to be maximized. Before proceeding to the proper definition of the utility used as *objective function* in the resource selection context, first lets describe its defining attributes: *node share*, *node power* and *node offering*.

The *node share* $\mathtt{share}(\mathtt{g}_i)$ is a relative measure of contribution (proportional share) that a node $\mathtt{g}_i \in \mathcal{G}$ contributes to the total capacity of the Grid in the form of $\mathtt{tcpu}(\mathtt{g}_i)$, that is

$$\mathtt{share}(\mathtt{g}_i) = \frac{\mathtt{tcpu}(\mathtt{g}_i)}{\sum_{\mathtt{g} \in \mathcal{G}} \mathtt{tcpu}(\mathtt{g})}$$

A *node power* $\mathtt{power}(\mathtt{g}_i)$ is a measure of the computing power of a node $\mathtt{g}_i \in \mathcal{G}$ in the form of *processor clock speed* $\mathtt{clock}(\mathtt{g}_i)$ relative to the maximum clock speed of any node $\in \mathcal{G}$, that is

$$\texttt{power}(\texttt{g}_i) = \frac{\texttt{clock}(\texttt{g}_i)}{max(\bigcup_{\texttt{g}\in\mathcal{G}} \texttt{clock}(\texttt{g}))}$$

The *node offering* $\texttt{offering}(\texttt{g}_i)$ is a measure of the remaining, available or free CPUs $\texttt{fcpu}(\texttt{g}_i)$ of a node that is currently being offered relative to the total capacity $\texttt{tcpu}(\texttt{g}_i)$ of a node $\texttt{g}_i$ that is contributed in the Grid.

$$\texttt{offering}(\texttt{g}_i) = \frac{\texttt{fcpu}(\texttt{g}_i)}{\texttt{tcpu}(\texttt{g}_i)}$$

The values of of *total CPUs* (`tcpu`), *free CPUs* (`fcpu`), and *clock speed* (`clock`) can be retrieved from a Grid information service $\in \mathcal{IS}$.

Definition 29. *The proportional share-based* utility $\mathcal{U}$ *as an objective function of resource selection problem is defined in terms of* $\texttt{share}(g_i)$, $\texttt{power}(g_i)$ *and* $\texttt{offering}(g_i)$ *of a node* $g_i \in \mathcal{G}$. *Let*

$$\mathcal{U}^{'}(g_i) = \frac{W_{NS} \times \texttt{share}(g_i) + W_{NP} \times \texttt{power}(g_i) + W_{NO} \times \texttt{offering}(g_i)}{3} \tag{3.1}$$

then, the utility $\mathcal{U}(g_i)$ *of resource* g_i *is*

$$\mathcal{U}(g) = \frac{(1 - e^{-\mathcal{U}^{'}(g)})}{(1 - e^{-1})} \tag{3.2}$$

where W_{NS}, W_{NP} *and* W_{NO} *are the weight factors of* $\texttt{share}(g_i)$, $\texttt{power}(g_i)$ *and* $\texttt{offering}(g_i)$ *respectively which are selected as*

$$\frac{(W_{NS} + W_{NP} + W_{NO})}{3} = 1$$

and the default values of these weight factors are as:

$$W_{NS} = 1.0, W_{NP} = 0.5, W_{NO} = 1.5$$

This combination of weight factors is selected because it gives smaller error or 'deviation from the expected' load distribution. This is explained experimentally in Section 3.5. The utility $\mathcal{U}(g)$ *(Equation 3.2) decreases the value generated by* $\mathcal{U}^{'}(g)$ *with a little biasness towards most and least loaded nodes. The leastly and mostly loaded nodes get relatively less distribution. However, share is fairly distributed among nodes with average load. This is acceptable since least loaded node has less contribution whereas most loaded node has has higher contribution but has already got higher proportion of load.*

Algorithm 3 provides a pseudo code for the selection algorithm used by the *selector*. It gets a list of candidates from *CSG* and selects a candidate with maximum utility from each list. Example 7 demonstrates Algorithm 3.

Algorithm 3 The Selection Algorithm.

select: $\xi()$
Input: $\mathcal{Q}, \mathcal{G}$ // *A set of Grid nodes* $\mathcal{G}$ *and a set of resource requests* $\mathcal{Q}$
Output: $\mathcal{S}$ // *Selected list of candidates*
$\mathcal{S} := \{\oslash\}$; // *initially empty selection*
for all $r \in \mathcal{Q}$ **do**
  $\mathcal{B}_r := generateCandidateSet(r, \mathcal{G})$;
  $g := \oslash$; // g *to be initialized to resource with best utility*
  for all $c \in \mathcal{B}_r$ **do**
    if $\mathcal{U}(c) > \mathcal{U}(g)$ **then**
      $g := c$; // c *has better utility*
    end if
  end for
  $\mathcal{S} := \mathcal{S} + \{g\}$;
end for
return $\mathcal{S}$;// *return selected candidates*

Example 7 (Candidate Selection).
Table 3.2 shows the utility of nodes specified in Table 3.1. By applying selection function as demonstrated in Algorithm 3, the selected candidate $\mathtt{b}$ is $\mathtt{b} = \xi(\mathcal{G}, \mathcal{Q}) = \{C\}$. As shown in Table 3.2, the utility $\mathcal{U}(C)$ is better than for other candidates $\in \mathcal{B}_{rr}$. Furthermore, its worth noting that the node $D \in \mathcal{B}_{rr}$ is considered best node based on user's selection criteria but with utility-based selection, the client is offered node C instead of D. This not only improves balance load distribution but also the client gets a node with better clock speed and less number of concurrent clients (only one CPU is under use on node C). □

Table 3.2. The utility of the nodes specified in Table 3.1.

Node	A	B	C	D	E	F
share	0.085	0.35	0.02	0.02	0.34	0.03
power	1	0.36	0.77	0.72	0.82	0.51
offering	0.5	0.3	0.88	0.77	0.75	0.42
util	n/a	n/a	0.692	0.634	n/a	0.416

Besides the *utility function*, *GridARM* uses an internal ranking of nodes that represents stability and availability of nodes. Let $\mathtt{uptime}(\mathtt{g}_i)$ be the uptime of $\mathtt{g}_i$ in milliseconds then:

$$\mathtt{rank}(\mathtt{g}_i) = (\mathtt{share}(\mathtt{g}_i) + \mathtt{power}(\mathtt{g}_i)) \times \mathtt{uptime}(\mathtt{g}_i) \tag{3.3}$$

This is not an ideal formula, however, a higher rank shows a highly available machine. In case of equal ranks of multiple nodes, then the ranking is done

using hash codes of the node names. This is to ensure uniqueness of ranks across the Grid.

3.3.5 A Steady System with Proportional Distribution

The resource selection based on *resource utility* as defined in Definition 29, ensures that client load distribution is done according to the proportional share of each resource in the Grid. A resource is requested to execute an activity (Definition 6) or an application on a selected node, and a selection or allocation is only done if the resource has some free capacity. Let

- e be the expected or requested *execution time*
- c be a factor that represents contention as the resource may be shared. $c > 1$ if there is a contention.
- d be the delay due to wait time in the queuing system.

Lemma 1. A load distribution according to the proportional share of resources improves system steadiness and optimizes resource allocations.

Proof. Lets adopt the usual definition for the standard deviation *(σ) Without proportional distribution the* standard deviation *of execution time is*

$$\sigma(e \times c + d)$$

Since there is no contention with proportional distribution that means the $c \to 0$ *and* $d < 1$*, in this case the* standard deviation *is*

$$\sigma(e)$$

it can easily be observed that

$$\sigma(e \times c + d) \geq \sigma(e) \tag{3.4}$$

Thus the *standard deviation* of execution time of a given request with proposed load distribution is always less than or equal to the *standard deviation* without proportional distribution. Reduced σ means greater *steadiness* and better optimization in allocations. □

3.4 Implementation

GridARM is implemented based on GT4 technologies. All components are developed as WS-Resources (Definition 12), that means they are stateful Web services. These services are configurable and customizable. An information service *plug-in* can be configured and loaded dynamically by the *discoverer* by providing a configuration file. For instance, a sample Grid information service configuration file is presented in Example 8. This configuration is represented

as XML document and it includes *address* of the service host, *port* on which service is listening, base distinguish name *basedDn* required for MDS2, *type* of information service, and *cacheEnabled* flag for enabling or disabling cache. The default GIS *type* is *MDS2* as it has been widely deployed in Grids.

Example 8 (Grid Information Service Config File).

```
1 <GISConfiguration xmlns="http://gridarm.askalon.org"
2         address="agrid.uibk.ac.at"
3         basedDn="mds-vo-name=local,o=grid"
4         name    ="agrid"
5         port    ="2170"
6         type    ="mds2"
7         cacheEnabled="true"
8 />
```

A graphical client application is developed that can be used to manually discover, select and browse underlying resources and their properties. The console application is integrated in the front-end Askalon development environment. Figure 3.5 depicts a snapshot of the *GridARM console*.

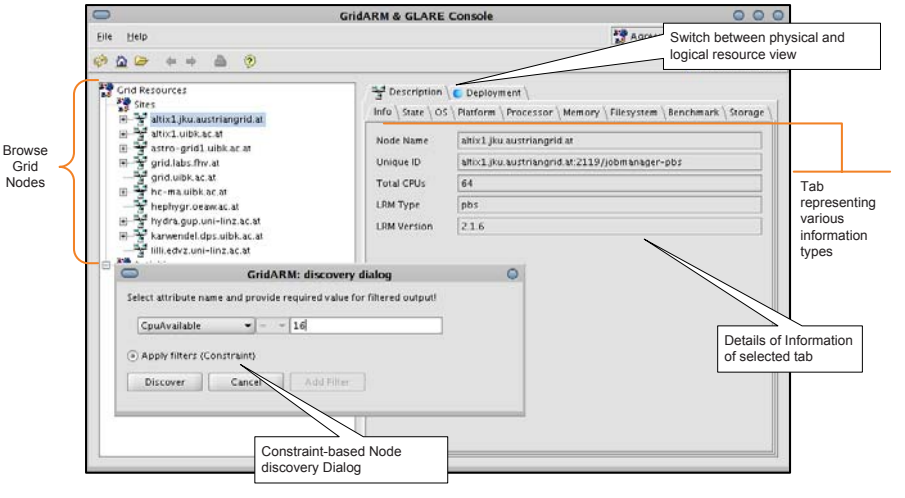


Fig. 3.5. The GridARM Console: A Graphical client application.

3.4.1 Customization

An interesting feature of *GridARM* is its customization. Customization (a) introduces new information services $\in \mathcal{IS}$ that represent resources (*nodes*) unavailable in any of the underlying information services $\mathcal{IS}$, (b) shields Grid

users from some of the existing nodes, and (c) optionally overrides or hides some information of a set of existing nodes $\subseteq \mathcal{G}$. This is not only useful for employing a coarse-grain access-control (by hiding/showing capacity of a node) but also helpful in preparing a Grid testbed according to the domain-specific experimental needs. The customization of *GridARM* is possible for each client and a customized *resource manager* can be instantiated against the Grid user's identity (proxy) (Section 2.3.3) as a non-persistent *WS-Resource* (Definition 12).

Example 9 (Configuration for a customized resource manager*).*

```
1  <CustomizedRMConfig xsi:type="CustomizedRMConfigType" ... >
2    <!--Customize a set of nodes filtered according to ResourceFilter-->
3    <ResourceFilter xsi:type="ResourceFilterType">
4      <ResourceConstraint name="TotalCPUs" maxValue="16" minValue="10"/>
5      <ResourceConstraint name="OSName" value="Linux"/>
6    </ResourceFilter>
7
8    <!-- Customize a specific named node -->
9    <AttributedNode nodeName="altix1.jku.austriangrid.at">
10     <!-- Overridden attributes of a named node -->
11     <NodeAttribute name="TotalCPUs" value="35"/>
12   </AttributedNode>
13
14   <AttributedNode nodeName="karwendel.dps.uibk.ac.at">
15     <NodeAttribute name="TotalCPUs" value="15"/>
16   </AttributedNode>
17
18    <expirationTime      xsi:type="xsd:dateTime">
19       2006-05-31T10:35:59.065Z                    </expirationTime>
20    <cacheEnabled       xsi:type="xsd:boolean"> true  </cacheEnabled>
21    <refreshAfter       xsi:type="xsd:duration">PT2H  </refreshAfter>
22    <ignoreOtherNodes  xsi:type="xsd:boolean"> false </ignoreOtherNodes>
23 </CustomizedRMConfig>
```

□

A sample configuration file for customization is described in Example 9 with following elements:

- *ResourceFilter:* element describes constraints that need to be satisfied in order to make a node visible to the clients. For instance in Example 9, only nodes with *Linux* OS and *TotalCPUs* between $10 - 16$ become visible.
- *AttributedNode:* element is used to override some of the attributes of a named node.
- The customization of nodes expires after *expirationTime*.
- If *cacheEnabled* is set the cache of nodes is refreshed after *refreshAfter* seconds.
- If *ignoreOtherNodes* is enabled then nodes other than *AttributedNode*s will be hidden.

3.4.2 Superpeer

GridARM works in a service-oriented (Section 2.6.4) interaction pattern based on a superpeer model (Section 2.6.4). This model provides an infrastructure in

which multiple nodes form smaller groups. The members of each group select one member as a group representative that is referred to as a *superpeer*. All *superpeers* collectively make a superpeer group. The intra-group interaction mechanism is service-oriented whereas a superpeer model is used for inter-group interaction.

In contrast to hierarchical or centralized models [39], the superpeer model works well with dynamic and large-scale distributed environments including the Grid [169]. This model makes *GridARM* scalable and a better load-balancer. The automatic formation of superpeer groups makes *GridARM* a self-managed system.

GridARM exploits the GT4 aggregation framework and the *default index* (a.k.a. WS-MDS or MDS4). In GT4, an information service that is local to a node is called *default index* whereas root *index* is referred to as a *community index*. The GT4-enabled resources register in a *community index*, and a *community index* may register itself with another *community index*. In this way, information about registered resources propagate toward *index service* in a hierarchical model. Figure 3.6 shows a typical hierarchical organization of WS-MDS. According to this configuration, there are 11 nodes ($C1 - C11$). The node $C4$ is *community index* of $C1$, $C2$ and $C3$ and in this way it is a smaller community (labeled as A) which belongs to a larger community (labeled as B). *Community index* at $C11$ has the entire view of the Grid, but the community indices at the node $C4$ and $C8$ are only aware of the underlying nodes registered with them.

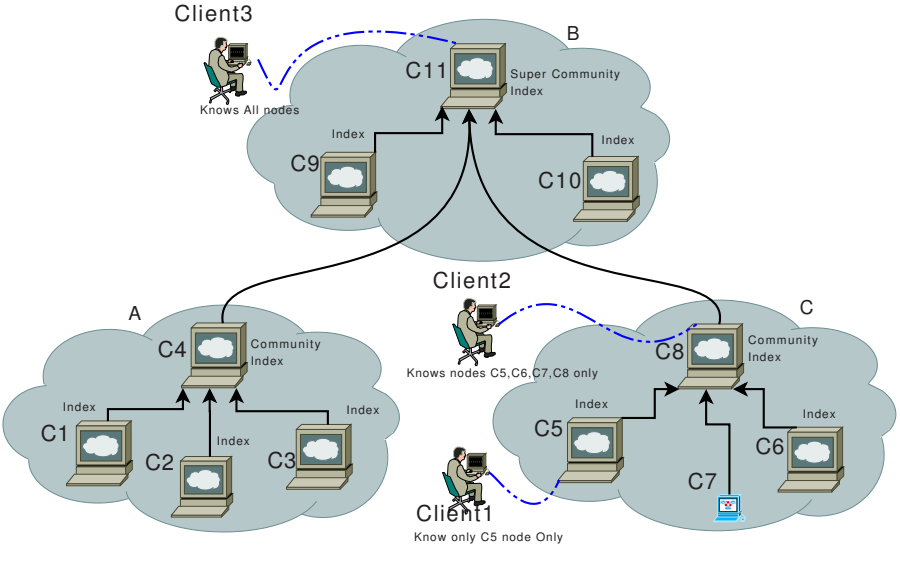


Fig. 3.6. Globus MDS-based hierarchical advertisement of nodes in the Grid.

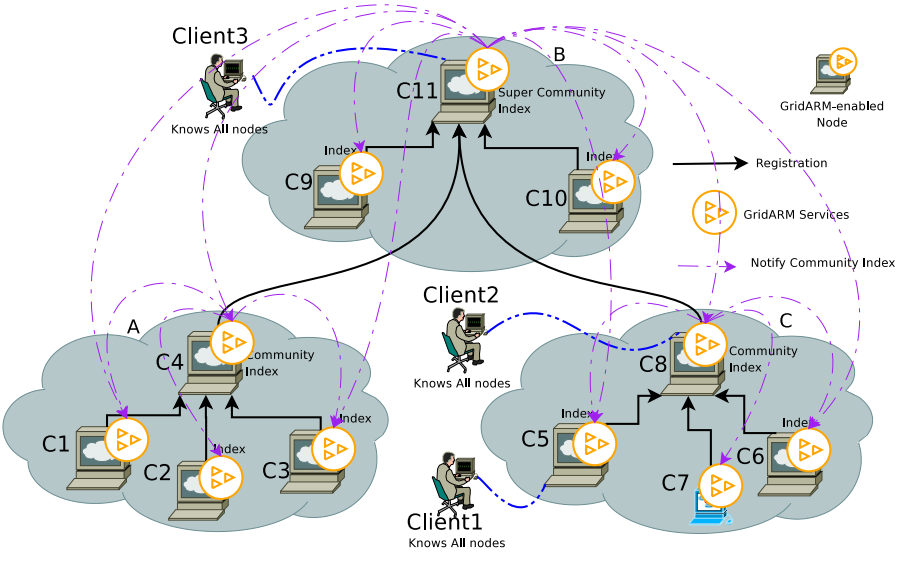


Fig. 3.7. Grid infrastructure with GridARM services and MDS hierarchy.

This hierarchical model has some limitations: with growing number of registered nodes, the *community index* may become a bottleneck. For instance, the client of node $C11$ (i.e. *client*3) knows all nodes in the Grid, but clients of node $C5$ (i.e. *client*1) has no knowledge about nodes other than the node $C5$. It has to know by some means the address of root index in order to have a global view of the Grid. Furthermore, there is no mechanism to split the *index service* load.

The new *superpeer model* is a remedy. The model is formed and scaled automatically and a client of any node may reach the entire Grid. This is explained in the following sub-sections.

Making of a Superpeer Group

The *GridARM* (resource manager) probes and identifies type of the *index service*. Once a *GridARM*-enabled node identifies itself as *community index* node, it becomes superpeer *election coordinator* and notifies all other nodes registered in the community index. As depicted in Figure 3.7, the node $C4$, $C8$ and $C11$ are identified as *community index* node, and the *GridARM* notifies all registered nodes about the discovery. The *GridARM*-enabled node $C4$ notifies nodes $\in \{C1,C2,C3\}$, node $C8$ notifies nodes $\in \{C5,C6,C7\}$ whereas $C11$ notifies all nodes $\in \{C1 - C10\}$. Notification is done twice (with a configurable time interval) and the second notification is acknowledged. A notification message includes number of registered nodes in the community index showing the community strength. In order to avoid multiple acknowledgments, a message from a smaller community is acknowledged if multiple

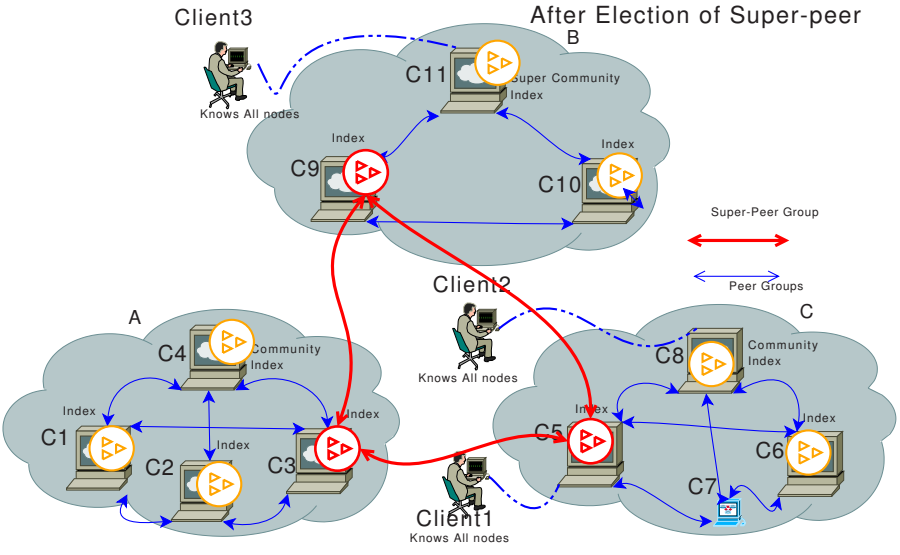


Fig. 3.8. Grid infrastructure after making of superpeers.

notifications are received. A responding node with higher *rank* is elected as a *superpeer*. Depending on the number of nodes, more than one nodes can be elected as superpeers and other members are then equally distributed among the elected superpeers. In this way each group can have exactly one superpeer node. Initially, these groups are made by the *election coordinator* who notifies all elected superpeers about their group members after the completion of their election.

Figure 3.8 depicts the structure of a Grid after making of superpeers with 3 groups and one superpeer group. Each member within a smaller group become peer of each other, whereas one member elected as superpeer from each group joins a superpeer group.

Role of a Superpeer

Once a node recognizes itself as a superpeer member after receiving notification from the *election coordinator*, it does the following:

- Discovers other superpeers distributed in a larger community by interacting with community or *super community* indices.
- Handles requests from group members. A superpeer is contacted when a member could not find resources according to the client requirements. It forwards requests to other superpeers and caches the results.

A service on a superpeer node is accessed when other members could not find required resources (nodes, activities etc.). The superpeer forwards unfulfilled requests to other superpeers and return the result if found. It also keeps cache

of the result found from other superpeers. Furthermore, a superpeer monitors the community index periodically to see if there is a new member. The new members are informed about the already elected superpeer.

Self Management

GridARM is self-managed and fault tolerant. It might be possible that a *superpeer* fails or become unavailable due to any reason. *GridARM* handles this situation nicely. Once a member discovers that the superpeer is not working, it immediately generates *ranks* of all member nodes excluding the missing superpeer and notifies the *highest ranked member*. The *rank* of a node is generated according to formula given in Equation 3.3. The highest ranked member then (a) verifies that the superpeer is missing (b) verifies its own rank and then (c) notifies all other member. As a result each member again verifies the unavailability of the superpeer and acknowledges back to the highest ranked node. An acknowledgment from a simple majority confirms that the superpeer is no longer available, and the highest ranked node takes over as a new superpeer. In this way election and re-election of superpeers takes place, and high availability and scalability of the distributed *GridARM* is ensured.

3.4.3 Standard Adaptation

The *GridARM* services are loosely coupled in a service-oriented fashion. This is in accordance to the Open Service-oriented Grid Architecture [71] (Section 2.3.1) proposed by the Open Grid Forum (OGF) [129] formerly known as Global Grid Forum (GGF) [81]. For better integration with the Askalon runtime environment, *GridARM* supports the Job Submission Description Language (JSDL) [100] as a query template which is synthesis by GridARM with matching resources. The JSDL provides a set of rich constructs for constraint specification. However, since JSDL is yet evolving, *GridARM* introduces a simple querying format in which different constraints are expressed (Example 6) and that can also be translated to LDAP [103] filters. LDAP is the default querying mechanism supported by the Globus MDS version 2.

3.5 Experiments and Evaluation

GridARM is developed based-on WSRF and uses mechanisms like subscription/notification and lifetime management of resources. It is deployed in the AustrianGrid [33] which consists of several Grid nodes with varying capacity contributions, having different architectures, operating systems, and various queuing system such as Portable Batch System (PBS) [11], Sun Grid Engine (SGE) [168] etc. The details of semi-persistent Austrian Grid testbed is given in Table 3.3. The Globus Toolkit (GT4) (Pre-WS) is installed on all Grid

Table 3.3. The Austrian Grid testbed.

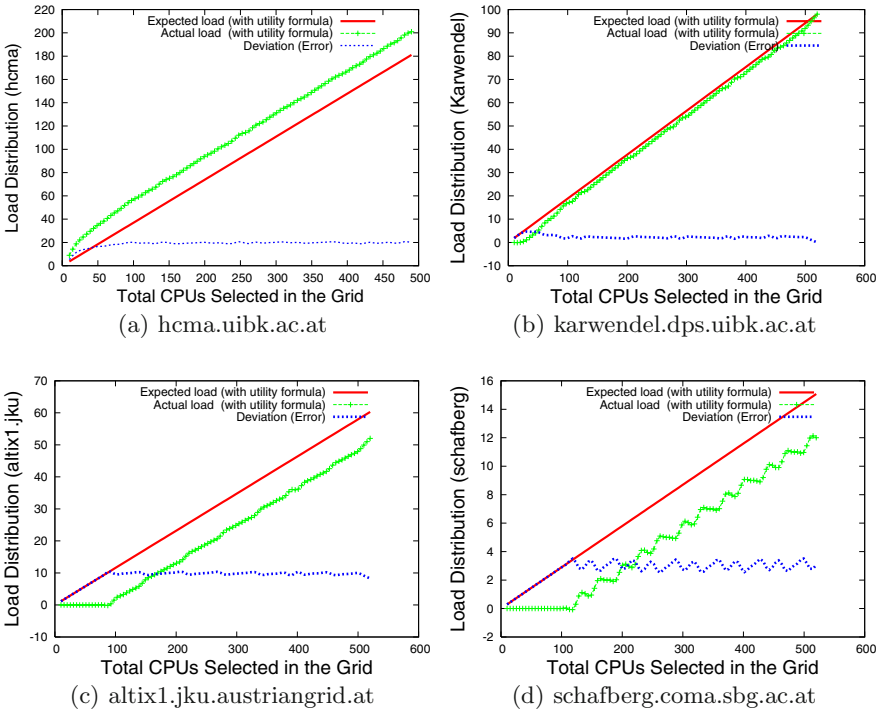
#	Node	#	CPU, Bit, GHz	RAM	Provider	Location
1	hc-ma.uibk.ac.at	204	Opteron, 64, 2.2	4096	GT2/SGE	Innsbruck
2	hephygr.oeaw.ac.at	84	Opteron, 64, 2.2	2048	GT2/torque	-
3	karwendel.dps.uibk.ac.at	104	Opteron,64,3.0	15026	GT2/SGE	Innsbruck
4	altix1.jku.austriangrid.at	64	Itanium2, 64, 1.6	61408	GT4/PBS	Linz
5	hydra.gup.uni-linz.ac.at	16	Athlon, 32, 1.6	2048	GT2/PBS	Linz
6	schafberg	16	Itanium2, 64, 1.6	15026	GT2/Fork	Salzburg
7	altix1.uibk.ac.at	16	Itanium2, 64, 1.6	15026	GT2/Fork	Innsbruck
8	grid.labs.fhv.at	12	Xeon,64,3.0	3986	GT2/SGE	Innsbruck
9	astro-grid1.uibk.ac.at	2	Opteron,64,2.2	11986	GT2/SGE	Innsbruck
10	agrid1.uibk.ac.at	21	Pentium4, 32, 1.8	512	GT2/PBS	Innsbruck

nodes whereas GT4 core (see Section 2.3.3 for GT4 details) is installed on some dedicated machines as part of Distributed and Parallel Systems (DPS) domain, University of Innsbruck - Austria. Apart from node specific services such as Grid Resource Allocation Manager (GRAM) [41] that works as the node gatekeeper, the Network Weather Service (NWS) [190] and MDS [39] version 2 with the Glue schema are also installed. The MDS service provides information of all AustrianGrid nodes whereas NWS was installed only within the DPS domain. All machines involved in the experiments were located on a lightly loaded network with a maximum latency between two computers of about 2 milliseconds.

First, the selection problem is analyzed with the help of uniform load distribution among available resources. The load distribution is done by selecting resources for clients according to the selection solution as described in Section 3.3.4. The experiments are performed in the Austrian Grid testbed with the relative contributions of each node in the Grid. Three perspectives are chosen, *utility-centric*, *offering-centric* and *share-centric*. For the *utility-centric* perspective the load distribution in which default weight factors (Section 3.3.4) are assigned to the resource. Table 3.4 depicts the combinations of weight factors for the three perspectives. These combinations are chosen in order to demonstrate the broader variation in the results with relatively fair load distribution. The combination for the *utility centric* perspective represents a combination with optimal load distribution.

Table 3.4. The Weight factors for resource selection.

Perspective	W_{NS}	W_{NP}	W_{NO}
Utility-centric	1.0	0.5	1.5
Capacity(offering)-centric	0.0	0.5	2.5
Share-centric	1.0	1.0	1.0



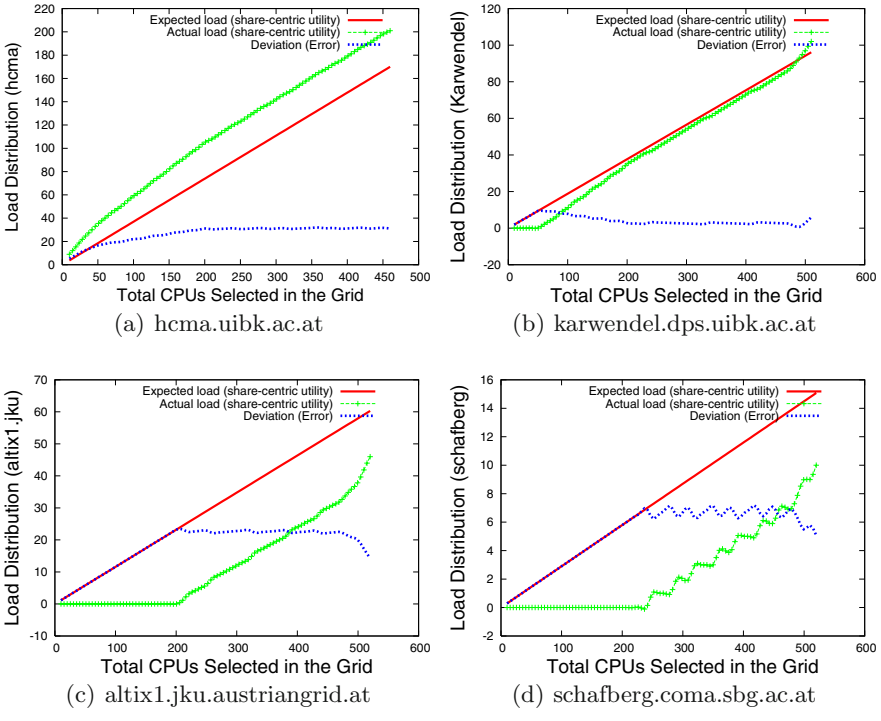
(a) hcma.uibk.ac.at (b) karwendel.dps.uibk.ac.at

(c) altix1.jku.austriangrid.at (d) schafberg.coma.sbg.ac.at

Fig. 3.9. Utility-centric Load distribution

A *resource request* (Definition 15) consists of a set of constraints, for instance, *free CPUS*, *free memory*, *operating system type*, *processor type*, *processor speed* etc. We have generated a random set of request with various constraints that are satisfiable within the Austrian Grid testbed (Table 3.3), however, the number of CPUs requested is always one. This makes sense since in the Askalon runtime environment (Section 2.4), a *resource request* is usually made separately for each activity (Definition 6) in a workflow (Definition 10).

Following is the comparison and evaluation of proportional distribution of load among the available Grid nodes as shown in Table 3.3 according the the three perspectives described above. The comparison is made between ideal (expected) and real (actual) values of the load share among the nodes. The expected value of load distribution is calculated according the formula shown in Equation 3.2 that gives an exact proportional load distribution. The load is distributed among all nodes available in the Austrian Grid (Table 3.3), however we have selected four nodes for graphical depiction of load distribution. These nodes includes:



(a) hcma.uibk.ac.at (b) karwendel.dps.uibk.ac.at

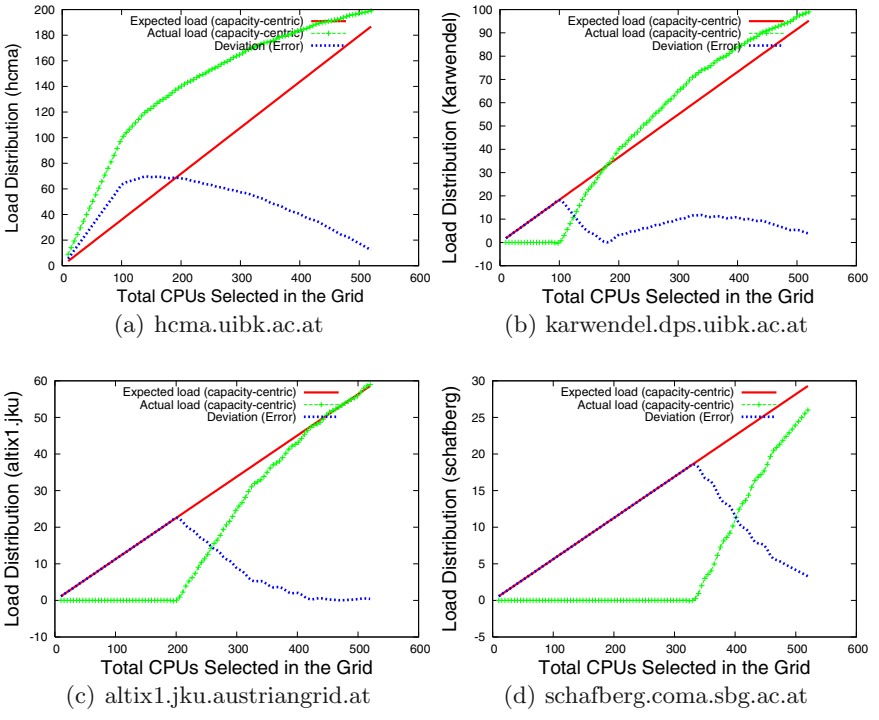
(c) altix1.jku.austriangrid.at (d) schafberg.coma.sbg.ac.at

Fig. 3.10. Share-centric Load distribution

1. `hcma.uibk.ac.at` with 204 CPUs
2. `karwendel.dps.uibk.ac.at` with 104 CPUs
3. `altix1.jku.austriangrid.at` with 64 CPUs
4. `schafberg.coma.sbg.ac.at` with 16 CPUs

The reason behind selection of these nodes is that they represent diverse configurations, and varying number of *node shares* and *node powers*. The graphs compare the expected load distribution with actual selection among four different types of nodes in the Austrian Grid. These nodes possess varying degree of *node share* and *node power* that gives a balanced view of the load distribution. Figures 3.9 3.10 3.11 show load distribution among the nodes shown above with the three perspectives for selection evaluation based on the weight factors in Table 3.4 and the utility formula given in Definition 29.

Figure 3.9(a), 3.9(b), 3.9(c) and 3.9(d) compares load distribution with *utility-centric* perspective in *hcma*, *karwendel*, *altix* and *schafberg* each with *node share (`share`)* as 204, 104, 64 and 16 respectively. The load assignment is relative to the *node share* (Section 3.3.4). Since *hcma* machine has highest share in the overall Grid capacity therefore it gets slightly higher than expected distribution whereas *schafberg* has lowest share thus it gets slightly



(a) hcma.uibk.ac.at (b) karwendel.dps.uibk.ac.at

(c) altix1.jku.austriangrid.at (d) schafberg.coma.sbg.ac.at

Fig. 3.11. Offering-centric Load distribution

less load than expected. The load distribution on the *karwendel* is almost similar as expected because its contribution in the Grid is closer to the average share, and the deviation of actual distribution from the expected is negligible. The load assigned to *altix1.jku* is reasonably lower, however, in contrast to the schafberg (Figure 3.9(d)), it growth is smooth. Furthermore, due to low share in the Grid, both *altix1.jku* and *schafberg* get their first allocations after 100^{th} overall allocation in the Grid.

Figure 3.10(a), 3.10(b), 3.10(c) and 3.10(d) compares *share-centric* load distribution. With this perspective, the distribution of load (resource allocation) is consistent but with higher degree of *deviations*. The depiction curves with *share-centric* are similar to *utility-centric* depictions. However, in the *share-centric* perspective, *hcma* (Figure 3.10(a)) gets higher than expected load but with almost double the deviation than in case of *utility-centric* perspective. Similarly, altix1 (Figure 3.10(c)) and schafberg (Figure 3.10(d)) get lower than expected load but with almost 2.5 times higher deviation. *Karwendel* (Figure 3.10(b)) allocation is almost consistent with slight deviation in initial and final allocations. Furthermore, due to low share in the Grid, both

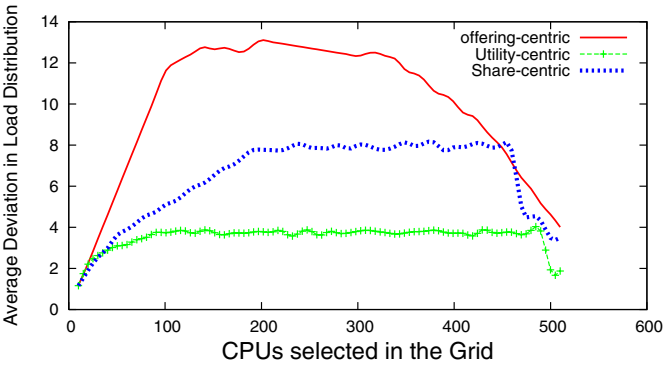


Fig. 3.12. Average Deviation in Load Distribution during Selection of (Physical) Resources.

altix1.jku and *schafberg* get their first allocations after 200^{th} overall allocation in the Grid.

Here deviation σ is calculated based on actual and expected values of load distributions. Let say x is actual and $exp(x)$ is expected value then deviation σ^d is

$$\sigma^d = \sqrt{x^2 - (exp(x))^2} \tag{3.5}$$

Figure 3.11(a), 3.11(b), 3.11(c) and 3.11(d) compares *offering-centric* load distribution. With this perspective the distribution of load is very inconsistent with much higher degree of deviation of actual load distribution (resource allocation) than expected. It is depicted that *hcma* (Figure 3.11(a)) initially gets exclusively more than 100 allocations whereas *schafberg* (Figure 3.11(d)) gets its first allocation after almost 350 overall Grid allocations.

Figure 3.12 compares average deviation of all three perspectives. It is quite obvious that proportional share-based load distribution with the *utility-centric* perspective is much better and has lowest deviations. This verifies that *GridARM* provides a steady system with a selection model that works better for the resources (and resource providers) without compromising over the client requirements, rather clients are being offered relatively better options as compared to what they asked at the first place.

Figure 3.13 depicts *GridARM* overhead with and without cache enabled with varying number of concurrent clients. Since the default information service is Globus MDS2 which works based in LDAP [103], therefore *GridARM* overhead is compared with LDAP overhead as well. The interesting finding is that without cache enabled, the *GridARM* overhead grows linearly and this linear growth is similar to the linear growth of LDAP overhead. In contrast, the overhead remains consistent if cache is enabled. The difference between LDAP and *GridARM* is due to additional overhead of WSRF (Section 2.3.2) middleware infrastructure. This proves that *GridARM* overhead is negligible if compared with combined overhead of LDAP and WSRF.

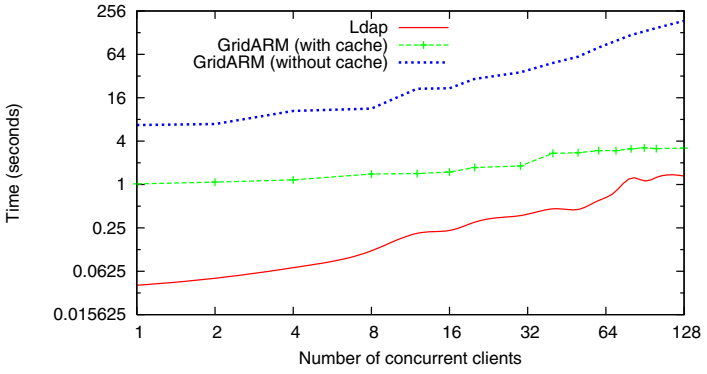


Fig. 3.13. The GridARM Overhead Comparison with Concurrent Clients.

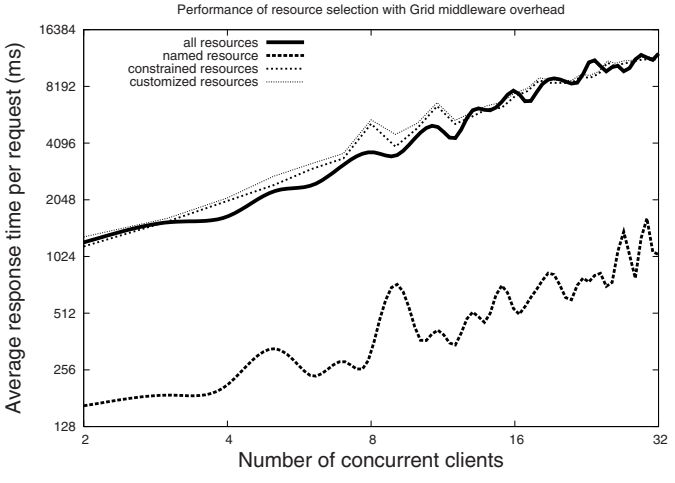


Fig. 3.14. Performance of resource discovery and selection with varying number of clients, including Grid middleware overhead.

Figure 3.14 depicts performance of resource selection with varying number of clients. The response time grows linearly with clients. According to the results, the request for the named resources is more economical. This is the simplest form of request in which no resource matching is required. The request for all registered resources and the request with multiple constraints is more expensive. In the Grid, which is meant for execution of time-expensive scientific application, *GridARM*'s performance is quite encouraging. The performance of the customized resource manager is also comparable with other curves even though it has to perform an additional step of overriding some of the attributes for a specific client.

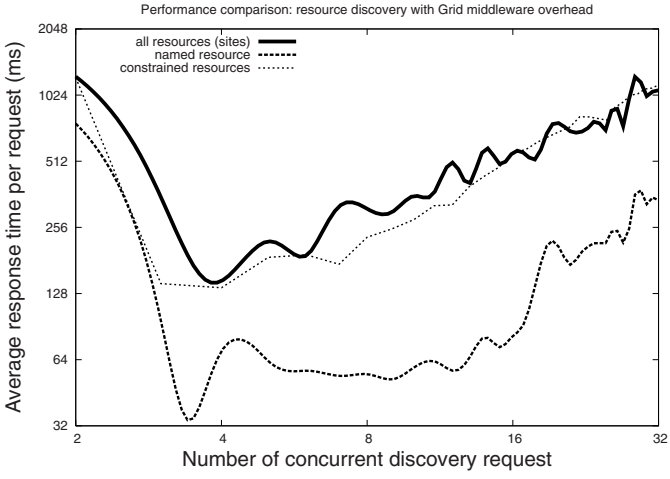


Fig. 3.15. Performance of resource discovery (excluding selection) with varying number of clients, including Grid middleware overhead.

Figure 3.15 shows average response time with concurrent discovery-requests. A *discovery request* is like a *selection request* but it does not include additional step of *candidate set generation* and *synthesis*. Again request for named resources is more economical whereas a resource discovery for multi-constrained request is relatively expensive. However, the overhead remains under one second. The middleware initialization overhead is significantly visible in the curves.

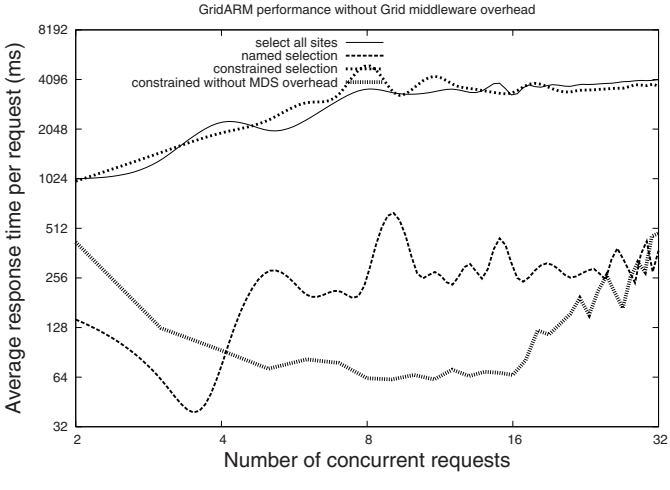


Fig. 3.16. Performance of resource provision with concurrent requests and excluding Grid middleware overhead.

Figure 3.16 depicts performance of the system without the Grid middleware (WSRF and network) overhead. The performance is almost consistent. The selection algorithm performance does not add any significant overhead as it is clear that main overhead is introduced by the middleware and the underlying information Services.

Figure 3.17 depicts average overhead of the *GridARM* components per request. Since *GridARM* is implemented as WSRF-compliant GT4-based Grid middleware, its quite logical to compare *GridARM* own overhead with WSRF middleware overhead. It is depicted that WSRF-based middleware overhead is much higher as compared to the *GridARM* component overhead. The fluctuation in the curves is introduced because of the multi-processor node on which *GridARM* was installed. Cache improves discovery phase and synthesis phase.

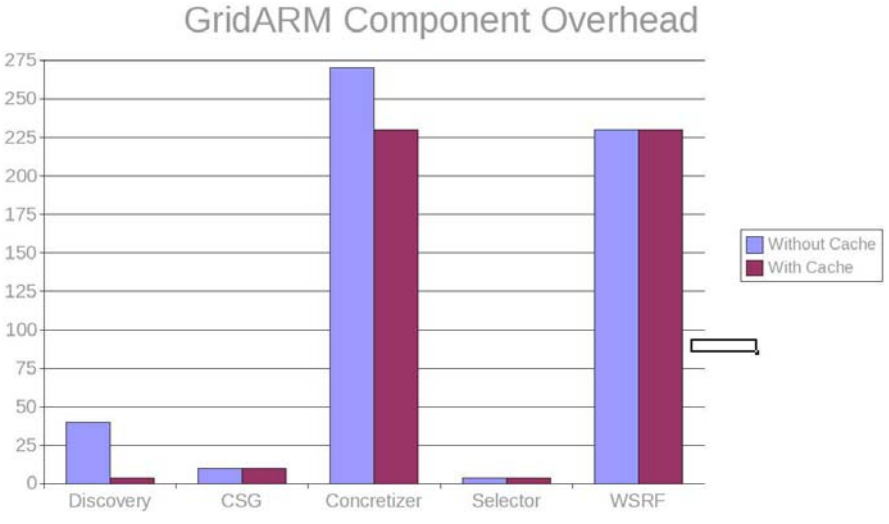


Fig. 3.17. The GridARM Components Overhead.

3.6 Related Work

Resource management in parallel systems is a well studied area of research [65]. In [110] performance of several resource matching heuristics is compared. The work in [166] introduces a proportional share resource allocation for real-time time-shared systems. However, mostly homogeneous resources under the full control of a resource manager are considered.

In the domain of resource management for the Grid, numerous projects and tools are available, but most of them do not provide the required level

of resource brokerage. This pervasive domain needs to split down further in more self contained and adaptable sub domains. Most of the existing Grid enabled systems try to address resource brokerage, job scheduling, and monitoring under the same integrated scenario. It works, but it's not scalable and adaptable. Also, this is un-compromising because negotiation is not possible. The resource broker in the Globus system is missing because Globus is more centered around operating environment.

A few Grid systems like Condor [128], Legion [119], GridLab [151], European Data Grid [147], Nimrod-G [25] and Maui [97] support resource management as a middleware functionary but the automatic brokerage is not a well divulged and concretely available functionality. Furthermore, none of the systems address resource management as self-contained module of the Grid middleware, in which distributed resource brokerage is supported as a super-peer model-based service-oriented infrastructure.

The resource selection problem occurs in many contexts with a variety of approaches for solution. Information systems such as SNMP [163], LDAP [103], MDS [39], UDDI [126] provide a mechanism for publishing, aggregating, and matching resources against client requirements. Such systems differ in various dimensions, such as supported syntax for describing resources and querying for the resources (e.g., SNMP MIBs, LDAP objects etc. Query languages such as SQL, Xquery [192], LDAP query [103]), and the techniques for publishing and aggregating resource descriptions. Among these, MDS and UDDI helps in brokering resources based on user constraints, however, planning for fair or optimal selection of resources is not addressed. Furthermore, querying mechanism supported by LDAP does not allow to specify complex constraints like $freeCPUs >= 0$.

Condor [128] is a resource provisioner that supports *high throughput computing (HTC)* on large collections of distributively owned computing resources. It provides a matchmaker [91] that supports a symmetric description mechanism in which both requests and resources are described using the same language called *ClassAd*s. A *ClassAd* can contain properties of a request or resource, and requirements that must be satisfied by a matching ClassAd, expressed as a boolean requirements statement. Selected resources are ranked according to a rank statement. Two ClassAds match if the requirements expression of each evaluates to true.

UNICORE [70] (Uniform Interface to Computing Resources) claims a ready-to-run Grid system that seamlessly makes distributed computing and data resources available in intranets and the Internet. The UNICORE resource broker is developed as an extension of EUROGRID [59] resource broker as part of Grid Reosurce Interoperability Project (GRIP). It is more centered around jobs instead of resources: it works with both Globus MDS and UNICORE IDB information services and performs resource matching with job descriptions. It also performs interoperability between Globus and UNICORE services. However, Globus MDS does not publish software resources whereas UNICORE does not publish dynamic information.

KOALA [117] is a Grid scheduler that supports co-allocation. It has been designed, implemented, and deployed by the Technical University Delft. It accepts job requests and uses a placement algorithm to try to place jobs. The placement algorithms selects a Grid node or a cluster depending on the vicinity of job's input data. KOALA is developed based on Globus operating environment and it ignores the issue of fair load distribution.

Some researcher are working on polices-base VO-wide resource allocation and reservation. The work described [176], introduces a framework for policy based allocation as a part of SPHINX that is a fault-tolerant system for scheduling in dynamic Grid. The allocation strategy in the framework adjusts resource usage accounts or request priorities for efficient resource usage management. Similarly, in [53] authors propose a usage policy-based allocation in VOs and evaluate both aggregate resource utilization and aggregate response time. The usage polices involved are fixed limit, extensible-limit, and commitment-limit, in which the limit is referred to as a fraction of the resources in a node shared to a VO. This is part of GRUBER that is a proposed usage resource broker [89].

The work presented in [101] is about fair resource sharing in hierarchical VOs. It proposed framework that uses a cooperative resource broker for VO-wide resource allocation. Each VO has a resource broker for VO users and resource providers. It gathers resource sharing information from VOs in a hierarchy and performs resource allocation based on gathered sharing information. A task can be distributed among several resources depending on their current load.

Elmroth et. al. has described in [56] a decentralized Grid-wide fair allocation system, where each local scheduler enforces Grid-wide hierarchical sharing policies using global resource usage data. The policy engine generates a fairshare factor for a job to support the Grid-wide share policy.

The work in [106, 107, 140] addresses resource matching problem that focuses on finding optimal resources for a single job with resource co-selection requirements. In contrast, instead of considering only job or client's perspective, the *GridARM* considers multiple allocations to archive optimal resource selection according to proportional share of the resources.

The work in [107] introduces a description language that improved expressiveness as compared with condor ClassAd. According to this approach resource selection is reinterpreted as a constraint satisfaction problem that exploits constraint-solving technologies to implement matching operations. However, the work focus only on client perspective and tries to improve only the client utility.

Ontology based resource matching proposed in [170] simplifies resource matching. In contrast to the symmetric resource matching as done by the *Condor*, ontological resource matching allows both resource requester and provider to specify resources and requests independently. These asymmetric description of resources leads to better expressiveness. However, there is no concrete semantic-based system.

In contrast, *GridARM* works to improve resource utility without compromising over the clients utility. The *GridARM* resource brokerage and selection mechanisms not only support resource matching but it moves a step forward to offer relatively better than requested resources. It makes planning by fair distribution of load among participating resources without compromising over the quality of service required by the clients.

The Open Grid Forum (OGF) is actively working on devising new standards in different areas of resource management. The *GridARM* system adopts OGF standards with minor modifications.

3.7 Summary

Unleashing the power of Grid infrastructures is a complex and tedious task without a sophisticated resource management system. The focus of this chapter is to render the boundaries of Grid resource management in general and resource selection and brokerage of physical resources in specific.

GridARM is implemented as a self-managed superpeer model-based resource management system distributed in a service-oriented fashion. Firstly, this chapter gives a general overview of *GridARM* architecture that represents a modular and dynamically extensible resource management for the Grid that is designed to fill the gap between the Grid job scheduler the underlying computing fabric. Three components of *GridARM* are introduced: *resource manager* covers selection and brokerage of physical resources or nodes, *activity manager* that deals with the lifecycle management of logical resources or activities, and *agreement manager* focuses on provisioning of optimal resource allocation and service-level agreement management.

Secondly, it introduces the *GridARM* model that performs resource discovery, candidate set generation, and brokerage. The resource selection is performed by making a fair load distribution according to the proportional share of Grid nodes.

Thirdly, it describes a new superpeer model-based distributed service-oriented infrastructure. This infrastructure is easily adaptable and scalable as compared to hierarchical distributed infrastructures such as MDS.

Finally, experiments and evaluation are shown to demonstrate the effectiveness of the *GridARM*, especially optimal load distribution based on the proportional shares of Grid nodes.

4

Grid Activity Registration, Deployment and Provisioning Framework

Resource provisioning is a key concern for implementing an effective resource management *as part of the Grid runtime environment; it delivers both physical and logical resources on-demand and shields the application developers from low level details. The previous chapter gives a general overview of the* resource management *(GridARM) with a detailed description of selection and brokerage problem of physical resources. This chapter introduces GLARE, an integral part of GridARM, that covers logical resources, particularly Grid* activities *[158] Existing Grid resource managers concentrate mostly on physical resources. However, some advanced Grid programming environments allow application developers to specify Grid application components (activities) at a higher level of abstraction which then requires an effective mapping between high level resource descriptions i.e. activity types and actual installations i.e. activity deployments This chapter describes* GLARE *that provides dynamic registration, automatic deployment and on-demand provisioning of activities that can be used to build Grid applications. GLARE simplifies description and representation of both activity types (abstract descriptions) and activity deployments (concrete deployments) so that they can easily be located in the Grid and become available on-demand. GLARE has been implemented as a distributed registry and deployment service by following the superpeer model of GridARM [154].*

4.1 Introduction

Advances in network technologies and emergence of the Grid have provided an infrastructure for computation and data intensive applications to run over collections of distributed and heterogeneous computing resources. Provision of a uniform access to these heterogeneous resources is one of the main goals of *resource management* for the Grid; this includes both physical and logical resources.

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 79–106, 2010.

Nevertheless, most existing resource management systems focus on *physical resources* and typically deal with clusters of computers, nodes, and job submission systems. Some efforts like GrADS [35], AppLeS [19], Askalon [62] and GridLab [151] have been made to provide automatic brokerage of *physical resources*. There is still much work to be done to effectively support deployment and configuration management of software components that essentially become part of workflow applications. Grid workflows [197] emerge as some of the most challenging and important classes of truly distributed Grid-enabled applications. Workflows require the composition of a set of application components, for instance executables or Grid/web services, which execute on the Grid in a well-defined order to accomplish a specific goal.

Most existing systems require manual or semi-manual deployment of software components (activities) and force application developers to hardcode into their workflows a set of software components deployed on specific nodes. In addition, currently available Grid information services are not well adapted to store complete description of software components, forcing the application developers to use only *(name,location)*-like information about available activities of workflow applications. As a consequence these applications are difficult to port to different Grid architectures, are sensitive towards dynamic changes in the Grid, and often imply an avoidable failure rate during execution. Such a manual and hardcoded approach forces an application developer to deal with low level details of the Grid. For instance, application components (activities) must be described along with their locations and access paths or URIs. All of that makes application development a time consuming, non-trivial, tedious, and error prone task.

There are some sophisticated Grid workflow programming environments and paradigms such as Pegasus [58] and Askalon [64, 62] that allow a Grid application developer to specify semantics of activities as part of a workflow application. However, there is a gap between the description of the functionality of an activity and the actual deployed services and executables that can provide such functionality. This gap can be eliminated or at least narrowed down by separating the description of the functionality of an activity from it's deployments, and through a sophisticated mapping mechanisms that goes beyond management of physical resources. Such an advanced management system should support dynamic registration of activities, automatic deployment on selected target nodes, on-demand provisioning, and optionally activity leasing.

This chapter describes *GLARE*, a Grid-level activity registration, deployment and provisioning framework that provides dynamic registration, automatic deployment and on-demand provisioning of Grid activities. *GLARE* is designed and implemented as a distributed framework that stores and provisions information about *activities.*Activities are the essential components of a Grid workflow that may reside on different computers and execute in a well defined order to accomplish a specific goal of the application. *GLARE* provides distributed registries for *activity types* and *activity deployments* along with

activity management service that perform registration, provisioning, monitoring, and automatic deployment of new *activities* on different nodes.Note that an *activity type* refers to as a functional description of an activity whereas an *activity deployment* relates to executables or (Grid/web) services that can actually be executed on a Grid node. Application developers can focus on *activity types* and thus must not be aware of specific *activity deployments*. *GLARE* simplifies the description and presentation of both *activity types* and *activity deployments* in such a way that they can easily be located in a distributed Grid environment and thus become available on-demand.

Moreover, *GLARE* introduces a leasing mechanism that enables a client (such as a scheduler) to *reserve* (or lease) an *activity deployment* for a certain time period. This leasing of activities is part of advance reservation mechanism described in reservation chapter.*GLARE* has been designed and implemented according to the underlying superpeer model [169] for *GridARM* that supports self management and fault tolerance. *GLARE*'s dynamic registration, automatic deployment and on-demand provisioning of *activities*, in combination with *GridARM*'s resource brokerage [158] and advanced reservation [158, 156],provides a powerful base for a Grid workflow management system and substantially improve the usability of the Grid while shielding its complexities.

4.2 On-Demand Provisioning Motivation

A workflow consists of *activities* [64, 197]. An *activity* is a high level abstraction that refers to a single self contained *computational task* that corresponds to an execution unit, initiated for instance by an executable program or a service, deployed on a Grid node. This section presents rationale behind *GLARE* and describes its usability in the Grid. Furthermore, it demonstrates *activities as generalized abstractions of the Grid tasks/jobs.*

4.2.1 An Example Using Bare Grid

In order to illustrate the advantages of *GLARE*, consider an example of a simple workflow with two activities: *ImageConverter* and *Visualizer* as shown in Figure 4.1. Formally,

$$\mathcal{W} = \{\mathcal{I}, \mathcal{O}, \mathcal{A}\} | \mathcal{I} = \{Image.POV\}, \mathcal{O} = \{Image.PNG\}, \mathcal{A} = \{ImageConverter, Visualizer\}$$

The input of *ImageConverter* is a *POVray*[1] [133] source file containing description of a scene, that is used to generate a 3-D image file. A client who wants to initiate *ImageConverter* activity on a Grid node (e.g. on a powerful computer), needs to deploy *POVray* on the target node, then sends a request to perform the image conversion, and finally transfer the resulting image to

[1] POVray is a high-quality tool for creating stunning three-dimensional graphics.

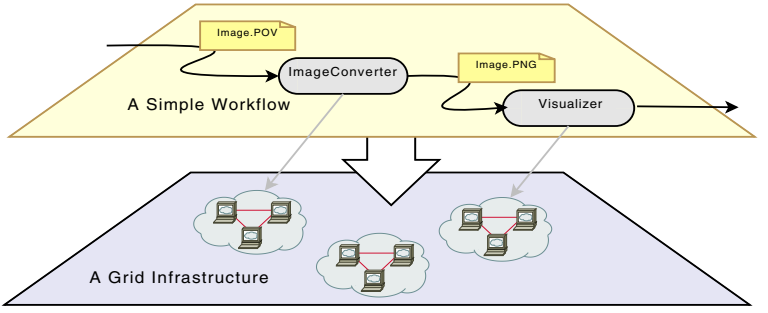


Fig. 4.1. A simple workflow execution on the Grid.

run a *visualizer* activity on its computer to analyze the resulting image. To simplify the understanding, lets assume that a Java version of *POVray*, called *JPOVray* (Figure 4.2), is available as an open source in the form of an executable and also a Web service, called *WS-POVray* which wraps the execution of *POVray* in a web service.

The required components to deploy the *JPOVray* application are: (a) the `javac` compiler (b) some (possibly required) libraries in the form of `.jar` files, (c) the `ant` (Another Neat Tool) build tool and (d) the source code of the *JPOVray* application itself. Once the application is (remotely) built and deployed, we need to store the information about deployed application in some information service: an *Endpoint Reference (EPR)* or URI in case the deployed application is a Grid/web service, and the *application name, path* and *home* in case the application is an executable.

The remote compilation[2] and deployment procedure requires information about the location of the compiler and built tool on the remote node, URL of required libraries and *JPOVray* source code. Example 10 shows a step-by-step procedure that is needed to perform the compilation, deployment and execution of the workflow using the basic Globus services, that is GRAM [41], MDS[3] [39] and GridFTP [9] on a target Grid site:

Example 10 (Step-by-step execution of the workflow).
`# Preparing environment`
`JAVA_HOME` = *Query* `MDS` *for location of* `java` *on target Grid node*
`if java` *not found* `then`
- *Query* `MDS` *for the location of* `JDK` *installation file*
- *Transfer installation file to target Grid node*
- *Create user-defined* `JDK` *deployment script*
- *Submit installation script using* `GRAM`

[2] Notice that the compilation of Java code is for exemplar purpose, otherwise authors are aware of *'Write once run everywhere'* slogan for Java.

[3] Usually, only physical resources are registered in MDS, but it can be used for logical resources like application components as well.

```
   JAVA_HOME = user-defined location used to deploy JDK
   - Update MDS with the information about the deployed JDK
endif
ANT_HOME = Query MDS for location of ant on target Grid site
if ant not found then
   - Do same steps to install ant as done for java and update MDS
endif
povray_libs = Query MDS for libraries
# Transfer needed application data for deployment
- Transfer the required libraries
- Transfer java application (JPOVray) source code
# Prepare build scripts
- Create script to remotely build and deploy JPOVray
using the information from MDS (JAVA_HOME, ANT_HOME
and set CLASSPATH)
- Submit deployment script through GRAM
povray_location  = user-defined location on remote Grid site
- Update MDS with information about newly deployed JPOVray
application (i.e. jpovray_location, libs_location etc.)
# Using the deployed application
- Query MDS to find JPOVRay service location
if deployed application is Grid/web service then
   - Contact the WS-POVRay service directly
elseif deployed application is an executable
   - Create script to run jpovray using
   java and libs_location
   - Submit execution script to run jpovray through GRAM
endif
# Visualization
- Retrieve result using GridFTP
- Visualize image on local station
```

In Example 10, it is necessary to put application-specific information of the JDK (Java Development Kit) and Ant in some information or registry service for (a) the deployment of the *JPOVray* and (b) the execution of JPOVray itself, i.e. there is a special need to store *activity*-specific description, so that the procedure can be automatized as much as possible. This becomes very complex for several activities, which must be orchestrated and executed as a Grid workflow [64].

The main problem is that the information stored in the information service (like MDS) maps the name of the activity directly to its location. Therefore, the description of the workflow cannot be done independently of a given application deployment, which represents a major disadvantage of current systems. A service which allows the registration, deployment and provisioning of *ac-*

tivities is needed, in order to simplify the automation of service composition and execution. The information stored in such a registry service should allow to map (a) the description of the deployed application activity, and (b) the access point (EPR or host:/path/to/application). We believe that such an activity registry should work in coordination with MDS, which is well adapted to store static information about available Grid resources (e.g. available Grid nodes along with information types like operating system, etc.), but not well adapted to store application-related information.

Creating an *automatic deployment* procedure for an application, as described in Example 10, using basic Grid services is non-trivial and very complex to achieve in practice. *GLARE* presents a more practical solution for this problem.

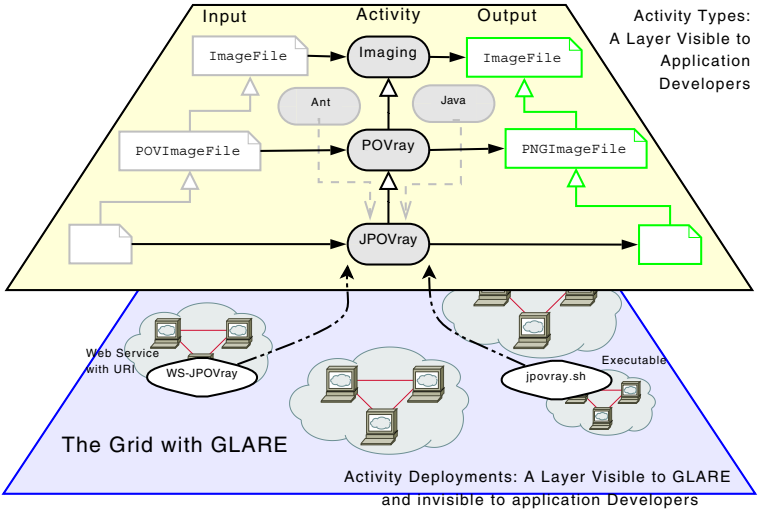


Fig. 4.2. Activity type hierarchy and *type* to *deployment* mapping.

4.2.2 *GLARE*-Based Solution

GLARE introduces a mechanism to specify abstract (functional or semantics) descriptions i.e. *activity types* as basic building blocks of an abstract workflow. Figure 4.2 depicts *JPOVray* as an abstraction that is dynamically mapped to its deployments: *WS-JPOVray* and *jpovray*. A developer uses only *activity types* while composing a Grid workflow. *GLARE* transparently maps *activity types* to matching *activity deployments* at runtime. This is a major advantage, since the Grid workflow composer does not need to know how and where the *POVRay* application is actually implemented (as an executable, Grid/web service, etc.) and deployed on the Grid.

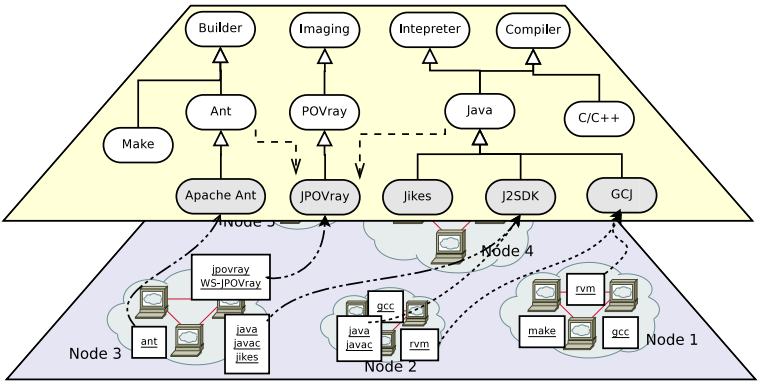


Fig. 4.3. Example activities with type hierarchy and Deployments on different Grid nodes.

Figure 4.3 shows a complete overview of the different activities that are needed to deploy and execute the sample workflow shown in Figure 4.1. In addition to hierarchies of *activity types*, the dependencies between components are also shown. *GLARE* handles these dependencies as well.

In order to provide dynamic registration, automatic deployment and on-demand provision of new *activities*, *GLARE* presents a distributed and fault tolerant infrastructure.It consists of distributed services which perform dynamic registration and automatic deployment of new *activities*. Each Grid node has a local *GLARE* service instance. The service provider describes the *activity types* to be registered with *GLARE*. The detailed information description that has to be provided is described in Section 4.3. Example 11 shows registration of *JPOVray activity type* in *GLARE*. Notice that the registration of an *activity type* is done only on a single Grid node, and *GLARE* takes care of distribution and deployment on other nodes on-demand.

Example 11 (Registration of JPOVray type).
`JPOVray.xml` = *Define* `JPOVray` *activity type in a xml template file*
`if Template` *does not exist* `then`
- *Transfer* `template` *xml from local GLARE service*
- *Modify* `template` *xml*

`endif`
- *Register* `JPOVray` *in the* `local` *GLARE service*

The workflow shown in Figure 4.1 can be composed of using *activity types* stored in the *GLARE* registry. The workflow description only specifies that a user needs an *activity* that can produce an image using a *POVray* scene description source file as input. The workflow description can then be submitted to the *scheduler*. The scheduler interacts with a *local GLARE service* and

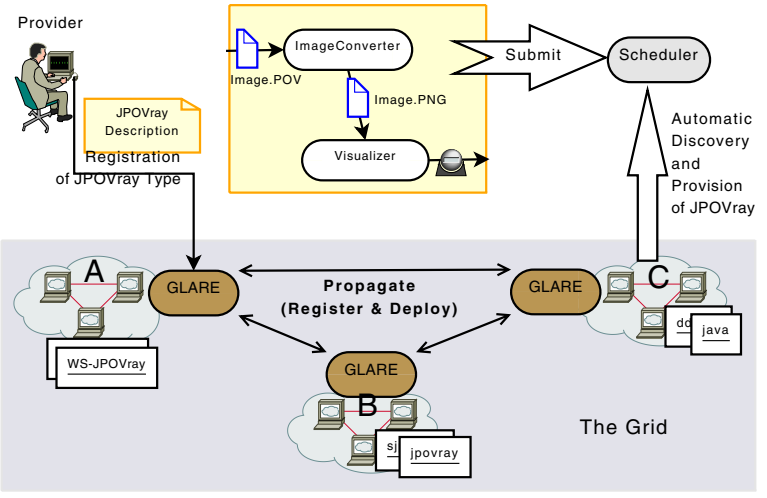


Fig. 4.4. A simple workflow execution by a scheduler with help of *GLARE*.

requests for an *activity deployment* capable to provide the requested activity, that is *ImageConverter*.

Example 12 shows steps involved in executing the workflow with *GLARE*. A client (*scheduler*) specifies *activity type* (any one in the type hierarchy) and *GLARE* returns a list of *deployment* references. Figure 4.4 demonstrates registration of *JPOVray activity type* on one Grid node by a provider and discovery of *JPOVray deployments* by the scheduler from an other node. Both activity provider and scheduler only need to interact with their local or frontend nodes.

Example 12 (Execution of workflow using GLARE*).*
`Result` = *Get* `ImageConversion` *deployments using local GLARE*
`if Result` *is empty* `then`
  - ***Deploy*** *and* ***register*** `ImageConversion` *automatically on a node*
  `Result` = *Get newly installed deployments using local GLARE*
`endif`
Deployment = *Select a* `Deployment` *from the* `Result`
`if Deployment` *is a Grid/web service* `then`
  - *Contact the service(WS-JPOVRay) directly*
`elseif Deployment` *is an executable*
  - *Instantiate JPOVray using* `jpovray executable` *as* `GRAM` *job*
`endif`
`# Visualization`
- *Retrieve result using* `GridFTP`
- *Visualize image on local station*

On a discovery-request by the scheduler, the local *GLARE* (e.g. at Grid node 2 shown in Figure 4.3) internally advances with following steps:

1. It looks up *ImageConverter* in the distributed *GLARE* registry and finds (after an iterative look-up) *JPOVray*, a specialized *activity* of the required type, but without any *deployment* anywhere in the Grid.
2. It analyzes the *JPOVray type* and found that (a) *JPOVray* can be installed on node 3 (b) *JPOVray* depends on activities *Java* and *Ant* and (c) both *Java* and *Ant* are not available on node 3.
3. It discovers *Java* and *Ant activity types* which are (a) suitable for target node and (b) has a *build-file* for automatic deployment.
4. If *build-file* exists, it invokes *deployment service* on the target node and sends the *build-file* to it. Deployment handler performs all steps described in the build-file automatically. Otherwise, it transfers installation files and required libraries on the target site using GridFTP.
5. It automatically connects to the target Grid node (as described in Section 4.4.1) to *build* and *install* both *Java* and *Ant activities* by automatizing the interactive installation procedure.
6. It identifies *deployments* (*e.g. java, javac and ant*) associated with newly deployed *activities* and registers them in the *deployment registry* along with information including executable name, path, home and type etc. The *templates* for deployment descriptions are provided in an *activity type* description by the activity provider, or automatically generated by *GLARE* service (e.g. by examining the *bin* sub directory of the deployed *activity* home for executables).
7. Finally, it transfers *JPOVray* installation file on to the target node and deploys it automatically. Furthermore, it *concretizes* JPOVray *deployments* (i.e. *jpovray and WS-JPOVray*), registers them in the *deployment registry* and returns their references to the client i.e. *scheduler*.

In this way, *GLARE* performs dynamic registration of new *types* and *deployments*, automatic installation and on-demand provision. The activity deployments *jpovray* and *WS-JPOVray* both provide same functionality but belong to different categories, one is an *executable* whereas the other is a web service. It is also possible that both deployments of the same type belong to different nodes and provide varying degree of QoS. Clients can select one of them suitable to their needs. *GLARE* hides *deployments* and the installation process of all *activities* thus shields the Grid complexities from its clients.

4.3 System Model

GLARE allows activity providers to describe *activities* in the form of *activity types* $\subseteq \mathcal{E}$and *activity deployments* $\subseteq \mathcal{D}$. *Activity types* are organized in a hierarchy of generalized types and specialized types.

A *generalized activity type* $\mathtt{at}' \in \mathcal{E}$ is an *activity type* which has no directly associated *activity deployments* whereas a specialized type $\mathtt{at} \in \mathcal{E}$ may have multiple activity deployments and it might be an extension of a generalized activity type. For each activity type $\mathtt{at}_i \in \mathcal{E}$ there is a set $\mathcal{D}_i \subseteq \mathcal{D}$ of activity deployments. If $\mathtt{at}_i$ extends a generalized activity type $\mathtt{at}'_i \in \mathcal{E}$ then $\mathcal{D}_i \subseteq \bar{\mathcal{D}}_i \subseteq \mathcal{D}$. Furthermore, $\mathcal{D}_i = \sum_{\mathtt{g} \in \mathcal{G}} \mathcal{D}_i^g$, where $\mathcal{D}_i^g$ is a set of activity deployments of $\mathtt{at}_i$ which are deployed on the node $\mathtt{g} \in \mathcal{G}$.

A generalized *activity type* $\mathtt{at}' \in \mathcal{E}$ may be used to represent an application that is extended by types of application activities. Nevertheless, an activity type may extend multiple generalized types and thus may belong to multiple applications. An activity deployment may have multiple *instances*.

Definition 30. *A running process of an activity deployment* $\mathtt{ad} \in \mathcal{D}$ *on a certain node* $\mathtt{g} \in \mathcal{G}$ *is referred to as* deployment instance.

As shown in Figure 4.2, *'Imaging'* and *'POVray'* are generalized types which perform image processing and define functionality (*render and export*) with possible input $\in \mathcal{I}$ and output $\in \mathcal{O}$ arguments. *'JPOVray'* is a specialized activity type that extends *POVray* and *Imaging* and thus inherits functional description of base types. *WS-JPOVray* $\in \mathcal{D}$ and *jpovray* $\in \mathcal{D}$ shown in Figure 4.2 are *deployments* of *JPOVray* which are installed on two different nodes. Activity *instances* are not shown in Figure 4.2. They are specific to a given execution of the Grid application and typically handled by the enactor [51].

Activities are registered in *GLARE* as *activity types* by activity providers. They are first installed on a set of nodes $\in \mathcal{G}$ and then registered in *GLARE* as *activity deployments*$\in \mathcal{D}$. This process of installation and registration of *activity deployments* is done automatically on-demand (Section 4.4). If a type is a generalized type $\mathtt{at}'_i \in \mathcal{E}$, then multiple deployments $\mathcal{D}_i^g \subseteq \mathcal{D}$ can be installed and registered collectively on a node $\mathtt{g} \in \mathcal{G}$. The address $\mathtt{ref}(\mathtt{ad}_i)$ of each *activity deployment* $\mathtt{ad}_i \in \mathcal{D}_i$ is associated with *activity type* $\mathtt{at}_i \in \mathcal{E}$ and

$$\forall \mathtt{ad}_i \in \mathcal{D}_i \exists \mathtt{at}_i \in \mathcal{E}$$

For an activity $\mathtt{a}_i \in \mathcal{A}$ there is a set $\mathcal{A}_i^d = \{\mathtt{a}_{i,1}^d, ..., \mathtt{a}_{i,n_i}^d\}$ of activities $\in \mathcal{A}$ on which $\mathtt{a}_i$ depends, n_i may be different for different $\mathtt{a}_i \in \mathcal{A}$. These dependencies need to be resolved, that means each activity $\mathtt{a}_j \in \mathcal{A}_i^d$ must be installed on a node $\mathtt{g} \in \mathcal{G}$ in order to install $\mathtt{a}_i$ on the same node. Figure 4.3 shows dependencies of activities that are needed for the sample workflow shown in Figure 4.1. As depicted,

$$\mathcal{A}_{JPOVray}^d = \{Ant, Java\},\ and\ \mathcal{D}_{JPOVray} = \{jpovray, WS\_JPOVray\}$$

where $jpovray \in \mathcal{D}$ is an executable and $WS\_JPOVray \in \mathcal{D}$ is as web service.

A workflow developer uses only *activity types* while composing a workflow. *GLARE* hides *deployments* before the runtime phase and transparently maps workflow *activity types* to *activity deployments* at runtime. This is depicted in Figure 4.4 where the registration of *JPOVray* activity is done on node $A \in \mathcal{G}$ by a provider and discovery of its deployments by the scheduler is done from node $C \in \mathcal{G}$.

Algorithm 4 The Concretization Algorithm.

synthesize()
Input: $\mathcal{B}, \mathtt{q}$ // *A set of candidates* $\mathcal{B} \in \mathcal{G}$ *and a resource requests* $\mathtt{q} \in \mathcal{Q}$
Output: $\mathcal{B}^{\mathtt{a}}$ // *A set of concretized candidates*
$\mathcal{G}$:= `The Grid`
$\mathtt{a}$:= `A requested activity a in q`
for all $\mathtt{b} \in \mathcal{B}$ **do**
  $\mathcal{D}^{\mathtt{a}}$:= **getDeployments**$(\mathtt{b}, \mathtt{a})$;
  if $\mathcal{D} == \emptyset$ **then**
    $\mathcal{D}$:= **deploy**$(\mathtt{b}, \mathtt{a})$ // *perform automatic deployment of* $\mathtt{a}$ *on* $\mathtt{b}$
    $\mathcal{D}^{\mathtt{a}}$:= **getDeployments**$(\mathtt{b}, \mathtt{a})$;
  end if
  consolidate activity deployments info $\mathcal{D}^{\mathtt{a}}$ with $\mathtt{b}$
  $\mathcal{B}^{\mathtt{a}} := \mathcal{B}^{\mathtt{a}} + \{\mathtt{b}\}$
end for
return $\mathcal{B}^{\mathtt{a}}$// *return concretized candidates*
.
getDeployments()
Input: $\mathtt{g}, \mathtt{a}$ // *A Grid node* $\mathtt{g} \in \mathcal{G}$ *and an activity* $\mathtt{a} \in \mathcal{A}$
Output: $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$ // *A set of deployments* $\mathcal{D}_{a}^{g}$ *of type* $\mathtt{at}_a$ *for node* $\mathtt{g}$
$\mathcal{D}_{\mathtt{a}}^{\mathtt{g}} := \emptyset$;
if $\mathtt{a} \neq \emptyset \wedge \mathtt{g} \neq \emptyset$ **then**
  $\mathtt{at}_{\mathtt{a}}$:= `Activity type of a`
  $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$:= `All registered deployments of type` $\mathtt{at}_{\mathtt{a}}$ `for g`
else
  if $\mathtt{a} \neq \emptyset$ **then**
    $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$:= `All registered deployments for g of any type` $\in \mathcal{E}$
  else
    $\mathtt{at}_{\mathtt{a}}$:= `Activity type of a`
    $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$:= `All registered deployments of type` $\mathtt{at}_{\mathtt{a}}$ `for any node` $\in \mathcal{G}$
  end if
end if
if $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}} = \emptyset$ **then**
  $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$:= `Contact superpeer to lookup alternative deployments`
  `cache` $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$
end if
return $\mathcal{D}_{\mathtt{a}}^{\mathtt{g}}$ // *available deployments*

Algorithm 5 The Deploy Algorithm.

```
deploy()
Input: g, a // A Grid node g ∈ G and an activity a ∈ A
E_a^d := dependancies(a) // all dependancies of a
for all a_i ∈ E_a^d do
  deploy(g, a_i)
end for
at := Get specialized activity type of activity a from activity type
registry
build_file := Download build-file of at;
steps := Get ordered list of steps defined in build-file;
for all step ∈ steps do
  execute step;
end for
D_new := Generate activity deployment descriptions
Register generated deployment descriptions D_new in activity
deployment registery
```

The architecture of *GLARE* is depicted in Figure 4.5. It consists of four components: *activity manager*, *deployment manager*, *activity type registry* and *activity deployment registry*. All components are stateful web services implemented based on Globus Toolkit 4. *Activity types* and *activity deployments* are maintained in separate registries. Each occurrence of an *activity type* and *activity deployment* in a registry is represented as a *WS-Resource*.

Definition 31. *A* WS-Resource *[191]* $\mathtt{wr} \in \mathcal{A}$ *is a stateful web service that implements a standard set of operations to access and/or manipulate its state in a service-oriented fashion. A special implied resource pattern [94] is used to describe a specific kind of relationship between a Web service and one or more stateful resources.* WS-Addressing *[184] standardizes the relationship with an endpoint reference construct.*

4.3.1 Activity Manager

The *activity manager* is a frontend service that represents a superpeer group. It organizes and manages *activity types* $\in \mathcal{E}$ and *activity deployments* $\in \mathcal{D}$ in separate registries and coordinates with multiple *deployment managers*. It complements *GridARM* selection and brokerage of physical resources and synthesizes them with logical resources by selecting required activities $\in \mathcal{A}$ and associating with generated list of candidates $\mathcal{B}$.

Definition 32. *The synthesis is a process of provisioning a set* $\mathcal{D}_i \in \mathcal{D}$ *of activity deployments of type* $\mathtt{at}_i \in \mathcal{E}$ *required for a candidate* $\mathtt{b}_i \in \mathcal{B}_i$ *generates by* `csg` *(Candidate Set Generator) according to a request* $\mathtt{q}_i \in \mathcal{Q}$*. If* $\mathcal{D}_i \in \mathcal{D}$ *is a set of deployments of type* $\mathtt{at}_i \in \mathcal{E}$ *then the synthesize function* $\kappa : \mathcal{D}_i \mapsto \mathcal{B}$ *maps a set* $\mathcal{D}_i^b \subseteq \mathcal{D}_i$ *to* $\mathtt{b} \in \mathcal{B}$*.*

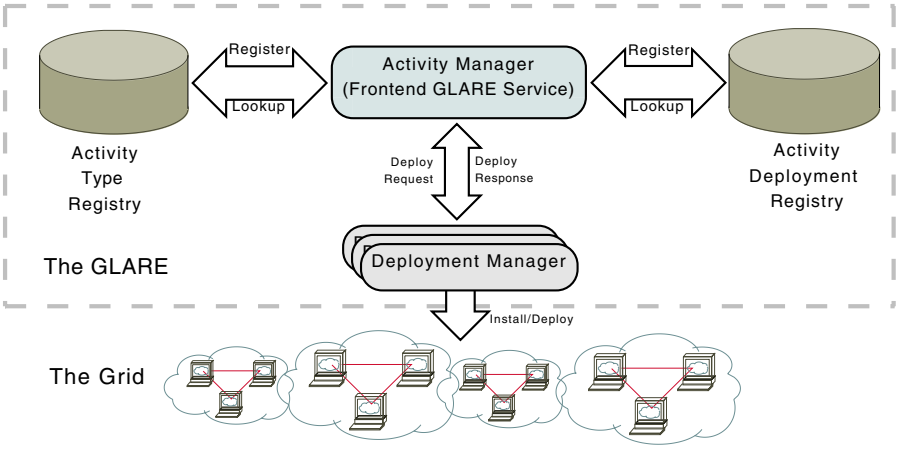


Fig. 4.5. *GLARE* system Architecture.

The pseudo code of the synthesis is given in Algorithm 4. According to this function, first *deployments* for a specific activity on a certain *node* are discovered assuming the requested activity has already been deployed. Otherwise, the *deploy* function is called in order to perform automatic installation of the activity requested for a certain node. The *deploy* function, shown in Algorithm 5, is explained in Section 4.4.1. Ideally, the synthesis should be called by *csg (Candidate Set Generator) function* (To be explained in previous chapter). Nevertheless, it can be invoked directly to look-up deployments by activity type or by node and activities can be deployed as well.

The *activity manager*, also refers to as *GLARE service* needs to be deployed only on a superpeer node. It receives and handles requests both from clients in the form of queries and from activity providers in the form of registration and updates. Furthermore, it performs monitoring of registered activities on other nodes. The *synthesize* algorithm is explained in Example 13.

Example 13 (POVray activity: from registration to on-demand provisioning). In order to understand *synthesize* algorithm we proceed with a simple example of *POVray* activity. Lets have a set of Grid nodes

$$\mathcal{G} = \{A, B, C, D, E, F, G, H, K\}$$

distributed in two superpeer groups and a set of activities $\mathcal{A} = \{POVray\}$. The abstract description of *POVray* is registered in node $C \in \mathcal{G}$ by its *provider* as shown in step 1*a* of Figure 4.6. This step is followed by step 1*b* where *GLARE* propagates POVray abstract description to node $A \in \mathcal{G}$ working as superpeer node. With this state of the Grid, a set of candidates $\mathcal{B} = \{E, K\} \subset \mathcal{G}$ is generated by *csg* on request. One of the constraints of the requester is that activity POVray must be available on selected nodes. This is ensured by the following steps:

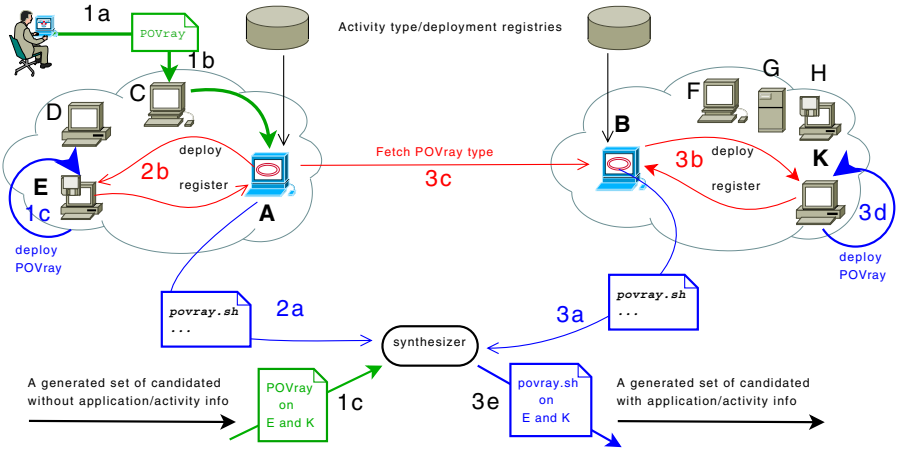


Fig. 4.6. POVray activity: from registration to on-demand provisioning

1. A request is fulfilled by *csg* that generates a set of candidates $\mathcal{B} = \{E, K\} \subset \mathcal{G}$ by filtering out unwanted nodes and forwarding $\mathcal{B}$ to the synthesizer.
2. Synthesizer further filters out and concretizes the candidates with required set of activities i.e. $\{POVray\}$. It fetches POVray deployments for candidates $\in \mathcal{B}$ as shown in steps $2a$ and $3a$ of Figure 4.6 and drops out candidates for which it could not find required activity deployments.
3. On request for POVray deployments, the *GLARE service* instances of node A and B discovers POVray deployments for candidate E and K respectively. If POVray is already deployed then its deployments are returned back. Otherwise *GLARE service* contacts *deployment manager* of each candidate $\in \mathcal{B}$ and initiates automatic installation on target candidate. This is depicted in steps $2b$ and $3b$ for candidate E and K respectively.
4. The *deployment manager* of each candidate downloads POVray type. Then it fetches most specialized POVray type, its *build-file* (Section 4.4) and all dependencies. Using POVray *build-file* it installs POVray along with its dependencies on the local node i.e. E and K and returns back POVray deployments. This is depicted in steps $1c$ and $3d$.
5. If superpeer node could not find activity type then it contacts another superpeer node for required activity type i.e. POVray. This is depicted in step $3c$ where superpeer $B \in \mathcal{G}$ contacts $A \in \mathcal{G}$ for registered POVray activity type.
6. Once POVray deployments are discovered or installed they are returned back to synthesizer. Finally, synthesizer consolidates deployment and candidate descriptions and returns consolidated set of candidates as depicted in final step $3e$.

4.3.2 Deployment Manager

The *deployment manager* deals with lifecycle management of activity deployments. It performs on-demand installation of new *activities* on a single or multiple nodes automatically as well as concurrently. It provides an interface to *deploy*, *undeploy*, *register* and *unregister* deployments by their association either with an *activity type* or a node. The *activity manager* is the main client of *deployment manager*. The *deployment manager* is an optional service that needs to be deployed on each node on which automatic deployment of activities is required. Nevertheless, in the absence of this service on a node, a remote deployment procedure can be initiated automatically by a deployment manager of the superpeer node. Otherwise, the automatic deployment is not possible on that node and a provider has to do it manually. The deployment entries still can be registered and unregistered by the *activity manager*. During the deployment process, all the dependencies of the activity to be deployed are resolved recursively. This is presented in Algorithm 5 and explained in Example 13.

4.3.3 Activity Type Registry

Activity type registry maintains a set of named *activity types* in the form of WS-Resources organized in a hierarchy. It presents a more *generalized activity type* as root type and uses it in discovering *specialized types*. *Specialized types* are installed on nodes in the Grid and may have associated *activity deployments* and/or a reference to a special kind of *build-file* that describes a series of steps needed for automatic deployment on a Grid node.

Activity types are described in terms of *base types*, *domains*, *functions/operations*, *input/output arguments*, installation procedure and constraints, and dependencies upon other activities. Also, types keep track of available deployments across the Grid distributed in a superpeer group. *activity types* are used to discover *activity deployments*. Similar to *activity manager*, registries are also required to be deployed on a superpeer node.

4.3.4 Activity Deployment Registry

The *activity deployment registry* complements *type registry* and maintains *activity deployments* as WS-Resources. An *activity deployment* refers to an executable or a web/Grid service and provides information required for the *selection* and *instantiation* of a deployed (installed) *activity*. The Endpoint Reference (EPR) of each *activity deployment* resource is registered in its *activity type* resource presented in the *type registry*. Each deployment entry as a *WS-resource* can be persistent as well as non-persistent. A persistent entry remains available after its deployment whereas a non-persistent deployment entry disappears after a configurable time interval.

Both activity type and deployment registries are part of a distributed framework. They can access all entries registered on different nodes enabled with *GLARE* services distributed across the Grid.

4.4 Implementation

Installation and deployment of scientific applications (a set of *activities*) on different nodes in the Grid is a time consuming and labour intensive task. The *GLARE* automatizes this process. It provides a mechanism in which an activity provider can register new *activity types* along with an *installation* procedure described in an associated *build-file*. An example *build-file* with a series of required steps for *POVray* installation is given in Example 14. A new *activity type* registered with one node can be discovered by other nodes in a superpeer group and installed on-demand (automatically) based on constraints specified in the *type* description. Furthermore, simultaneous installation can be performed on multiple Grid nodes, with least involvement of sysadmins.

Example 14 (JPOVray build-file with steps for automatic deployment).

```
<Build baseDir="$DEPLOYMENT_DIR" xmlns=http://build.glare.askalon.org"
  defaultTask="Deploy" name="jpovray">
  <Step name="Init" task="mkdir -p" baseDir="$DEPLOYMENT_DIR">
      <Env name="ACTIVITY_HOME" value="jpovray"/>
      <Env name="ACTIVITY_TARBALL" value="$ACTIVITY_HOME.tgz"/>
      <Env name="SRC_URL" value="http://dps.uibk.ac.at/glare"/>
      <Env name="ACTIVITY_HOME_PATH"
                    value="$DEPLOYMENT_DIR/$ACTIVITY_HOME/"/>
      <Env name="BIN_DIR" value="$ACTIVITY_HOME_PATH/bin/"/>
      <Property name="argument" value="$DEPLOYMENT_DIR"/>
  </Step>
  <Step name      ="Download" depends="Init"
        task      ="$GLOBUS_LOCATION/bin/globus-url-copy"
        baseDir   ="$DEPLOYMENT_DIR" timeout="80">
    <Property name="source" value="$SRC_URL/$ACTIVITY_TARBALL"/>
    <Property name="destination"
              value="file:///$DEPLOYMENT_DIR/$ACTIVITY_TARBALL"/>
    <Property name="md5sum" value=""/>
  </Step>
  <Step name="Expand" depends="Download" task="tar xfz"
                    baseDir="$DEPLOYMENT_DIR" timeout="30">
    <Property name="argument" value="$ACTIVITY_TARBALL"/>
  </Step>
  <Step name="Build" depends="Expand" task="make"
                    baseDir="$ACTIVITY_HOME_PATH" timeout="180"/>
  <Step name="Deploy" depends="Build" task="make deploy"
    <Dialog expect="Install as root or normal user (R/U)?:" send="U"/>
  </Step>
  <Step name="Clean" task="make clean" baseDir="$ACTIVITY_HOME"/>
  <Step name="Undeploy" depends="Clean"/>
</Build>
```

□

4.4.1 Automatic Deployment Using *Expect*

Currently, installation with *autoconf* (`configure, make, make install`) and auto build using *ant* is supported. The *build-file* should be either included in activity type description or accessible with *GridFTP*. Similarly, source URLs must be accessible with GridFTP for transfers to the target Grid node. An activity provider can specify different constraints which must be fulfilled before the installation, for example, pre-requisite platform and operating system etc. An activity can be restricted to a certain number of nodes or can be revoked temporarily. An activity provider can use default environment variables `DEPLOYMENT_DIR`, `USER_HOME`, `GLOBUS_SCRATCH_DIR` and `GLOBUS_LOCATION` in the build-file, and *deployment manager* substitutes their values dynamically at runtime. These environment variables are normally set by sysadmins of the node.

After successful installation, the *activity type* is marked as deployed and specified executables or services are registered in the *deployment registry* in the form of *activity deployments* as WS-Resources. The templates for deployments entries are provided in the *activity type* description by the activity provider, or *deployment manager* automatically generates, for instance by examining the *bin* sub directory of the *deployed activity home*. In case of failure, or installation *mode=manual deployment manager* can notify administrator of the target node by email referring to the website of the activity or contact of its provider. Making automatic deployment at registration time eliminates the overhead of manual or *on-demand* deployment. This leads towards concurrent installation on all nodes in the Grid. However, in order to control unwanted installations on different nodes, only constraint-based or on-demand deployment can be supported. A smart scheduler can reduce overhead of on-demand deployment by providing intelligent look-ahead scheduling.

Expect-Based Installation

The *deployment manager* provides a backend *deployment handler* that performs interactive installation on a node programmatically that otherwise is not possible with batch jobs.

> The *deployment handler* is an *Expect* [60] based virtual terminal that is used to automatically interact with operating systems of different Grid nodes and perform interactive process of installation on a target node. Expect is a method of programmatically automatizing interactive applications/tools such as `telnet, ftp, password, ssh, glogin` etc.

The *GLARE* uses local shells (e.g. `tcsh`) or remote shell such as `ssh/glogin`[4] [84] to login on a node securely with the *expect* mechanism. Alternative imple-

[4] Glogin is a secure shell that uses standard Globus GRAM and GSI mechanism, i.e. the users can use their proxy certificates to log into a remote Grid node, without any additional server running (as gsissh).

mentations of *deployment handler* with different mechanisms is possible. For instance, as an alternative to *glogin*, the deployment handler can use GRAM on target node and issues commands in the form of GRAM jobs. By default local shell is used by the *deployment handler* running on a node.

The *deployment handler* exploits *Expect* for interactive installation. For instance, the installation of *POVray* requires human interaction and prompts for license acceptance, user type and install path, and activity provider specifies this interaction dialog in *build-file* in the form of *send/expect patterns* as shown in Example 15 as a *Dialog* element.

4.4.2 Static and Dynamic Registration

As stated earlier, an activity or a set of activities in the form of an *application* is to be registered by an activity/application provider. This can be done by interacting with the local *activity manager* either by using a graphical console application or a command-line tool. The graphical console provides a user-friendly interface whereas the command line tool needs a *Grid Workflow Deployment Descriptor (GWDD)* for registration of an application along with its activities. A sample *GWDD* for registration of *POVray* application is shown in Example 15. It specifies minimum required attributes for *JPOVray* application and its associated activities i.e. *converter* and *renderer*. Furthermore, executables, usage, in/out ports are defined for each activity. These information has been described by the *JPOVray* provider. Figure 4.7 shows the *GLARE* console application that provides a 'single-click' functionality to `register, unregister, deploy` and `undeploy` an application.

Example 15 (JPOVray build-file with steps for automatic deployment).

```
1
2  # Grid Workflow application deployment descriptor
3  # Fri Jun 22 09:40:40 CET 2007
4  application           = JPOVray
5  JPOVray.type          = jpovray
6  JPOVray.domain        = Imaging
7  JPOVray.environment = gt2
8  DELIM                 = AND
9
10 JPOVray.buildFileURL = http://dps.uibk.ac.at/glare/jpovray.build
11 JPOVray.activities    = Converter Render
12
13 Converter.executable = convert.sh
14 Converter.usage      = [binDir] [outFileName]
15 Converter.inports    = \
16   frameTarballs frames.tar agwl:collection AND \
17   outFileName frames.png xs:string
18 Converter.outports=outFile [outFileName] agwl:file
19
```

```
20 Render.executable=render.sh
21 Render.usage=[binDir] jpovray.ini jpovray.pov jpovray.arg \
22   [startFrame] [numFrames] [totalFrames]
23 Render.inports=iniFile jpovray.ini agwl:file AND \
24   povFile jpovray.pov agwl:file AND \
25   argFile jpovray.arg agwl:file AND \
26   startFrame 1 xs:integer AND \
27   numFrames 25 xs:integer AND \
28   totalFrames 250 xs:integer
29 Render.outports=frameTarball frames[startFrame].tgz agwl:file
30
31 JPOVray.accessPaths=
32   karwendel.dps.uibk.ac.at:/var/deployments/jpovray AND \
33   altix1.jku.austriangrid.at:/var/agridx/deployments/jpovray
```

□

In contrast to *activity type* registration, *activity deployments* can be registered manually as well as automatically and on-demand. After successful deployment of an activity on a node by a *deployment manager*, the possible deployment entries associated with the installed activity are automatically generated and registered with the *deployment registry*. The deployment registry must be available either on the same node or on the *superpeer* node. The corresponding *type registry* is notified for newly registered deployments. The *type registry* is responsible for preserving and discovering a matching *activity type*. In case of failure in discovering a matching *activity type*, the *deployment registry* requests *type registry* for dynamic registration of types of newly registered deployments.

A new *activity type* registered statically or dynamically with one node can be discovered automatically by other nodes. A resource discovered from a remote registry is optionally cached.

4.4.3 On-Demand Provisioning

Activity type and deployment registries provide an aggregation of all locally registered and cached resources, based on a WSRF [191] service-group framework, in which aggregated resources are periodically refreshed. This enables the service to discover resources (*activity types* or *deployments*) by using standard XPath-based querying mechanism. In order to answer queries for *named resources* efficiently the registry services use hash tables to access named resources. This eliminates XPath-based search requirements for *named resources* and significantly improves the performance.

Caching and Cache Monitoring

To ensure an efficient on-demand provision, the *GLARE* supports a two-level cache; cache at normal Grid node and cache at *superpeer* node, and provides a

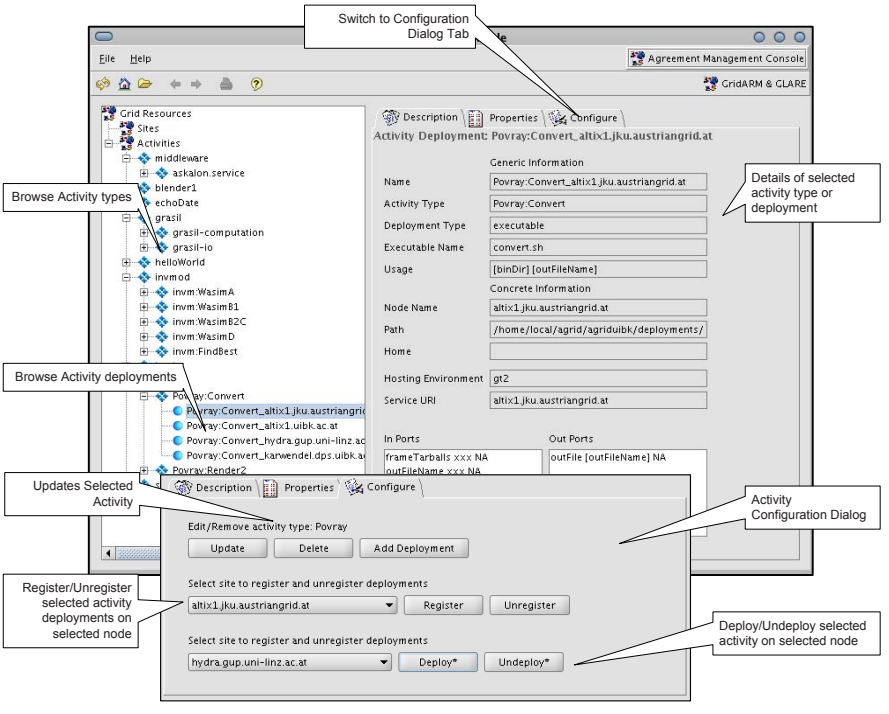


Fig. 4.7. A snapshot of *GLARE* console.

mechanism to refresh cache of updated resources. The cache of *activity types* and *activity deployments* is maintained in their registries (Figure 4.5) and a *Cache Refresher* component, as part of *activity manager*, updates cached resources if and when they change on the source node. Outdated resources are discarded automatically.

A *deployment status monitor*, another component of *activity manager*, periodically checks status of locally registered *activity deployments* and updates their entries (WS-Resources) and endpoint references registered with their *activity types*. The *deployment Endpoint Reference (EPR)* may contains an additional attribute *Last Update Time (LUT)* that can be used by the *cache refresher*.

The *deployment status monitor* can register in local WS-GRAM service to get the latest metrics associated with *deployment instances* (Definition 30). For example, attributes like *last execution time*, *return code*, *last invocation time* etc. can be useful in better scheduling and in an agreement enforcement process that is described in next chapter.

Deployment Leasing

The *activity manager* provides a capability to lease an activity (deployment) with the help of *GridARM Reservation* mechanism. A fine-grained reservation of a specific *activity deployment*, instead of the entire node, can be possible. A user with valid *reservation ticket* is authorized to instantiate the reserved activity. A lease can be exclusive or shared. In case of an *exclusive lease* no one else is allowed to use the activity during the leased timeframe. In case of *shared lease*, multiple clients can use the *leased activity* but *GridARM reservation* service ensures that the number of concurrent clients does not exceed the allowed limits and the required QoS is ensured. A detailed description of *GridARM* reservation service is given in next chapter.

Local Access

The distributed *GLARE* ensures that clients of different Grid nodes have the same view of the entire superpeer group. An activity is discovered and provisioned by a local Grid node independent from the location of its associated *deployments*. This is in contrast to the hierarchical model of MDS, in which a client has to contact root or the *community index* in order to get the entire view of all nodes [39]. This enables clients (end-users) to interact only with their local nodes and get all distributed *activity types* and *deployments*. Clients don't have to consider or remember a centralized service and its access mechanism.

4.4.4 Self-Management and Fault Tolerance

The *GLARE* framework is self-managed and fault tolerant. It is developed based on superpeer model that uses Globus Toolkit 4 (GT4) built-in hierarchical aggregation and indexing mechanism to discover Grid nodes and form superpeer groups [158]. If some nodes or services fail, the rest of the system continues functioning. A superpeer failure leads to the re-election of a new superpeer. The *GLARE* system is designed as a set of WSRF services distributed in the Grid with platform-independent interaction mechanism. This makes it acceptable for both Grid and web services technologies. The openness of underlying infrastructure and superpeer model based design makes *GLARE* a scalable middleware that shields application developers from the Grid. Furthermore, automatic superpeer election and activity installations upgrade *GLARE* system to become self managed and fault tolerant.

The following section shows experiments which demonstrate the effectiveness of the *GLARE*.

4.5 Experiments and Evaluation

The *GLARE* has been implemented based on GT4 and integrated in Askalon Grid environment [61] and deployed on different nodes in the Austrian Grid infrastructure [33].

The Austrian Grid is a national computing Grid infrastructure distributed across several cities and institutions across Austria. The infrastructure is composed of more than ten Grid sites that aggregate over 200 processors. Each local Grid site system administrator independently installed his favorite local job manager and the Globus toolkit (GT2 or GT4) for integration within the Austrian Grid.

The *on-demand* deployment of new activities is evaluated with calibration of the deployment overhead of real world scientific applications. For this purpose three applications are selected;

1. *Wien2k* [21] (pre-compiled) which performs electronic structure calculation of solids based on density functional theory.
2. *Invmod*, a hydrological application for river modeling which has been designed for inverse modelling calibration of the WaSiM-ETH program [99],
3. and *counter service*, a sample GT4 service that represents GT4 features and used here to demonstrate the deployment of a Grid service.

Table 4.1 illustrates time spent in different operations and components of the *GLARE* framework. On-demand deployment is performed in two alternative ways; with JavaCoG (using GRAM and GridFTP) and with *Expect* by programmatically acquiring local system shell and automatizing the installation process. The communication and deployment overhead depends on the size of installation files and compilation respectively. As shown in the Table 4.1, the registration of a new *activity type* and its *deployments* and notification to the node administrator imply reasonable costs. Downloads take some time but significant time is spent in compilation and installation. Also, *Expect* is more efficient than Java CoG. The overall scheduler overhead shown in the Table, can be eliminated by employing automatic deployment, or reduced by providing a *schedule-ahead* mechanism by a scheduler.

The efficiency, performance and scalability of *GLARE* is tested by deploying it on up to 7 Austrian Grid nodes. We have compared an integral component of the *GLARE* framework, that is, *activity type registry* with the GT4 *Index Service* (WS-MDS) by registering multiple *activity types* as WS-Resources in both services. The experiments are performed with and without transport level security enabled (i.e. with http and https). Note that, although *index service* is normally used for physical resources but the underlying aggregation framework (WSRF-based GT4 aggregation framework) is same for both GT4 Index service and *GLARE* registries. Therefore it is logical to make this comparison.

Figure 4.8 shows performance of both services with and without security enabled. Throughput decreases almost by 50% for both services with trans-

Table 4.1. Time spent (in ms) by different operations.

Deployment Method	Operation/Overhead	Wien2k	Invmod	Counter
Expect	Activity Type Addition	633	632	665
	Communication Overhead	1,667	1,381	1,279
	Activity Installation/Deployment	8,068	27,776	29,843
	Activity Deployment Registration	355	350	352
	Notification	345	345	345
	Expect Overhead	2,100	2,100	2,100
	Total overhead for meta-scheduler	11,068	30,484	32,484
Java CoG	Activity Type Addition	633	632	665
	Communication Overhead	5,600	2,500	2,400
	Activity Installation/Deployment	18,068	49,700	39,756
	Activity Deployment Registration	355	350	352
	Notification	345	345	345
	JavaCoG Overhead	9,800	9,900	9,800
	Total overhead for meta-scheduler	25,001	53,527	43,518

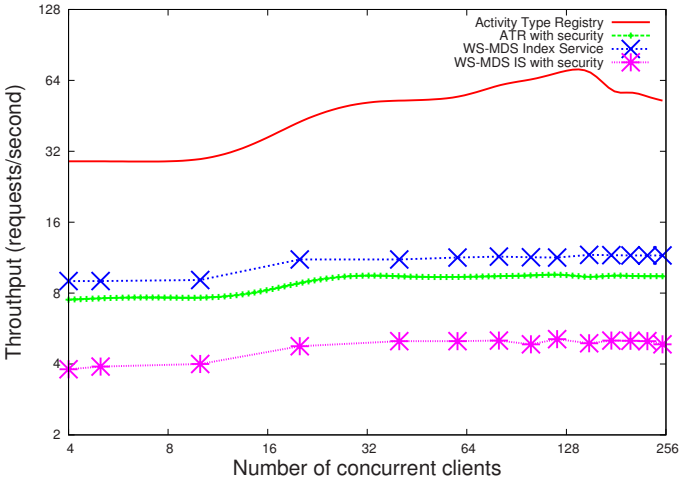


Fig. 4.8. Comparison of Activity Type Registry and WS-MDS Index Service both with and without transport level security. Throughput with varying number of concurrent clients.

port level security. Index Service is 50% slower than *activity type registry* because of its XPath-based querying mechanism and hashtable indexing. This experiment was performed with both WS-MDS *Index* and *activity type registry* services running on the same Grid node with same number of registered activity types, whereas clients were distributed among 7 other nodes.

Figure 4.9 shows a comparison of *activity type registry* with GT4 *index service* with a varying number of *activity type* resources in the registry and index service, again with and without security. Throughput of *Index*

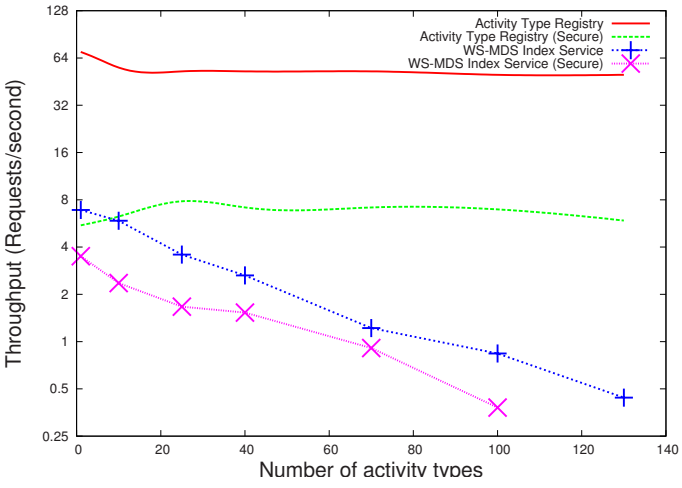


Fig. 4.9. Performance comparison with an increasing number of activity types.

Service decreases significantly with increasing number of resources whereas it can be observed that throughput of an *activity type registry* is consistent. A good performance comparison of previous versions of MDS is given in [105, 200].

The scalability of *GLARE* on 1, 3 and 7 nodes is also tested, with and without cache enabled. Figure 4.10 shows response time per request for a list of *deployments* associated with an *activity type*. Deployment entries are equally distributed on all nodes. It is observed that there is a significant improvement in performance by increasing number of nodes or by enabling caching mechanism.

Figure 4.11 shows the change in the 1-minute load average as the number of clients (requesters) and event notification listeners (sinks) increases; the load average is measured as the load on the *Activity Type Registry* during the last minute (using Unix *uptime* command). The load average is therefore a measure of the number of jobs waiting in the run queue. The highest load average occurs when the notification rate is 1 sec. It peaks slightly above 16 corresponding to 210 sinks. Load average is proportional to the notification rate. The load average against the number of requesters peaks just below 5, which shows consistency.

Finally, it is observed that sometimes *index service* stops responding when more than 130 *activity type* resources are registered in it and number of concurrent clients exceeds 10 (Figure 4.9). This is quite strange behavior and could be a real shortcoming of the index service, which may become a bottleneck when registered number of nodes increases. In contrast, the *GLARE* registry services works well with a reasonable large number of registered resources.

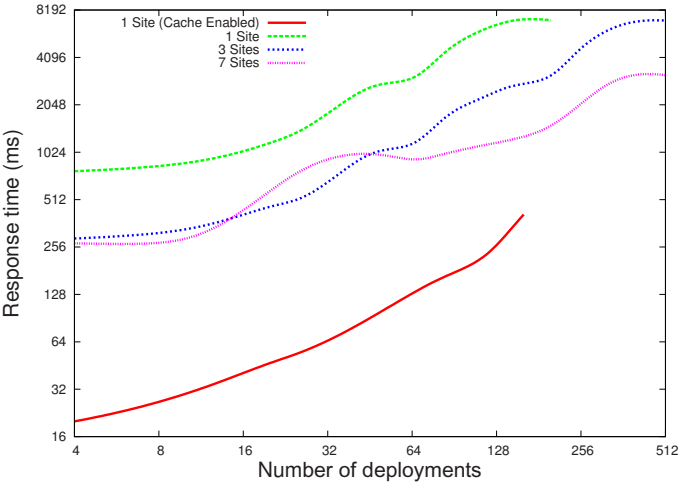


Fig. 4.10. Response time per activity *deployment* request with cache on 1 Grid node and without cache on 1, 3 and 7 Grid nodes.

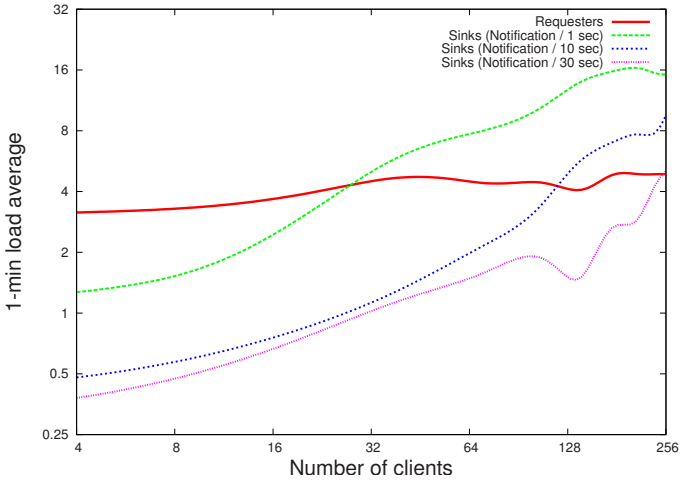


Fig. 4.11. Average 1-min CPU load with various number of concurrent client and notification sinks.

4.6 Related Work

A number of efforts have been made within the Grid community to develop automatic Grid resource management and brokerage solutions but very few of them are addressing the issue of resource management covering software components (activities) and their automatic deployment. A separation between meaning, behavior, and implementation of the Grid application components is described in [113]. The work in [80] matches a high-level application speci-

fication to a combination of available components. In contrast, *GLARE* provides a high-level application specification in a hierarchy of *activity types* and provides dynamic registration and automatic deployment of software components.

Pegasus [58] uses *Chimera* [79] and *Transformation Catalog* [57] for transforming an abstract workflow into a concrete workflow. The transformation Catalog is used to map a logical representation of an executable (transformation) to a physical representation, which describes its functionality and accessibility. The catalog uses MySQL as a centralized backend database. Chimera Virtual Data System [79] describes and stores data derivation procedures and derived data in a central database. It provides a special language interpreter that translates user requests. This system is useful for datagrid applications, but works with a dedicated querying mechanism. Pegasus uses Globus middleware services and automates replica selection. It does not provide automatic/on-demand deployment of software components.

GrADS [44] resource selection framework [108] addresses the discovery and configuration of physical resources that match with application requirements. It provides a declarative language using set matching techniques, which extend Condor matchmaking [128] and support both single and multiple resource matching. This system does not cover Grid application components. S. Decker et al describe in [170] Grid resource matching using semantic web technologies. This work proposes physical resource matching by using ontologies, background knowledge and rules. It highlights the need of semantic description of Grid resources and resource matching but does not address issues of performance and efficiency. Both systems [44, 170] do not cover software resources like Grid computational activities.

CrossGrid [37] provides a distributed component registry with peer-to-peer technology. It supports inter-registry communication for maintaining table coherency. *Grimoire* [96] extents UDDI [126] to provide invokable activities such as workflows or legacy programs.

GridLab capability registry [151], CrossGrid component registry [37] and MyGrid Grimoire [96] provide registries for static information of the Grid applications. UDDI [126] and Handle System [69] can be used to augment the *GLARE* but they have their own limitations. UDDI is a specification for distributed web-based information registries for web services but unsuitable for legacy scientific applications. Also it does not support dynamic updates. Handle System supports a very basic querying mechanism. Furthermore, it requires domain specific naming authorities to be registered in a root naming authority which is not managed efficiently. Globus MDS [39] provides a hierarchical aggregation framework for distributed Grid resources.

The main difference between *GLARE* and the systems described above is that while most of the above systems focus on discovering and brokering physical resources, *GLARE* framework focuses on logical resources (activities). Furthermore, *GLARE* provides dynamic registration, automatic deployment and on-demand provisioning and leasing of logical resources. The framework

is self-managed, fault tolerant, distributed and scalable. In contrast to MDS, *GLARE* provides a superpeer model based distributed framework which works well for large scale environments. It is implemented in Globus Toolkit 4 a state-of-the-art implementation of Web-Services Resource Framework [191].

Ka-tools [15], *LCFG* [13] and Quattor [134] provide auto deployment but mostly deal with configuration of physical nodes or perform OS cloning in a fabric. *SmartFrog* [1] requires specific components or wrappers to support automatic deployment of software components.

Open Grid Forum *CDDLM* working group [31] is addressing issues of automatic deployment and provisioning of Grid services with security and fault tolerance. The focus of this group is how to describe configuration of services, deploy them on the Grid and manage their deployment lifecycle (instantiate, initiate, start, stop, restart, etc.). The group is also standardizing APIs for this purpose. The focus of the group is a WSRF-based Grid services whereas *GLARE* targets both Grid services and legacy scientific applications.

4.7 Summary

In this chapter we have presented Grid *activity management* system that complements *GridARM* and is referred to as *GLARE*. In contrast to most of the existing resource management systems, which mainly focus on brokerage of physical resources, *GLARE* focuses on logical resources. It extends resource management to cover application components that can be part of distributed workflow applications.

GLARE is a Grid-level application component registration, deployment and provisioning mechanism that provides dynamic registration, automatic deployment and on-demand provision of application components (activities) that can be used to build Grid applications. Application components are described as *activity types* and *activity deployments*. By separating *activity types* from *activity deployments*, *GLARE* can shield the application developer from low level complexities of the Grid operating environment. *GLARE* automatically correlates *activity types* to a set of *activity deployments* that can then be selected for instance by workflow composition tools to create a workflow application for execution. Moreover, *GLARE* provides a mechanism in which new activities can be registered dynamically, installed automatically and provisioned and leased on-demand. We believe that this functionality is a major step forward towards an invisible Grid from the application developer's perspective.

The chapter starts with an introduction and a powerful motivating example that describes how *GLARE* improves and automates lifecycle management of activities in contrast to existing systems in which deployment of software components is manual or semi-manual. After introduction, the chapter proceeds with activity management model that defines how abstract and concrete descriptions of activities can be separated and correlated a runtime.

We have examined the performance of *GLARE* and compared its registries with GT4 *index service* (WS-MDS) and found it quite encouraging. We also exhaustively verifies the efficiency of the registration and provisioning mechanism with varying number of *activity types*, *activity deployments* and concurrent clients.

5

Allocation Management with Advance Reservation and Service-Level Agreement

A resource allocation for execution of an application in the future requires advance reservation*. The enssurance of agreed upon terms and conditions for allocation becomes possible with the provision of* service-level agreement (SLA)*. Advance reservation plays a significant role in improving the provisioning quality of a resource manager and predictability of behavior of Grid resources. However, advance reservation in the Grid has been largely ignored mainly due to under utilization concerns and lack of support for agreement enforcement. As part of* GridARM*, this chapter introduces a mechanism for advance reservation of Grid resources with new fairsharing algorithm and a practical solution for agreement enforcement based on off-the-shelf Grid middleware technologies.* Service-level agreement (SLA) *provides a set of terms and conditions that are to be agreed upon through a* negotiation *process. A negotiation mechanism for allocation of Grid resources is proposed in which a client can negotiate with the resource manager for improved compromises between its goals and resource capabilities offered by the resource providers. A resource management system works as negotiator for resources and resource providers. The proposed mechanism contributes towards better capacity planning and improvement in predictability.*

5.1 Introduction

In the Grid, the resource manager lacks control over the Grid resources. On the one hand Grid resources are controlled and administered by their local operating and management systems, whereas on the other hand Grid applications, which are the main consumers of the computing power, compete for resources. This makes resource management a non-trivial process: it has to make allocation of resource capabilities to contending applications with optimal capacity distribution and in accordance with constraints set by resource providers.

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 109–126, 2010.

One of the main tasks of a resource manager is on-demand provisioning of Grid resources, no matter where they reside or who owns them, resources should be available according to the required quality of service and policies of participants (requesters and providers). A pervasive Grid with higher usability is possible if its resource management becomes simple, smart, robust and invisible. This is a real challenge to achieve because of the unpredictability of distributed resources. A better control over underlying resources, without undermining their autonomy, can be possible with the help of *advance reservation* (Definition 21). A possible solution in which a user can reserve resources in order to make sure that resources will be available within the requested period of time.

However, *advance reservation* in the Grid has been barely addressed due to dynamic Grid environment, concerns about under-utilization of resources and lack of support for agreement enforcement. Firstly, the highly dynamic and unreliable Grid environment makes any assumptions concerning resource availability and possibility of any agreement enforcement extremely difficult if not impossible. Secondly, advance reservation in the Grid is considered as a mean to waste computing power. This makes provision of a reliable allocation with advance reservation as one of the greatest challenges in the Grid. Grid environments usually cannot guarantee that requests for future executions will be fulfilled within expected time intervals. Moreover, time-critical applications, that is an important class of Grid workflow applications, cannot be effectively executed without any guarantee of resource availability for an expected execution time.

Advanced reservation of Grid resources enhances the predictability of the *makespan* of Grid workflows (Definition 10) that leads towards better planning for execution. A. S. McGough et. al. state in [115] that:

> "Executing with reservations reduces the variance of the application's execution time if there is contention in the system. A reduced variance results in more predictable execution time".

In order to address these challenges, this chapter introduces advanced reservation and co-allocation of Grid resources along with a mechanism to deal with the dynamic Grid behavior, and a practical solution for agreement enforcement. The under-utilization concern is addressed with proper capacity planning and is presented in Chapter 6. Capacity planning is a forward looking process in which resources are allocated in such a way that overall resource utilization is optimized. However, in the Grid, capacity planning is not possible without advance reservation of underlying resources and sophisticated allocation algorithms.

A flexible mechanism is provided to plug-in different allocation algorithms suitable to a provider's policy. A set of allocation algorithms includes fairsharing, optimal utilization, load balancing and capacity planning. An allocation algorithm intends to improve Grid utility and load balancing whereas advance reservation improves availability and predictability. Predictability can

be considered as an important criterion, because of a substantial impact on the execution of time-constrained applications. Furthermore, a promising idea of *open reservation* is introduced that deals with the dynamic Grid behavior. An open reservation is a kind of priority provision, in which a promise is made that is fulfilled by allocating next available resource at runtime.

A new mechanism for agreement enforcement is provided and implemented based on the authorization framework of WSRF-compliant [191] Globus toolkit version 4 (GT4) [10]. According to this mechanism, a chain of authorization is possible by providing several *policy decision points (PDP)*. By exploiting this functionality, we introduce a special *policy decision point (PDP)* for the authorization of reserved allocations. The emergence of Askalon application development with execution environment and sophistication in the underlying middleware infrastructure has lead to the introduction of this solution for agreement enforcement that is very important to ensure advance reservation. Since *GridARM* is developed based on GT4, its extension for advance reservation can work with WS-GRAM and GT4 authorization framework, and introduces the special *policy decision point (PDP)* for advance reservation validation.

A client can negotiate with the *GridARM allocation management* for an allocation of a single resource (i.e. Grid node) or a set of resources in order to make a better compromise over the allocation.

5.2 Model

An *advance reservation* (Definition 21) is a form of priority provision by making allocation of resources sometime in future and ensuring that an allocated resource remains available during the reserved timeframe and terms and conditions agreed upon during a negotiation process are not voilated. In the Grid, where an application consists of multiple activities, simultaneous allocation of a set of heterogeneous resources is possible. Such kind of allocation is called *co-allocation*.

Definition 33. *A* co-allocation $\mathcal{L}^{co} \subseteq \mathcal{L}$ *is a set of allocations in which multiple resources are reserved for an application* $\mathcal{W} = \{\mathcal{I}^w, \mathcal{O}^w, \mathcal{A}^w\}$. *For each sequential activity* $\in \mathcal{A}^w$ *there is an allocation as advance reservation* $\in \mathcal{L}^{co}$. *Parallel activities may have multiple allocations as well.*

In a workflow, activities are executed in a sequential or parallel order. For sequential activities, multiple allocations need to be done in future.

5.2.1 Agreement

An advance reservation is represented in an agreement document that is encoded in XML format and used in communication and updation during a negotiation process (Definition 22). The agreement document comes in three types: template, offer, and agreement.

Definition 34. *An* agreement template $\mathtt{agr}_i \in \mathcal{LR}$ *is a logical resource that is used in the negotiation process for allocation of resources and for the representation of a contract between a requester and a provider. It consists of a set* $\mathcal{T}_i \subseteq \mathcal{T}$ *of terms and conditions on which participants negotiate to confirm a contract.*

The structure of an agreement document is depicted in Figure 5.1 that is adapted in accordance with *WS-Agreement* [87]. It consists of:

- An *agreement context* that covers details of negotiators, agreement metadata, expiration time etc.
- A set of functional properties called *service description terms* that provide information needed to instantiate or otherwise identify a service to which this agreement pertains and to which guarantee terms apply. These are further refined as *service description*, *service reference* and *service property* terms. Example may include an activity type (Definition 2.2.1) that is required during the agreed upon timeframe.
- A set of non-functional properties or quality of service constraints called guarantee terms. The guarantee terms specify the service levels that is to be agreed upon between parties. The *allocation management* may use the guarantee terms to monitor the service and enforce the agreement. An example may include minimum required number of processors or reliability of activity specified in service description terms.

An agreement template may be transformed into an *agreement offer* or an *agreement* during a negotiation process.

Definition 35. *An* agreement offer *or simply an* offer *is an* agreement document *that is proposed by the allocation manager to the requester. The offer is to be confirmed (accept/reject) by the requester or can be used as a new* template *for further negotiation.*

A client may accept or reject an offer, otherwise may opt for re-negotiation. Once an offer is accepted (by the client) it becomes a sealed *agreement.*

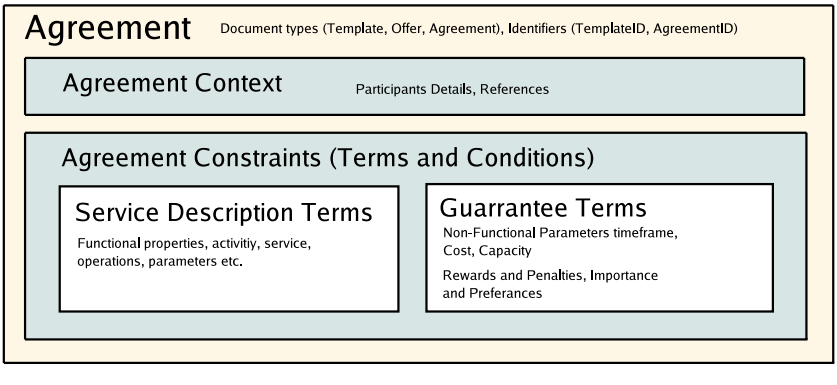


Fig. 5.1. Agreement Document Structure.

Definition 36. *An* agreement $\mathtt{agr} \in \mathcal{L} \subseteq \mathcal{LR}$ *is a* document *with agreed upon terms and conditions* $\in \mathcal{T}$ *defined in the form of* service description terms *and* guarantee terms *which are to be ensured by the allocator. An* agreement *refers to an advance reservation in which a possibly limited or restricted delegation of a particular resource capability is made available for a certain timeframe possibly along with a set of additional quality of service constraints or term and conditions* $\in \mathcal{T}$.

Some of the important attributes of an *agreement document* are as follows:

- *Timeframe:* It includes agreement *start time* $\mathtt{startt}(\mathtt{agr})$, *end time* $\mathtt{endt}(\mathtt{agr})$ and *duration* as guarantee terms. The difference between *start time* and *end time* could be greater than the specified $\mathtt{duration}(\mathtt{agr})$ in order to present flexibility by the client for negotiation.

$$\mathtt{startt}(\mathtt{agr}) - \mathtt{endt}(\mathtt{agr}) \geq \mathtt{duration}(\mathtt{agr})$$

  A smaller difference represents low flexibility and higher importance of the requested *duration*.
- *Agreement Type:* The *agreement type* attribute, as part of the *agreement context*, identifies a document as an *agreement template*, *agreement offer* or an *agreement*. An agreement template is used as a request that initiates a negotiation process.
- *Reference:* An agreement is represented as a stateful service that can be accessible with a service *endpoint reference* (EPR). The reference is used for future interactions with the system for manipulation of the agreement. The endpoint reference consists of a URI of the agreement management service and an identifier of the agreement resource.

$$\mathtt{ref}(\mathtt{agr}) = EndpointReference(EPR)$$

  A reference is a part of the *agreement context* and is different from a service reference that is an optional part of service description terms.
- *Identifier:* There are two identifiers, *TemplateID* and *AgreementID*. Once an agreement is confirmed, an *agreement identifier (AgreementID)* is created based on user credentials. The identifier must be consistent with the *ticket* identifier.
- *Flexibility:* This is an important attribute that is part of each term and condition and represents the level of importance of a term. This is used as part of negotiation and alternative offer generation process.

An *agreement* represents an advance reservation, however, besides the *timeframe*, it may include additional functional and non-functional properties $\in \mathcal{T}$. Once an agreement is sealed after the negotiation and accepted by the client, the system returns an agreement *ticket* which is used:

- to probe agreement status and to update an instance of agreement;
- to acquire reserved resources. Nevertheless, this is optional and required only if the owner of agreement is different from its initiator.

Definition 37. *An* agreement ticket *is referred to as a permit that is used to represent and authenticate an* agreement *and its ownership. It consists of agreement reference as well as agreement identifier i.e.* AgreementId.

5.2.2 Agreement Lifecycle

An *agreement template* passes through different states during its lifecycle from start of negotiation to the end of resource acquisition. It can be used to manage and monitored an agreement. Figure 5.2 shows the state transition diagram of an agreement. The transition of an agreement may include the following states.

- ***Created***: A template is created and the process of negotiation is started for a compromise over a resource allocation in future. The template is discarded after a few seconds if there is no further action. A newly created template is used as a request for the start of negotiation.
- ***Offered***: A created template is populated with an available timeslot along with other terms and conditions offered by the provider to the client. In this state, a further confirmation is required by the client, otherwise the allocation is discarded automatically within a configurable time interval that defaults to 40 seconds.
- ***Pending***: An agreement offer is tentatively held by the client. The client may get notification before its termination by the system. Notification is done well in time (defaults to 300 seconds) so that requester can acquire the offer or proceed for re-negotiation.
- ***Accepted***: An *agreement offer* is sealed once accepted by the client. Afterwards, only the owner is able to claim or cancel an accepted agreement by providing valid credentials or a valid ticket. The reservation system can terminate a confirmed reservation in case of contract violation.
- ***Cancelled***: This state shows that a confirmed agreement has been cancelled actively by the client.
- ***Active***: This state shows that the agreement *start time* has been reached and now it can be claimed by its owner anytime before its termination. In this state,

$$\texttt{startt(agr)} \leq \textit{current time} \leq \texttt{endt(agr)}$$

- ***Claimed***: An agreement is active and the reserved resource has been claimed and acquired by the client. An allocation with sealed agreement cannot be claimed if agreement is not in *active* state.
- ***Terminated***: This state shows that system has forcefully terminated the agreement. This could happen in case of a violation made by the client or if reserved resource is not claimed well in time.
- ***Completed***: The claimed agreement was successfully passed through its lifecycle and resource has been utilized by the client. In this state,

$$\texttt{endt(agr)} \leq \textit{current time}$$

A client can register for the event notification. Each time an agreement changes from one state to another, a registered client receives a state change notification. For example, a client can initiate re-negotiation with a *terminated* state notification or submit a job to the resource with an *active* state notification.

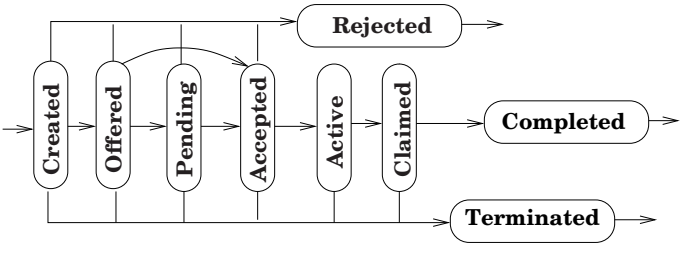


Fig. 5.2. State transition diagram of an agreement template.

5.3 Negotiation

Negotiation (Definition 22) is an alternative offer generation process, where one of the offer generated by allocation manager may be accepted, rejected or converted into a new template for renegotiation by the initiator.

By default a set of appropriate offers closer to the requested timeframe and other terms and conditions is generated on request. A default algorithm generates at least 3 offers with reference to the requested timeframe of a request `q` as follows:

1. an offer `agr` is generated only if possible with exactly the same timeframe as requested, that is

$$\texttt{agr} \iff \texttt{startt(agr)} = \texttt{startt(q)} \wedge \texttt{endt(agr)} = \texttt{endt(q)}$$

2. an offer `agr` with time slot available with the requested duration earliest possible after the current time, that is

$$\texttt{startt(agr)} \geq \textit{current time}$$

3. An offer `agr` with time slot available with the requested duration earliest possible after the requested timeframe, that is

$$\texttt{startt(agr)} \geq \texttt{startt(q)}$$

4. An offer `agr` with time slot available with requested duration latest possible before the requested timeframe, that is

$$\texttt{endt(agr)} \leq \texttt{startt(q)}$$

A client selects a suitable offer and does one of the following:

- accepts selected offer
- rejects selected offer
- converts selected offer into a new template, readjust some of the constraints and renegotiates

A possible client-side algorithm for negotiation may be similar to the one shown by Algorithm 6.

Algorithm 6 A Pseudo Code of a Possible Client Negotiator.

```
client_negotiate()
make an agreement template
while not accepted do
  send agreement template to the allocator as a request for negotiation
  offers := get agreement Offers generated by the allocator
  offer := select an offer closer to the request (template)
  if offer is_acceptable then
    accept offer
    set accepted
  else
    convert offer into a new template
    if template can not be negotiated then
      break // Negotiation cannot be continued
    end if
    continue // for re-negotiation
  end if
end while
```

This interaction mechanism is also useful for *co-allocations*, where a *co-allocator* can re-adjust various attributes while considering the dependencies between different selected resources or allocation offers.

In its simple (default) form, an *allocator* can generate offers with reference to the timeframe in one of the three ways called an *attentive*, *progressive* and *share-based*. The offer generation process is depicted in Algorithm 7 and described as follows:

5.3.1 Attentive Allocation

The *attentive* allocation generation algorithm always offers requested slot if available exactly as requested, otherwise it generates alternative offers according to the available slots closer to the requested timeframe. While generating alternative offers, it tries to keep the reserved segments as minimum as possible, by proposing alternative options which are overlapping or adjacent to the existing reserved slots, i.e. it tries to find slots available in parallel, latest

Algorithm 7 A Pseudo Code of an Offer Generator.

```
generate_offers()
Input: q, g // An allocation request and node g ∈ G selected for an allocation ∈ L
Output: OF // A set of offers generated
d := endt(q) − startt(q) // d is duration, startt(q) is start time, and endt(q) is end time of q
if allocation_mode is_SHARE_BASED then
  g := select(q) // Select a node according to Algorithm 3
end if
OF := ⊘
L^g := A set of current allocations (reservations) L^g ⊆ L ordered by start time
for all alloc ∈ {{q} ∪ L^g} do
  // generate an offer exactly as requested or a one with start time earliest possible after current time
  offer := generate_offer(alloc)
  if offer ≠ ⊘ then
    OF := OF + {offer}
    break
  end if
end for
i := Last index of an allocation alloc ∈ L^g such that startt(alloc) ≤ startt(q)
while i > 0 do
  offer := generate_offer(alloc_i)
  if offer ≠ ⊘ then
    OF := OF + {offer}
    break
  end if
  i := i − 1
end while
i := First index of an allocation alloc ∈ L^g such that startt(alloc) > startt(q)
while i ≤ |L^g| do
  offer := generate_offer(alloc_i)
  if offer ≠ ⊘ then
    OF := OF + {offer}
    break
  end if
  i := i + 1
end while
return OF // Generated set of offers
```

```
generate_offer (alloc)
for all st ∈ {startt(alloc), startt(alloc) − d, endt(alloc)} do
  create offer with startt(offer) = st, endt(offer) = st + d
  if slot for offer is_free on node g then
    if allocation_mode is_PROGRESSIVE and user share is_consumed then
      continue
    end if
    return offer
  end if
end for
return ⊘
```

before or earliest after an existing reservation for all the available processors ($\mathcal{P}$) in a node.

Example 16 (Attentive Allocation).
Lets a node having 2 processors and 5 reservations (solid boxes), as shown in Figure 5.3, and what happens when another slot is requested. Dotted boxes show possible alternative offers which are generated and offered to the client by the attentive algorithm (offers before the requested timeframe are also generated, if possible). It is then up to the client to select one of the offers or re-negotiate. □

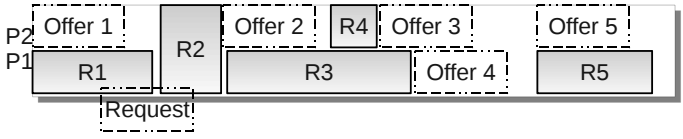


Fig. 5.3. Possible options offered by attentive algorithm when a node with two processors was occupied by five accepted allocations (reservations).

Offers can be generated by automatically adjusting the timeframe and other constraints (e.g. processors) in such a way that the overall requested QoS is optimized. For this purpose, we propose adjustment like:

- Change 10% of the requested duration by adding or removing one processor,
- or adjust the requested timeframe by scaling the attributes according to the speedup, which can be defined using different theoretical models like Amdahl's model [12] or using a database with application benchmarks.

The allocations for multiprocessor jobs is possible though the main focus of this chapter is to generate allocations for single processor jobs.

5.3.2 Progressive Allocation

The *progressive* algorithm is an extension of the *attentive* algorithm that considers fairness as well. It attempts to fairly distribute available capacity of resources among competing clients instead of allowing a single client to reserve the entire capacity of a node $\in \mathcal{G}$. Having said that, the minimal time when the reservation can be established for a specific timeframe depends on the number and duration of reservations already made by the client during that timeframe. This is done by introducing a new restriction on the number of processors ($\in \mathcal{P}$) which is configurable for a Grid node $\in \mathcal{G}$. Furthermore, it is also possible to allow number of allowed reservations during a certain time window. The progressive approach provides a more fair distribution so that multiple clients can get better chances to obtain offers that fit to their requirements.

Example 17 (Progressive Allocation).
Consider a node with several allocations as shown in Figure 5.4. The allocation policy is set as progressive with a fixed duration of *time windows*. According to the policy, a client can have maximum two allocations in each time window. Lets a client $\mathtt{c}_1$ makes an allocation request `Request`, shown as a rectangle with dotted lines in Figure 5.4. Based on this `request` three offers can be generated according to attentive algorithm. Nevertheless, with progressive allocation policy, only `offer 3` is valid because `offer 1` and `offer 2` fall in first *time window* that has already two reservations $R1, R3$ from the same client $\mathtt{c}_1$, and thus, the first two offers are ignored.

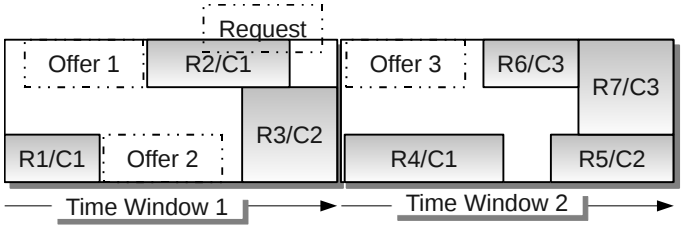


Fig. 5.4. Progressive allocation: two time windows each can have maximum 2 allocations per client.

□

5.3.3 Share-Based Allocation

In contrast to *attentive* and *progressive* allocations, in which allocation offers are generated for a user's selected *node* (Definition 11), *share-based allocations* are generated based on the proportional share of the node to be reserved. First a Grid node is selected by the *allocation manager* and then attentive approach is applied for offer generation. The main difference between attentive and share-based approach is the selection of a node.

5.4 Implementation

The proposed *GridARM allocation management* system consists of two main components: a node-level component called *allocator* and a Grid level components called *co-allocator*. This is shown in Figure 5.5). The *allocator* is responsible for the negotiation and provision of advance reservation of a Grid node, whereas co-allocator handles allocations of multiple nodes for a single workflow application. Both allocator and co-allocator are implemented as WSRF Grid services (WS-Resources), based on the Globus Toolkit 4 (GT4). A *co-allocator* works on a superpeer node in coordination with the *resource*

manager and negotiates with multiple underlying allocators in order to generate co-allocations.

5.4.1 Allocator

An *allocator* is to be associated with a node and further consists of two sub components: an *allocation manager (AM)* and an *AuthzManager*. The *allocation manager (AM)* generates allocation offers by invoking a specific allocation algorithm. An *allocation manager* provides a mechanism in which different algorithms can be plugged-in and configured according to the resource usage constraints and provider's strategy. The *AuthzManager* authorizes whether or not a client should be permitted to acquire the reserved resource e.g. by submitting and executing a job.

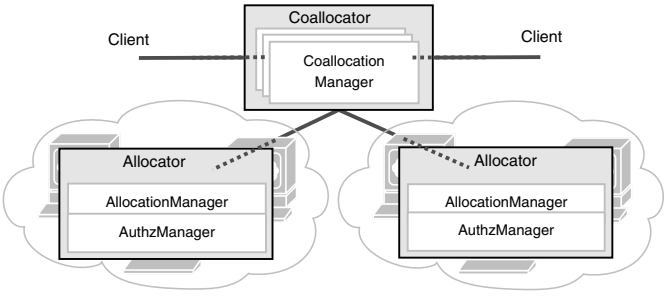


Fig. 5.5. Allocation Management System Architecture.

In order to ensure sophisticated integration of the local resource management with reservation system, the *GridARM* chooses not to change the low level mechanism of the Job Submission service (as proposed in [55]), but rather exploit the customizability of the GT4 Job submission service, that is WS-GRAM [41, 10], which allows addition of multiple customized resource authorization policies. The *AuthzManager* acts as a special *policy decision point (PDP)* for WS-GRAM and ensures that whether or not a resource was actually reserved by the client through the *allocation manager*.

5.4.2 Co-allocator

A *co-allocator* is a high-level component that covers a set of *allocator*s and runs on a superpeer node. A superpeer node is one that works as a frontend or root node for a group of nodes and can coordinate with one or more other superpeer nodes (Section 3.4.2). In such a way, if a *GridARM co-allocator* cannot find an answer within its own group, then it refers to the peer services running on other superpeer nodes for an answer.

A *co-allocator* works as a factory service of a *co-allocation manager (CM)* and instantiates a separate *CM* for each request. The *CM* then handles further negotiation between a client and underlying *allocator*s, and performs ongoing monitoring of the accepted *co-allocation*. Furthermore, a *co-allocator* can interact with the *GridARM resource manager* for candidate selection Section 3.3.4 and may filter out nodes for which hard constraints (i.e. requests with $flexibility = 0$) cannot be fulfilled. The *flexibility* is an attribute that describes importance of a term and needs to be set by requesters for each term and condition.

After making a successful reservation, the clients can submit their jobs for execution within the reserved timeframe. By default, only a client with valid credentials and having a valid reservation can submit a job. Moreover, a *CM* generates an *agreement ticket* which can be delegated to other clients who can acquire associated reservations by presenting a ticket. A client is not authorized if it fails to provide a valid *ticket* or fails to prove itself as an owner of the reservation.

Both *allocator* and *co-allocator* provide a flexible mechanism to plug-in different offer generation algorithms. In the offer generation process, an *allocator* tries to maximize resource utility while fulfilling client's requirements. Only free slots which are in accordance to the reservation policy are offered on request.

5.4.3 Agreement Enforcement

A special *policy decision point (PDP)* for the authorization of reserved resources is the essential component of *GridARM* allocation management that exploits customizability of the WS-GRAM [41] (Section 2.3.3) in which multiple authorization points (PDPs) can be integrated. *GridARM* introduces this additional PDP called *ReservationPDP* for the authorization of clients, it works as part of the chained authorization points invoked by WS-GRAM. As depicted in Figure 5.6, this additional PDP interacts with *AuthzManager* and gets verification whether or not the client has advance reservation at that particular time. The *ReservationPDP* is used for the enforcement of an agreement.

AuthzManager may be configured with a customized policy, for instance, executions can be performed only when the resource is reserved by the user or if the resource is not reserved at all by anyone during the requested timeframe. If an application is not finished within the reserved timeframe, the application should be either terminated or suspended. Termination of an application that is about to complete could be counter productive, especially when application execution time is much longer. On the other hand, suspension of an application requires low-level system interaction, which is an open research topic.

A reservation made by the *GridARM allocator* is independent from the underlying *local resource manager (LRM)*, e.g. PBS [11], LSF [132], SGE [168]. This means that a job can be submitted through WS-GRAM to any of the

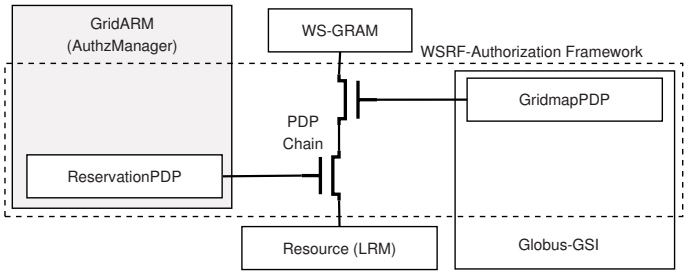


Fig. 5.6. A Policy Decision Point (PDP)-chain with a Special ReservationPDP for WS-GRAM.

LRM and is honored only if proper reservation is made by the client. Each time WS-GRAM is invoked, it interacts with *AuthzManager* through the special *Reservation PDP* for verification of the caller.

In some cases (depending on the Grid node policy and configuration), it may be possible that a client bypasses WS-GRAM and submits jobs directly to any of the deployed LRMs. In such a case the reservation will not be enforced. This is, in fact, a well-known open issue. One possible enhancement is to provide low-level reservation with the help of LRMs (e.g. based on Maui [98]). However, this will break the generality of the proposed solution. As WS-GRAM aims to provide a higher-level job submission functionality abstracting from various LRMs, integrating advance reservation mechanisms directly to interact with WS-GRAM (i.e. making a LRM-independent reservation service) is a more portable and practical approach for agreement enforcement in the Grid operating environment.

5.4.4 Priority Provision

Priority provision is an *open reservation* with soft allocation of resources. In open reservations, actual node binding either can be changed over time or deferred until application runtime. The resources are allocated to an application, but real binding is shielded from the client. In this way, a promise is made that a certain capability will be available at a specific time in the future but without making assignment of any specific node to the client. The assignment of the next available node is done at runtime. That means that a next available node that fulfills the requested QoS constraints is allocated at runtime.

The priority provision is a way of keeping the promise of advance reservation by dynamically associating physical resources with allocations in an underlying unpredictable and dynamic Grid environment. Furthermore, it allows reservation of logical resources as well. This is an important feature, which according to our understanding, has not been considered for advance reservation so far in any other Grid computing infrastructure.

5.4.5 Standards Adaptation

GridARM adapts *WS-Agreement*, a proposed Grid Resource Allocation Agreement Protocol [87], for contract specification. Allocations are composed as WS-Agreement documents and maintained in the form of WS-Resources [191]. JSDL is another proposed standard for job submission description [100] that consists of a set of constructs which are used to specify constraints as part of the WS-Agreement specification. Furthermore, constraints like *maximum number of allowed processes* and *job termination time* can be specified as part of the job description in JSDL, and can be used to further strengthen the process of agreement enforcement by exploiting these parameters.

5.5 Experiments and Evaluation

The *GridARM allocation management* is developed as a set of coordinating WSRF-complaint middleware Grid services. It provides reservation in the form of agreement documents and managed as WS-Resources [191], which makes the service as efficient as the underlying infrastructure. We integrated allocation management in Askalon runtime environment and tested overhead of the system by performing experiments in Austrian Grid [33]. Table 5.1 depicts average overhead of different operations of the system for making an allocation including initialization, negotiation, confirmation, and authorization. It shows that the negotiation, which also involves a compute intensive offer generation process, is a bit expensive whereas confirmation does not add any significant overhead, as it deal with existing reservation instances that are manageable as WS-Resources.

Table 5.1. Average time overhead of different reservation service operations.

Function	**Time (MS)**
Initialization (template creation)	110
Negotiation (Attentive)	190
Negotiation (Progressive)	194
Confirmation	30.5
WS-GRAM (Default)	600
WS-GRAM (with ReservationPDP)	636

Table 5.1 (lower part) shows the overhead of job submission to WS-GRAM both with and without *ReservationPDP*. We performed this test by submitting a small job to WS-GRAM in *"quiet"* and *"batch"* mode, that means the client did not wait for the completion of job and returned immediately after successful submission. The average overhead of the *ReservationPDP* is $36ms$ per job, which is just 6.0% of the total submission overhead by WS-GRAM.

Table 5.2. Average response time per transaction for different concurrent users.

Concurrent Users	2	4	6	8
Response Time (ms)	105	180	248	315

Table 5.2 shows an average response time for a single negotiation session with a simple offer generation algorithm with varying number of concurrent clients. A simple offer generation algorithm generates allocations exactly as requested by assuming unlimited resource capacity. Response time increases with number of users due to offer generation algorithm which is compute intensive. As most of the Grid applications run for a longer duration, therefore the overhead of allocation management system is quite negligible even for the time critical Grid applications. These experiments were performed on AMD Opteron 64*bit* nodes located on a lightly loaded network with a maximum latency between two computers of about 2 milliseconds.

Figure 5.7 shows an average response time for a single *negotiate-confirm* session with varying number of pre-existing confirmed allocations. It is important to mention that number of time segments varies from 3 to 10 during the course of experiments, as number of generated offers depends on the number of segments. Both attentive and progressive algorithms give the same performance but average response time increases slightly with the increase in reserved slots. The simple algorithm is slightly efficient. This means that the main overhead lies in communication and WSRF part and not in offer generation algorithm. Confirmation overhead is almost consistent as it deals with already created agreement resources.

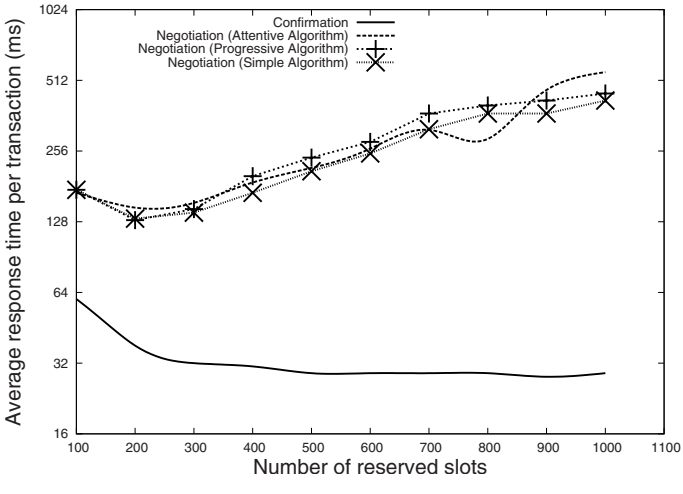


Fig. 5.7. Average response time for a single negotiation and confirmation session with different number of existing reservation instances.

If an application does not finish within the reserved timeframe, the allocator will detect a violation of the agreement. In such a case, the tasks should be either terminated or suspended. Termination of an application which is about to complete could be counter productive especially when application execution time is longer. On the other hand, suspension requires low-level system interaction, which is still an open research topic.

In our work published in [188], we have demonstrated that advance reservation can have a major impact on execution time and can increase considerably predictability of a Grid environment. We also showed the importance of requesting for longer reservation time periods, because of the low reliability of execution time predictions. This may provide very high probability that the execution will finish within the reserved time. However, this happens at the cost of lower resource usage and fairness of resource distribution. Fairness can be increased by employing a special reservation strategy, for instance the progressive reservation strategy,which gives the best results when the Grid is not saturated. Finally, we showed that the strategies without reservations and with progressive reservations can be applied together, and result in good performance and fairness.

5.6 Related Work

Numerous researchers have investigated approaches of resource reservation for networks [49], CPUs and other resources. *SNAP* [40] presents a resource management model in which resource interactions are mapped onto a set of SLAs. It defines a negotiation protocol for SLAs.

The Globus Architecture for Reservation and Allocation (*GARA*) [73] provides a mechanism for allocation of a resource preceded by an additional step of reservation. Open Grid Forum [129] is actively working on standardization of Service Level Agreement with the help of *WS-Agreement*.

The Maui [98] scheduler is an advanced job scheduler for cluster systems in which an advance reservation scheme makes it possible to manually allocate local resources in the future, however it does not support negotiation and allocation offer generation.

The KOALA [117] Grid scheduler supports co-allocation. In order to synchronize start-times of multiple jobs to be executed concurrently, it predicts possible start-time based on transfer rate of input files of the jobs. Once start-time is predicted, KOALA makes advance reservation. However, advance reservation is done only if the underlying local resource manager supports advance reservation.

Thomas Roebliz et. al. presents an elastic Grid reservation with user defined optimization policies and co-reservation with concept of virtual resources in [143].

The work described in [55] introduces a resource broker that supports advance reservations. However, in order to make it work, the proposed reserva-

tion enforcement mechanism requires modifications in the Grid operating environment i.e. low-level middleware services like GRAM [41] and GridFTP [9]-client/server.

In contrast to above mentioned reservation and negotiation mechanisms, the *GridARM allocation management* provides not only a flexible multi-phase negotiation mechanism along with advanced reservation of a single as well as multiple resources, but also generates multiple alternative offers. This reduces communication overhead required for multiple interactions during the negotiation and enables participants to seal a contract efficiently. The implementation is WSRF complaint and works with WS-GRAM. This makes our system LRM independent which can work with multiple LRMs. Furthermore, we propose provision of LRM-specific drivers which can be plugged in with our system to provide low level resource specific reservation so that a resource could not be exploited by bypassing WS-GRAM. It provides a generalized mechanism for agreement enforcement that is practical, efficient, and does not require any modification in the Grid operating environment.

5.7 Summary

This chapter introduces allocation management with advance reservation and service level agreement of Grid resources. A client can negotiation for reservation of a resource capability with the *allocation management* system. The system generates multiple alternative options that enables to reach and seal an agreement efficiently. The system consists of distributed middleware Grid services which coordinate with each other in order to provide simple as well as compound reservations. We introduce an additional Policy Decision Point (PDP) in the authorization chain to be invoked by the WS-GRAM. In this way the authorization is done independent of the local resource manager (LRM). In the negotiation process the system can provide best offers matching to client requirements. Furthermore, the negotiation is a multiphase process in which requesters and providers can adjust their constraints, QoS and offered resource capabilities.

6

Optimizing Multi-Constrained Allocations with Capacity Planning

This chapter introduces capacity planning that exploits advance reservation mechanism. Capacity planning plays a critical role in management of an infrastructure for optimized utilization of perishable resources. This applies to the Grid as well. Once the time has passed the computing power is perished. However, in the Grid, capacity planning is largely ignored due to the dynamic Grid behavior, multi-constrained contending applications, lack of support for advance reservation and its associated challenges like under utilization and agreement enforcement concerns. These issues force a resource manager to make resource allocations at runtime with reduced quality of service (QoS). To remedy these, we introduce Grid capacity planning and management with negotiation-based advance reservation and multi-constrained optimization. A 3-layer negotiation protocol is introduced along with algorithms that optimize resource allocation in order to improve the Grid utility. We model resource allocation as an on-line strip packing problem and introduce a new mechanism that optimizes resource utilization and other QoS parameters while generating contention-free solutions. We have implemented the proposed solution and experimented to demonstrate the effectiveness of our approach.

6.1 Introduction

Capacity planning plays a critical role in management of an infrastructure for optimized utilization of its perishable resources. The Grid is no exception as its computing power is perished once time is passed. The computing power of the Grid has a great potential for proper capacity planning to be provided as part of the resource management. *Resource management* is critical in making Grid infrastructure reliable and pervasive, and in delivering resources on-demand while dealing with their heterogeneity, dynamic behavior, and association with different trust domains. As the resources are controlled and administered locally on a Grid node, capacity management for the entire Grid becomes non-trivial and requires a sophisticated resource

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 127–153, 2010.

management as part of complex middleware infrastructure. The problem becomes even more challenging with the rapid growth of Grid resources and applications, where *allocation management* has to allocate the Grid resource capacity among applications contending for scarce resources. Allocations are requested with multiple parameters and the *allocation management* has to perform multi-constrained optimization.

This challenge leads to the requirement of a robust allocation management with a sophisticated capacity planning and management for optimized resource utilization. *Capacity management* is a strategic process that focuses on the present, whereas *capacity planning* is a strategic forward looking process of monitoring, understanding, and reacting to the client's behavior in order to maximize the global utility. However, in the Grid, capacity planning and management is largely ignored due to the dynamic Grid behavior, multi-constrained applications, lack of support for advance reservation and its associated challenges like under utilization concerns, and the non-supportive environment for enforcement of an agreement. These problems force an *allocator* to employ adhoc solutions with limited view of the overall Grid capacity available along a time horizon. Any ad-hoc solution mostly results in resources with wasted capacity and Grid applications with reduced utility.

To remedy this, *GridARM* introduces a capacity planning and management mechanism that supports optimized allocation and negotiation-based advance reservation of Grid resources. Advance reservation ensures that a certain resource capability will be available at some time in future thus provides a more predictive environment suitable for the planning. It anticipates adequate needs of its clients and protects a proper share of the resource capacity for the clients who could be more profitable in the future. The *resource manager* works as resource provisioner and performs reservations for applications on behalf of resource provider through a negotiation process. We introduce a 3-layered cooperative negotiation protocol that is used to efficiently reach an acceptable agreement. Furthermore, the dynamic nature of the Grid is handled with a priority provisioning mechanism, in which a certain capacity is reserved in advance but bound later-on and provisioned on-demand. The negotiation process involves generation of multiple allocation offers based on different QoS parameters. The first layer of negotiation protocol deals with allocation of a single Grid node. We model it as an on-line strip packing problem [38] and introduce a new solution for it. The second layer deals with co-allocation of multiple Grid nodes. It receives a set of allocation offers generated by the first layer for a set of available nodes and then generates a set of co-allocation offers with optimized global utility. We frame co-allocation as *constraint satisfaction problem (CSP)* [152] and employ a new approach to solve it. The third layer eliminates contentions that may be introduced during the first and second layer of negotiation.

We have implemented the proposed system and demonstrated through experiments that the proposed system accepts more allocation requests and ensures improved resource utilization with the capacity management and planning.

6.2 System Model

The Grid resource allocation corresponds to on-demand provisioning of resources for Grid workflows. The allocation process first selects matching nodes and then determines allocatable time slots on each node during which an activity can execute. Finally, it makes a combination of all available time intervals on all nodes. In this way, the workflow may run efficiently along with optimized resource utilization. The resource selection mechanism is already described elsewhere. This chapter models resource allocation as multiple-constrained optimization problem.

The allocation management is described in previous chapter that consists of two main components: an *allocator* that performs allocation of a single node, and a *co-allocator* that performs allocation of multiple nodes for a single Grid workflow application. A *co-allocator* accepts requests from the clients and generates alternative co-allocation offers. It instantiates a *co-allocation manager (CM)* for each request that handles further negotiation between the client and the underlying allocators, and then performs ongoing monitoring of the agreements sealed through the negotiation. The CM negotiates with clients as a resource trader and with allocators as a cooperative negotiation mediator. Cooperative negotiation enables the system to generate offers efficiently with optimized as well as contention-free capacity distribution. A *co-allocation request* corresponds to a workflow and may consist of a set of multi-constrained *allocation requests*, each for a single activity as part of a workflow.

In previous chapters, allocation has been modelled with reference to the entire node. This chapter models the possibility of allocating a part of the whole (node $\in \mathcal{G}$) with reference to the logical components i.e. the activities.

6.2.1 Allocation Problem

Grid resource allocation as defined in is a problem of assigning a set of available nodes $\in \mathcal{G}$, each with a scarce capacity (e.g. a number of processors $\mathcal{P}$) to a set of *co-allocation request*s

$$\mathcal{CQ} = \{\mathcal{CQ}_1, ..., \mathcal{CQ}_n\} \subseteq \mathcal{Q}, n \in \mathbb{N}$$

each for a workflow application $\in \mathtt{W}|\mathtt{W} = \{\mathcal{W}_1, ..., \mathcal{W}_n\}$ by a set of clients $\mathcal{C} = \{\mathtt{c}_1, ..., \mathtt{c}_n\}$. Here

$$\mathcal{CQ}_i = \{\mathtt{q}_{i,1}, ..., \mathtt{q}_{i,n_i}\} \; and \; \mathcal{W}_i = \{\mathcal{I}_i, \mathcal{O}_i, \mathcal{A}_i, \mathcal{V}_i\}$$

where $i \in n$ and $n_i \in \mathbb{N}$ may be different for different requests $\mathcal{CQ}_i \in \mathcal{CQ}$. The request $\mathtt{q}_j \in \mathcal{CQ}_i$ is referred to as a single *allocation request* for a specific *activity* $\mathtt{a}_j \in \mathcal{A}_i$, which may have some dependencies $\mathcal{V}_i$ on other requested activities $\in \mathcal{A}_i$ in a *co-allocation request* $\in \mathcal{CQ}_i$. Each *allocation request* $\mathtt{q}_j \in \mathcal{CQ}_i$ requires a certain capacity $\mathcal{P}_j \in \mathcal{P}$ for a specific duration of time

$$\texttt{duration}(\texttt{q}_j) = \texttt{endt}(\texttt{q}_j) - \texttt{startt}(\texttt{q}_j)$$

and may have the potential for varying utility $\mathcal{U}(\texttt{q}_j)$ depending on the application constraints $\in \mathcal{T}_i$.
Figure 6.1 depicts the format of a request $\in \mathcal{CQ}_i$ that asks for of a set of activities $\mathcal{A}_i \in \mathcal{A}$ along with a set of constraints

$$\{\mathcal{T}_{i,1}, ..., \mathcal{T}_{i,n_i}\}, n_i \in \mathbb{N}$$

in the form of service description terms, and of a context referring to the participants. The right side of Figure 6.1 shows the format of a constraint $\in \mathcal{T}$ that includes boundary values of a named constraint and flexibility $\texttt{flex}(\texttt{t}_j) \in \{0..10\}$ that represents client's willingness for negotiation. The term *flexibility* is similar to a reciprocal of the term *importance* defined in WS-Agreement [87].

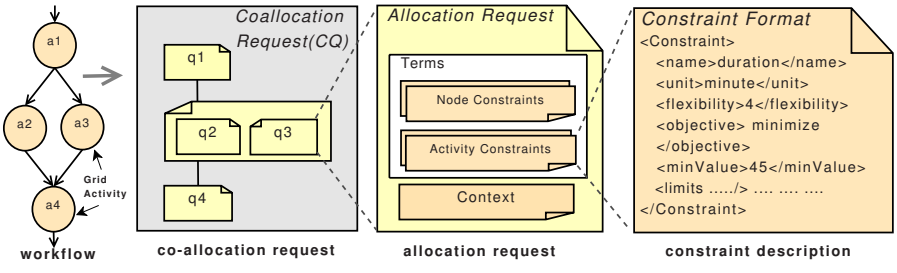


Fig. 6.1. A co-allocation request that corresponds to a workflow.

The goal is to maximize the global utility $\mathcal{U}$, suggesting the right allocation options for applications, and achieving an optimal compromise over constraints of the negotiators. By considering the time horizon $\texttt{T}$, the capacity of the Grid $\mathcal{G}$ can be modelled as $\bigcup_{\texttt{g}_i \in \mathcal{G}} \mathcal{P}_i \times \texttt{T}$ where $\texttt{T} \in \texttt{time}$ is the time dimension that represents a planning horizon. $\mathcal{P}_i$ represents capacity of a node $\texttt{g}_i \in \mathcal{G}$ in terms of processors and $\mathcal{P}^g = \sum_{i=1}^{n} \mathcal{P}_i$ is the total capacity of the Grid $\mathcal{G}$.

The global utility

$$\mathcal{U} = \sum_{i-1}^{n} \mathcal{U}_i \; with \; \mathcal{U}_i \in 2^{|\mathcal{P}^g|+\texttt{T}} \mapsto \mathbb{R}$$

can be maximized by generating co-allocation offers

$$\mathcal{CL} : \mathcal{CL} = \{\mathcal{CL}_1, ..., \mathcal{CL}_n\} \; with \; \mathcal{CL}_i \mapsto 2^{|\mathcal{P}^g|+\texttt{T}}$$

such that the utility is maximized and there is no overlap of allocations, i.e.

$$\sum_{i=1}^{n} \mathcal{U}_i = \top \wedge \bigcap_{i=1}^{n} \mathcal{CL}_i = \varnothing$$

$2^{|\mathcal{P}^g|+\mathtt{T}}$ indicates the power-set of the available allocation options (maximum allocations possible on resource and time horizon ($2^{|\mathcal{P}^g|+\mathtt{T}}$)). A co-allocation $\mathcal{CL}_i$ consists of a set of allocations as:

$$\mathcal{CL}_i = \{\mathtt{alloc}_{i,1}, ..., \mathtt{alloc}_{i,n_i}\}, n_i \in \mathbb{N}$$

n_i is different for different $\mathcal{CL}_i$. Note that the co-allocation process is a total function $f : \mathcal{CQ}_i \mapsto \mathcal{CL}_i$ that maps each request $\mathtt{q}_j \in \mathcal{CQ}_i$ to an allocation $\mathtt{alloc}_j \in \mathcal{CL}_i$.

As resource requirements may change over time, or a particular pattern of resource usage may be needed to obtain a utility, allocation options are reduced on both the resource and time dimensions, hence the need for a planning horizon. Increasing the number of resources or the time horizon significantly increases the overall complexity of the allocation problem, which is *NP-complete* [152, 38].

6.2.2 Multi-Constrained Optimization

Optimization is done along a planning horizon: enough capacity is allocated for each single application in the requested order while planning for maximum resource utilization. Depending on QoS-constraints, the utility function associated with each application and resource may change. Generally, increasing the resource capacity or allocating a resource with closer match to requested QoS, improves the application utility. We model application utility as a function of distance between (co-)allocation requested and actual option offered. If $\mathtt{dist}(\mathtt{t})$ is the distance between the `requested value` and `offered value` of a constraint $\mathtt{t} \in \mathcal{T}_i$ of a single allocation $\mathtt{alloc} \in \mathcal{CL}_i$, then the utility $\mathcal{U}(\mathcal{W}_i)$ of an application $\mathcal{W}_i$ is the aggregated utility of all allocations i.e.

$$\mathcal{U}(\mathcal{W}_i) = \sum_{\mathtt{alloc}_j \in \mathcal{CL}_i} U(\mathtt{alloc}_j)$$

where

$$U(\mathtt{alloc}_j) = \sum_{\mathtt{t} \in \mathcal{T}_j} \mathcal{U}(\mathtt{dist}(\mathtt{t}))$$

and

$$\mathcal{U} = \sum_{\mathcal{W}_i \in \mathtt{W}} \mathcal{U}(\mathcal{W}_i)$$

is the global utility.

An allocation is generated, by using a set of objectives defined by the utility functions, to assign resource capacity to an application. Each function is expressed in terms of application's utility or in terms of resource utility which is the function of its offered QoS, for instance, the *capacity*, *cost*, and *timeframe*. The application utility is derived by aggregating the distances between ideal and real values of all QoS constraints. For a specific constraint $\mathtt{t} \in \mathcal{T}$, if

$\texttt{requested\_value(t)}$ is the required (or ideal) value, $\texttt{offered\_value(t)}$ is the offered (or real) value, and $\texttt{flex(t)}$ is the level of flexibility for negotiation, then distance $\texttt{dist(t)}'$ for a constraint $\texttt{t}$ is calculated as:

$$\texttt{dist(t)}' = \frac{\texttt{requested\_value(t)} - \texttt{offered\_value(t)}}{\texttt{flex(t)}}$$

Here $\texttt{flex(t)} \in \{0, ..., 10\}$, where 0 means no flexibility over a given constraint. This makes it a *hard constraint* that has to be fulfilled in order to make an agreement. On the other hand, 10 shows maximum flexibility for negotiation (a *soft constraint*). A negative distance may have a different meaning for different constraints, for instance, in case of cost, a negative distance reflects that the offer is unacceptably expensive, whereas in case of capacity, a negative distance shows that more capacity is offered than requested that makes an offer an attractive offer and could temptate the client to accept the attractive offer. To generalize, we use the term *objective* of the constraint $\texttt{objective(t)} \in \{\texttt{minimize, dontcare,maximize}\} \equiv \{-1, 0, 1\}$ that tells whether or not a negative distance matters. Depending on $\texttt{objective(t)}$ and $\texttt{flex(t)}$ of a constraint $\texttt{t} \in \mathcal{T}$, the distance $\texttt{dist(t)}$ can be refined as:

$$\texttt{dist(t)} = \begin{cases} \infty & \texttt{flex(t)} = 0 \wedge \texttt{requested\_value(t)} - \texttt{offered\_value(t)} \neq 0 \\ 0 & \texttt{objective(t)} = \texttt{maximize} \wedge \texttt{dist(t)}' < 0 \\ 0 & \texttt{objective(t)} = \texttt{minimize} \wedge (-1 \times \texttt{dist(t)}') < 0 \\ |\texttt{dist(t)}'| & \texttt{otherwise} \end{cases} \tag{6.1}$$

The $\texttt{dist(t)} = \infty$ leads to a situation where no solution is possible with available capacity and thus capacity planner gives up in this situation. A negative distance becomes 0 if $\texttt{objective} \neq \texttt{dontcare}$, and becomes positive otherwise.

Example 18 (Application of the Distance Formula).
This example demonstrates the proposed distance formula. In the case of a constraint $\texttt{cost} \in \mathcal{T}$: if

$$\texttt{requested\_value(cost)} = 100, \texttt{offered\_value(cost)} = 50, \texttt{objective(cost)} = \texttt{minimize}$$

then

$$\texttt{dist(cost)} = \frac{100 - 50}{\texttt{flex(cost)}} * -1 = \frac{-50}{\texttt{flex(cost)}} < 0$$

a negative distance that is converted to 0 because of a cheaper offer which is acceptable. □

The distance of an allocation $\texttt{alloc}_i$ is an aggregated distance of all its constraints $\mathcal{T}_i$ i.e.

$$\texttt{dist(alloc}_i\texttt{)} = \sum_{\texttt{t} \in \mathcal{T}_i} \texttt{dist(t)}$$

whereas the distance of a co-allocation $\mathcal{CL}_i$ is an aggregation of the distance of all allocations $\in \mathcal{CL}_i$, that is

$$\texttt{dist}(\mathcal{CL}_i) = \sum_{\texttt{alloc} \in \mathcal{CL}_i} \texttt{dist}(\texttt{alloc})$$

Example 19 (Ranking with Distance Formula).
In order to demonstrate ranking of generated offers with distance formula, consider a client that requests for a resource with required constraints as:

$$\{\{procs = 3, \texttt{flex}(procs) = 1\}, \{mem = 2GB, \texttt{flex}(mem) = 4\},$$

$$\{cost = 150, \texttt{flex}(cost) = 6\}\}$$

An allocator generates offers for the request with offered constraints as shown in Table 6.1. According to the distance formula, the offer B has the least distance (closest match) and thus has maximum utility. It is ranked as the best offer among the available set of offers $\{A, B, C\}$. □

Table 6.1. Ranking of offers using distances.

offer	procs	mem	cost	dist
A	4	5	180	$\frac{4-3}{1} + \frac{5-2}{4} + \frac{180-150}{6} = 6.8$
B	2	6	140	$\frac{2-3}{1} + \frac{6-2}{4} + \frac{140-150}{6} = \mathbf{3.9}$
C	6	1.5	220	$\frac{6-3}{1} + \frac{1.5-2}{4} + \frac{1.5-2}{6} = 15$

The *distance formula* given in Equation 6.1 is inappropriate for constraints with lexicographical values, typically used for information types such as *operating system*, *processor* etc. In order to deal with lexicographical values, we propose a different approach that uses a semantics-based hierarchical structure of the all possible values of a constraint `t`. If i and j refer to the levels of $\texttt{requested\_value}(\texttt{t}) = I$ and $\texttt{offered\_value}(\texttt{t}) = J$ respectively in the hierarchy and $k = \texttt{root}(i, j)$ is the level of their common root, then

$$\texttt{dist}(\texttt{t}, I, J) = \begin{cases} 0 & i \leq j \wedge i = k \\ \frac{2^{i+j-2\times k}}{\texttt{flex(t)}} & \texttt{otherwise} \end{cases} \tag{6.2}$$

This covers two special cases related to semantically defined classes I and J, for requested and offered values respectively:

- $I \sqsubseteq J \mid I \equiv J$: The ideal value belongs to a sub or equivalent concept of the offered value class, with $\texttt{dist}(\texttt{t}, I, J) = 0$ i.e. an ideal match.
- $I \sqsupseteq J$: The ideal value is a super concept of the offered value and thus it leads to a next satisfiable decision. This is explained in Example 20.

Example 20 (A Lexicographical Constraint (OS)).

```
1
2          OS              (Operating System)      Level 1
3         /  \
4      Unix   Windows      (Sub classes of OS)     Level 2
5     /   \
6  Linux   Solaris         (Specialized Classes)   Level 3
```

For instance, in the hierarchy of *operating systems* as shown above, the `Linux` to `Unix` distance

$$\texttt{dist}(OS, Linux, Unix) = \frac{1 * 2^1}{\texttt{flex}(OS)}, i = 3, j = 2, k = 2$$

and the *Linux* to *Windows* distance

$$\texttt{dist}(OS, Linux, Windows) = \frac{1 * 2^3}{\texttt{flex}(OS)}, i = 3, j = 2, k = 1$$

and for the same value of flexibility $\texttt{flex}(OS)$, it can be deduced that

$$\texttt{dist}(OS, Linux, Windows) > \texttt{dist}(OS, Linux, Unix)$$

That is, *Linux* is a close match with *Unix* than *Windows*. This makes constraints with lexicographical values, which can be described in a hierarchy of subsumption tree [155], comparable with the constraints having numerical values.

6.3 Negotiation Protocol

Negotiation between the requester and provider is a 3-layer (co-)allocation offer generation process as shown in Figure 6.2. Once initiated, the process continues until participants reach an agreement. The requester (e.g. scheduler) selects a best suitable offer or re-negotiates by changing some of the constraints. The protocol also introduces negotiation within system components to generate co-allocation offers with minimized requester-provider interactions while maximizing the resource utility. A co-allocator accepts requests and generates (co-)allocation offers in the form of an agreement document.

At the first layer, the *allocators* deal with reservations of individual Grid nodes $\in \mathcal{G}$. The main objective of this layer is to perform resource-level capacity planning and management. Offers are generated, albeit not necessarily optimal and/or conflict-free. The *co-allocators* at second layer takes the client's preferences into account and generates co-allocations for optimization of the Grid utility. Therefore, the second layer improves the quality of the generated offers in a broader sense.

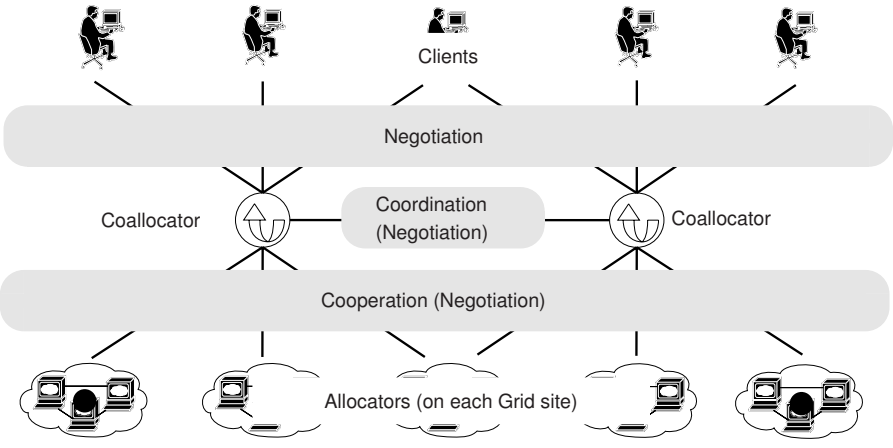


Fig. 6.2. Cooperative Negotiation Protocol Layers.

Intra-application contentions between allocations generated for activities of the same application, are eliminated. It is possible that an allocation made at the second layer is optimal and contention-free for a specific application, but, as resources are shared, the same option can be offered to multiple applications. This introduces the inter-application contentions that needs coordination among co-allocators for contentions elimination. Such contentions are propagated to and handled by the third layer, which can be activated by clients and also by the system when resources join or leave the Grid. The third layer also ensures provision of *open reservations* on-demand according to the agreement. Different algorithms have been introduced for resource allocation that is a *NP-Complete* problem.

6.3.1 Allocation Offer Generation

The allocation layer deals with the capacity planning of a single node in which each allocator works as a capacity planner and considers only local view of the resource capacity. Formally, an allocation request $\mathtt{q} \in \mathcal{CQ}_i$ is forwarded by a *co-allocator* to its underlying *allocators*, which then generate a set $\mathcal{L}_i$ of allocation offers and return $\mathcal{L}_i$ ordered by client utility. The resource utility may depend on the providers strategy. If k is the total number of offers, then:

$$\mathcal{L}_i = \{\mathtt{alloc}_{i,1}, ..., \mathtt{alloc}_{i,k}\} \; with \; \mathcal{U}(\mathtt{alloc}_{i,x}) > \mathcal{U}(\mathtt{alloc}_{i,y}), x > y\}$$

An allocation $\mathtt{alloc}_i$ is modelled as rectangle given by its width as capacity and height as duration. The start time of $\mathtt{alloc}_i$ is $\mathtt{startt}(\mathtt{alloc}_i)$, the requested execution time is $\mathtt{duration}(\mathtt{alloc}_i)$, and the deadline $\mathtt{endt}(\mathtt{alloc}_i)$ is

$$\mathtt{endt}(\mathtt{alloc}_i) \geq \mathtt{startt}(\mathtt{alloc}_i) + \mathtt{duration}(\mathtt{alloc}_i)$$

The capacity of an entire node is also modelled as a rectangle but with fixed width $\mathcal{P}$ and infinite height $\mathtt{T}$ (time horizon). The *allocator* tries to locate a set of available slots while fulfilling hard constraints and minimizing the distance of soft constraints. That is,

$$(\mathtt{startt}(\mathtt{alloc}_i) + \mathtt{duration}(\mathtt{alloc}_i) \leq \mathtt{startt}(\mathtt{alloc}_j) \vee$$

$$(\mathtt{startt}(\mathtt{alloc}_j) + \mathtt{duration}(\mathtt{alloc}_j) \leq \mathtt{startt}(\mathtt{alloc}_i)$$

whereas

$$\forall_{\mathtt{alloc}_j \in A_i} \mathtt{alloc}_j \neq \mathtt{alloc}_i \wedge \mathtt{dist}(\mathtt{alloc}_j) = \bot$$

The *allocation problem* shares similarities with *strip packing problem*. In the strip packing problem we try to place a set of two dimensional boxes into a vertical strip of a specific width and an infinite height while minimizing the total packed height of the strip. Translated to our allocation problem, the width of the strip corresponds to the resource capacity, for instance, number of processors $|\mathcal{P}|$, and the vertical dimension corresponds to time horizon $\mathtt{T}$. If the list of rectangles is unknown in advance, the strip packing problem is called an *on-line strip packing*, which is *NP-hard* [38], and exactly maps to our problem. On-line strip packing is addressed by different heuristics such as shelf algorithm [38].

The simplest on-line method is to check whether a newly requested allocation finds an immediate placement. If there is none, the request is rejected. This is a very simple but crude technique that needs to know only current view and shows a low resource utilization resulting in the wastage of the strip capacity. A sophisticated on-line method increases the acceptance ratio by planning, i.e. looking into the future according to the client's flexible condition for negotiation.

We introduce a new algorithm called *Vertical Split and Horizontal Shelf-Hanger (*VSHSH*, pronounced as* wish*)*, which provides a hybrid approach and fits better to our *resource allocation problem*. In the classic *shelf algorithm*, the strip is horizontally split into shelves and only bottom-left justified packing in a shelf is possible. In contrast to the shelf algorithm, the VSHSH allows top-right justified packing as well. In this way, the VSHSH keeps unused area of the strip minimum but significantly increases application utility (Algorithm 8). As stated earlier, the utility is derived from the distance between offered and requested values of a constraint. This also include the constraint *cost*: depending on the cost model (Section 6.3.4), an allocation far in future could be cheaper whereas an earliest possible allocation might be expensive.

As the goal is to provide allocations as close to the requested QoS as possible, we propose either top or bottom justification for an allocation depending on the distance from the requested timeframe. This approach increases the global utility (more satisfied QoS constraints) as probability of wider time-constraint distance is reduced.

Lemma 2. *The VSHSH - with allocations justified to either the top or the bottom of a shelf whichever has a shorter distance to the requested timeframe - increases Grid utility.*

Proof. Let $dist^{shelf}(alloc_i)$ *and* $dist^{vshsh}(alloc_i)$ *be the distance of an allocation* $alloc_i \in \mathcal{CL}_i$ *calculated with shelf and VSHSH algorithms respectively for an allocation request* q_i*, then the distance for start time with shelf algorithm is*

$$dist^{shelf}(startt(alloc_i)) = startt(q_i) - sbot(shlf)$$

whereas with the VSHSH algorithm is

$$dist^{vshsh}(startt(alloc_i)) = \min(startt(q_i) - sbot(shlf), stop(shlf) - endt(q_i))$$

where $sbot(shlf)$ *and* $stop(shlf)$ *are base and top of a shelf* $shlf$ *respectively.This is depicted in Figure 6.3 and clearly shows that*

$$dist^{vshsh}(startt(alloc_i)) \leq dist^{shelf}(startt(alloc_i))$$

A reduced distance of an allocation results in improved allocation utility that contributes towards improvement in the overall Grid utility $\mathcal{U}$*.* □

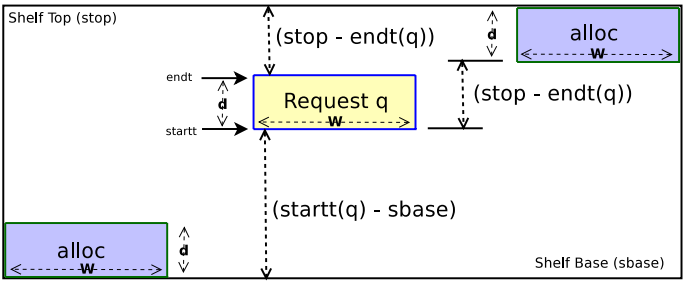


Fig. 6.3. VSHSH top or bottom justified allocations.

Furthermore, the VSHSH also vertically splits the strip into multiple sub-strips so that the width of all sub-strips equals the width of main strip. The formation of multiple sub-strips is used to protect an appropriate share of the resource capacity for the different communities and may apply different allocation strategies for each strip, such as different shelf height and cost models. In this way a sub-strip becomes an independent strip.

The pseudo code of the allocation offer generation algorithm (VSHSH), as explained above, is represented in Algorithm 8 that can be summarized as follows:

Algorithm 8 The Pseudo-Code of the Allocation Offer Generation Algorithm.

vshsh()
Input: Request q // *A request* $q \in \mathcal{Q}$ *with a set of constraints* $\in \mathcal{T}$.
Output: $\mathcal{L}$ // *A set of allocation offers*
shlf := A set of shelves in the strip to be allocated
shelf_height := Shelf height that is fixed for all shelves
strip_base := sbot(shlf_0) // *start time of the Strip*
$curr := \frac{(\text{startt(q)} - \text{strip\_base})}{\text{shelf\_height}}$ // *current shelf index*
capacity:= capacity (processors) requested in q
duration:= endt(req) − startt(q) // *duration requested*
$\mathcal{L} := \oslash$ // *Initially an empty offer set*
while $curr \leq (curr + \text{flex(endt(q))}$ **do**
  // *generate a bottom justified* offer *in current shelf* shlf_{curr}*, if possible.*
  $\mathcal{L}_{curr}$ // *Existing allocations belonging to* shlf_{curr}
  startt := strip_base + $curr \times$ shelf_height // *start time of* offer *to be generated*
  offer := create a template of an allocation
  offer := {capacity, duration, startt, startt + duration}
  // *Initialized offer with requested capacity, duration and possible available start and end times*
  if $\mathcal{L}_{curr} \cap \{\text{offer}\} = \oslash$ **then**
    // *allocation* offer *is possible in* shlf_{curr}
    $\mathcal{L} := \mathcal{L} + \{\text{offer}\}$;// *Add time-constrained generated offer in* $\mathcal{L}$
  else
    for all alloc $\in \mathcal{L}_{curr}$ **do**
      offer := {capacity, duration, startt(alloc), startt(alloc) + duration}
      if $\mathcal{L}_{curr} \cap \{\text{offer}\} = \oslash$ **then**
        // *allocation* offer *is possible on top of* alloc *in* shlf_{curr}
        $\mathcal{L} := \mathcal{L} + \{\text{offer}\}$;// *Add time-constrained generated offer in A*
      end if
    end for
  end if
  $curr := curr + 1$ // *Index of next shelf*
end while
$curr := \frac{(\text{startt(q)} - \text{strip\_base})}{\text{shelf\_height}}$
while $curr < (curr - \text{flex(startt(q))}$ **do**
  // *generate a top justified* offer *in shelf* shlf_{curr} *if possible.*
  endt := strip_base + $curr \times$ shelf_height + shelf_height // *endtime of the* offer *to be generated*
  offer := create a template of an allocation
  offer := {capacity, duration, endt − duration, endt} // *Initialize*
  if $\mathcal{L}_{curr} \cap \{\text{offer}\} = \oslash$ **then**
    // *top justified allocation* offer *is possible in* shlf_{curr}
    $\mathcal{L} := \mathcal{L} + \{\text{offer}\}$;// *Add time-constrained generated offer in* $\mathcal{L}$
  else
    for all alloc $\in \mathcal{L}_{curr}$ **do**
      offer := {capacity, duration, startt(alloc) − duration, startt(alloc)}
      if $\mathcal{L}_{curr} \cap \{\text{offer}\} = \oslash$ **then**
        // *allocation* offer *is possible below* alloc *of* shlf_{curr}
        $\mathcal{L} := \mathcal{L} + \{\text{offer}\}$ // *Add time-constrained generated offer in A*
      end if
    end for
  end if
  $curr := curr - 1$ // *Index of previous shelf*
end while
// *generate cost-constrained offers*
if flex(cost(q)) < 4 or user belongs to economy class **then**
  generate an offer after 2^{nd} week or in economy strip
else
  generate an offer in 1^{st} *week* or in expensive strip
end if
return $\mathcal{L}$;

- The VSHSH algorithm is a new solution for on-line strip packing problem, and applies to a strip of fixed width (number of CPUs as node capacity), and infinite height i.e. an infinite time during which a node is available for allocations;
- The strip is divided in an infinite number of *shelves* of equal height, and a request $\mathtt{q}$, as a rectangle of a fixed width (requested capacity) and a fixed height (timeframe: $\mathtt{endt}(\mathtt{q}) - \mathtt{startt}(\mathtt{q})$), is proposed to be packed in the strip, and multiple possible slots (options) in the strip, where $\mathtt{q}$ can be packed, are proposed as follows;
- For the rectangle $\mathtt{q}$, a matching slot is proposed in a shelf, closer to the requested timeframe, with free space. The packing in a shelf is always proposed either top justified or bottom justified depending on the utility (Section 6.2.2) of the proposed offer;
- Additional packing possibilities (offers as $\mathcal{L}$) for packing $\mathtt{q}$ are proposed based on the client's flexibility over the requested timeframe. Depending on the flexibility $\mathtt{flex}(\mathtt{startt}(\mathtt{q}))$ over the start time $\mathtt{startt}(\mathtt{q})$, a number of top-justified offers ($\leq \mathtt{flex}(\mathtt{startt}(\mathtt{q}))$) are generated latest possible before the $\mathtt{startt}(\mathtt{q})$, and depending on the flexibility $\mathtt{flex}(\mathtt{endt}(\mathtt{q}))$ over the end time, a number of bottom-justified offers ($\leq \mathtt{flex}(\mathtt{endt}(\mathtt{q}))$) are generated earliest possible after $\mathtt{endt}(\mathtt{q})$. Only one offer in a shelf can be generated at a time for a specific client;
- If a client is more flexible over the cost ($\mathtt{flex}(cost(\mathtt{q})) \geq 4$), then the VSHSH generates a relatively expensive offer with earliest possible start time after the current time, most likely in the first week, otherwise a relatively cheaper offer is generated after the second week in the strip. This is to give an attractive offer that is suitable for a specific class of users;
- All generated offers are collected in a list $\mathcal{L}$ and sent to the client for its acceptance.

Example 21 (Application of the VSHSH).
This example demonstrates the *VSHSH* algorithm as shown in Algorithm 8. Figure 6.4 depicts an example strip visualizing shelf $1, 24, 45$ with allocations (already reserved) in shaded-solid boxes. We assume that all other shelves are fully packed. In this situation, an allocation request arrives for which the VSHSH generates three allocation offers. As the request is closer to the top of the shelf 24, therefore the VSHSH generates a top justified time-constrained offer, i.e. *offer* 1. The second offer (*offer* 2) targets the economy class of clients and is generated with least cost but higher distance of other constraints. The third offer (*offer* 3) is more expensive but gives the earliest possible time. The allocator will offer these options and the co-allocator will choose the best offer closer to the client's overall requirements. □

VSHSH also proposes dynamic scaling of the underlying sub-strips with k shelves of variable widths and heights. In a variable shelf height version of VSHSH, a new shelf is created if:

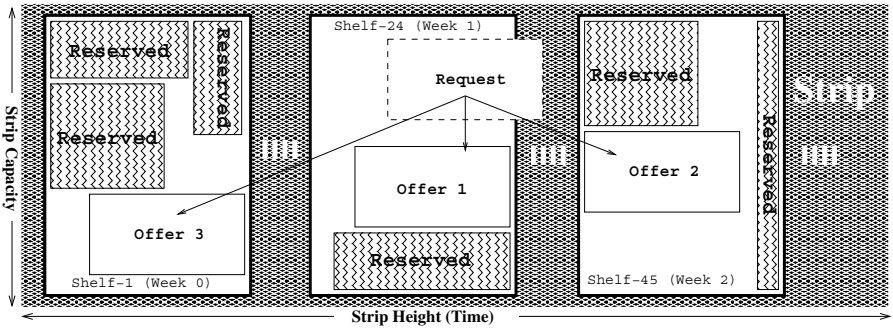


Fig. 6.4. Possible offers generated by the VSHSH applied to a strip with free space in three shelves (1,24 & 45). Offers are generated according to the client's economic preferences.

- an allocation does not fit in any existing shelf found within the client's flexible range.

  $$\texttt{sbot}(\texttt{shlf}_i) \leq \texttt{startt}(\texttt{alloc}) \leq \texttt{sbot}(\texttt{shlf}_j) \ \vee \ \exists \mathcal{L}_i \in \mathcal{L}(\mathcal{L}_i \cap \{a\} = \{a\})$$

  In this case a new shelf $\texttt{shlf}_{k+1}$ is created, such that

  $$\texttt{sbot}(\texttt{shlf}_{k+1}) = \max(\texttt{stop}(\texttt{shlf}_k), \texttt{startt}(\texttt{alloc}))$$

- an existing shelf can be split into two shelves so that the new allocation can be bottom-justified in upper shelf and there is no overlap with lower shelf.

Furthermore, *VSHSH* also proposes the notion of borrowing space from the adjacent shelves. For instance, if an allocation requests longer duration that fits in the multiple adjacent shelves, then it may be honored at a higher price.

The *VSHSH* generates alternative offers according to different constraints such as timeframe, cost, capacity, and client's flexibility set for these constraints. In case of client's flexibility over height (duration) and width (capacity), we propose to change the area of the rectangle according to the *isospeed scalability* [156] i.e.

$$\psi(p, p^{'}) = \frac{Time^{'}.P}{Time.P^{'}}$$

of the system for the requested application component (activity in case of workflow). For example, if the requested capacity is not available, then an offer with reduced capacity but increased duration might be acceptable.

The contentions among multiple allocations are propagated to the co-allocation layer instead of handling it locally by the allocators with a limited view of overall Grid capacity. This is due to the possibility that higher local objective may lower the global utility.

6.3.2 Co-allocation Offer Generation

The *co-allocation* (Algorithm 9) is responsible for the generation of a set of possible co-allocation offers. Formally, a co-allocator receives a co-allocation request

$$\mathcal{CQ}_i = \{\mathtt{q}_1, ..., \mathtt{q}_k\}, k \in \mathbb{N}$$

and instantiates a *co-allocation manager* which then sends each allocation request $\mathtt{q}_i \in \mathcal{CQ}_i$ to k allocators of selected nodes and receives a set of allocation offers $\mathcal{L}_i$, where

$$\mathcal{L}_i = \bigcup_{j=1}^{k} (\mathcal{L}_{i,j}) : \mathcal{L}_{i,j} = \{\mathtt{alloc}_{i,1}, ..., \mathtt{alloc}_{i,k_j}\}, k_j \in \mathbb{N}$$

where k_j may be different for each $\mathcal{L}_i$. Now the CM generates a set of co-allocation offers

$$\mathcal{CL}_i = \{\mathcal{CL}_{i,1}, ..., \mathcal{CL}_{i,n_i}\}$$

with function $\Psi : \mathcal{L}_i \mapsto \mathcal{CL}_i$ such that,

$$\mathcal{CL}_{i,j} = \Psi(\mathcal{L}_{i,1}, ..., \mathcal{L}_{i,k_j}) = \{\mathtt{alloc}_1, ..., \mathtt{alloc}_{k_j}\} \mid \forall_{i \in k_j}(\mathtt{alloc}_i \in \mathcal{L}_{i,j})$$

Here Ψ is a set reduction operator whose domain is a set of all possible sets of allocation offers received for each request $\mathtt{q}_j \in \mathcal{CQ}_i$ from the different allocators. We frame a co-allocation as an optimization problem, that is similar to CSP, and propose a new algorithm which is a modified form of an existing min-conflict local search algorithm [152]. In contrast to min-conflict local search, the new algorithm allows to change the resource objective depending on the number of conflicts, so that, a better compromise could be found with optimal global utility. The co-allocator logically considers each allocation offer as a resource and applies the new solution to these 'resources' in order to make a set of acceptable combinations. This is in contrast to the first layer where resources are actual physical computers. This significantly reduces the complexity of the problem, i.e. from $2^{|\mathcal{P}^g|+\mathtt{T}}$ to $2^{|\mathcal{L}_i|}$.

A *co-allocation manager (CM)* returns to its client a set of co-allocation offers $\mathcal{CL}_i$ ordered by its utility. The client then filters again, by eliminating the offers which are not acceptable and sends back in a preferred order for confirmation. The confirmation process is two phase committable. In the first phase, the CM tentatively reserves each allocation offer $\mathtt{alloc} \in \mathcal{CL}_i$, and in the second phase, it confirms each of the accepted offers provided there is no contention. Otherwise, it propagates the contentions to the third layer.

The pseudo code of the co-allocation offer generation, as explained above, is represented in Algorithm 9 that can be summarized as follows:

- The co-allocation offer generation algorithm addresses a constraint satisfaction problem. It receives a co-allocation request $\mathcal{CQ}$ and generates a set of co-allocation offers $\mathcal{CL}$;

Algorithm 9 The Pseudo code of the Co-Allocation Offer Generation Algorithm.

coallocation()
Input: $\mathcal{CQ} = \{\mathtt{q}_1, ..., \mathtt{q}_n\}$ // *co-allocation request*
Output: $\mathcal{CL}$ // *a set of co-allocation offers*
$\mathcal{B}$ // *A set of candidates of nodes* $\in \mathcal{G}$ *generated for each* $\mathtt{q} \in \mathcal{CQ}$
$\mathcal{L}$ // *a set of allocation offers to be received from* $\mathcal{B}$
for all $\mathtt{q}_i \in \mathcal{CQ}$ **do**
  $\mathcal{B} :=$ `generate candidates`$(\mathtt{q}_i)$ // *Candidates* $\mathcal{B} \subseteq \mathcal{G}$
  // *negotiate with each candidate (allocator) by calling VSHSH*
  for all $\mathtt{b} \in \mathcal{B}$ **do**
    $\mathcal{L}_i := \mathcal{L}_i \cup$ `contact` $\mathtt{b} \in \mathcal{G}$ `and negotiate for allocation vshsh`$(\mathtt{q}_i)$
  end for
  $\mathcal{L} := \mathcal{L} \cup \mathcal{L}_i$ // $\mathcal{L}$ *becomes* $\{\mathcal{L}_1, ..., \mathcal{L}_n\}$
end for
while $n \leq |\mathcal{L}|$ **do**
  $\mathcal{CL}_j :=$ `create` a `co-allocation` template
  $\mathcal{CL}_j := \{\mathtt{alloc}_1, ..., \mathtt{alloc}_n\}$ // *A combination of allocations* $\mathcal{L} \mapsto \mathcal{CL}$
  if $\forall_{i \in n}(\mathtt{alloc}_i \in \mathcal{L}_i) \bigwedge \mathcal{U}(\mathcal{CL}_j) > 0 \bigwedge \bigcap_{\mathtt{alloc} \in \mathcal{CL}_j} \mathtt{alloc} = \varnothing$ **then**
    // *Each* $\mathtt{q} \in \mathcal{CQ}$ *has a* $\mathtt{alloc} \in \mathcal{L}$*, utility of* $\mathcal{CL}_j$ *is non-zero, and no resource and time contentions*
    $\mathcal{CL} := \mathcal{CL} + \{\mathcal{CL}_j\}$ // *add a co-allocation offer in offer set* $\mathcal{CL}$
    for all $\mathcal{L}_i \in \mathcal{L}$ **do**
      $\mathcal{L}_i := \mathcal{L}_i - \{A_i \cap \mathcal{CL}_j\}$ // `exclude` *from* $\mathcal{L}$ *the allocation offers that are already* `consumed`
    end for
  end if
end while
return $\mathcal{CL}$;

- A co-allocation request consists of a set of allocation requests, and for each allocation request $\mathtt{q}_i \in \mathcal{CQ}$, a set of candidate nodes is generated;
- Each cadidate for $\mathtt{q}_i$ is negotiated for allocation offers, and a set of allocation offers $\mathcal{L}_i$ for $\mathtt{q}_i$ is received from all candidates;
- After the phase of negotiation with all candidates, there is a set $\mathcal{L}_i$ of allocation offers for each $\mathtt{q}_i \in \mathcal{CQ}$;
- A set of co-allocation offers is generated in such a way that each co-allocation offer contains a set of allocation offers in such a way that for $\mathtt{q}_i \in \mathcal{CQ}$ there is an allocation offer $\in \mathcal{L}_i$ received for $\mathtt{q}_i$ from one of its candidates;
- The order of start time of allocation offers in a co-allocation offer is preserved according to the order of start times of allocation requests $\in \mathcal{CQ}$;
- Finally, all generated co-allocation offers are collected in a list $\mathcal{CL}$ and sent to the client for acceptance.

6.3.3 Contention Elimination

In a shared environment such as the Grid where clients compete for the scarce resources, the contentions are unavoidable. It is likely that same slots can be offered to multiple clients simultaneously that may lead to reduced QoS and even unacceptable solutions. The coordination layer deals with such situations and produces contention-free solutions by eliminating either conflicting offers from the solution domain or by lowering the objective level of some of the underlying allocators. The contention are raised by the allocator. The notified co-allocator then mediates cooperative negotiation among the contentious CMs.

Solution Generation

First, the notified co-allocator collects all information needed to generate alternative solutions, including allocation options offered to the conflicting applications, and then enters in the solution generation process. The solution generation process starts with ordering the solution domain according to application utility. A solution for the highly constrained application is generated first, assuming that this will reduce the possibility of further contentions. In order to accommodate all requests, the mediator may lower the objective levels of the underlying allocators, depending on the number of contentions associated with each offer. The process terminates: 1) if all suitable contention-free co-allocations are generated, 2) or the objective of any of the allocators cannot be reduced further.

Solution Evaluation

Second, a mediator enters in the solution evaluation phase by sending each of the CMs a set of contention-free solutions. The CM then filters out some of the generated solutions and orders the rest of them from best to worst, based on the application's overall utility. Once a mediator has the ordering from the CMs, it generates an overall solution by choosing the highest ranked alternatives from each of the CMs which lead to a consistent solution.

Solution Implementation

Finally, the assigned solutions are sent back to each of the CMs, which then implement the final solution by confirming the reservations and notifying the client.

6.3.4 Cost Model

In order to deal with cost-centric optimization, we introduce a concept of fictitious money (Grid$) and provide a configurable cost model for different

strips or different shelves of a strip. The client's account for fictitious money can be maintained by a co-allocator, and the underlying allocators can charge the client's account. This is in contrast to the open resource allocation model applied to *mobile code* [175], in which a fictitious money is maintained based on the *lottery scheduling* [185]. We support three categories of clients i.e. economy, moderate, and privileged to be derived from the `flex(cost)`. We are considering different options for recharging the fictitious money account by associating it with the lifetime of the Grid proxy that is used to access the resources, and/or distributing fictitious money earned by a resource among its users according to their category.

6.4 Experiments and Evaluation

The proposed system is implemented in GT4 [10] as part of *GridARM* that is an integral component of Askalon [62], and deployed on the Austrian Grid [33] for evaluation.

The *VSHSH* algorithm is evaluated against a set of existing on-line strip packing algorithms [18] that includes *First Fit (FF)*, *Best Fit (BF)*, and *Next Fit (NF)*. The experiments were performed in order to compare the allocation (at node-level) as well as the co-allocation (at Grid-level) algorithms. The start time of each allocation request varied from 1-min to 14-days. A set of requests was randomly generated and then consistently used for all tests. The duration of each allocation request was also randomly selected in such a way that 80% of the requests were smaller than 4 hours, and 20% were between $4 - 36$ hours. These values were chosen based on our experience running real Grid workflow applications as described in [63]. For co-allocations, we have generated requests following the structure of the workflow application composed of 4 sequential regions with 2 parallel regions. The values corresponding to the parallel region were randomly selected as a multiple of 6 minutes depending on the problem size.

A *first-fit, next-fit*, and *best-fit* are different approaches of legacy shelf algorithm that are used to decide which shelf an allocation rectangle should be put on once all appropriate shelves have been determined. For the next-fit approach, a rectangle is put onto the next available shelf on which it will fit. Next-fit does not allow back tracking. For the first-fit approach, each rectangle is put onto the lowest shelf that it will fit on. The *best-fit* approach is modified version of first-fit in which instead of selecting lowest shelf, a shelf with lowest allocations is selected.

For the measurements, we used strip and cost model as shown in Figure 6.5, where the cost varies between strips as well as between shelves. We have split main strip into three strips with 50%, 25% and 25% of the total capacity respectively.

Figure 6.6 demonstrates comparison of VSHSH with the first-fit, next-fit and best-fit approaches of the legacy shelf algorithm. It can be observed that

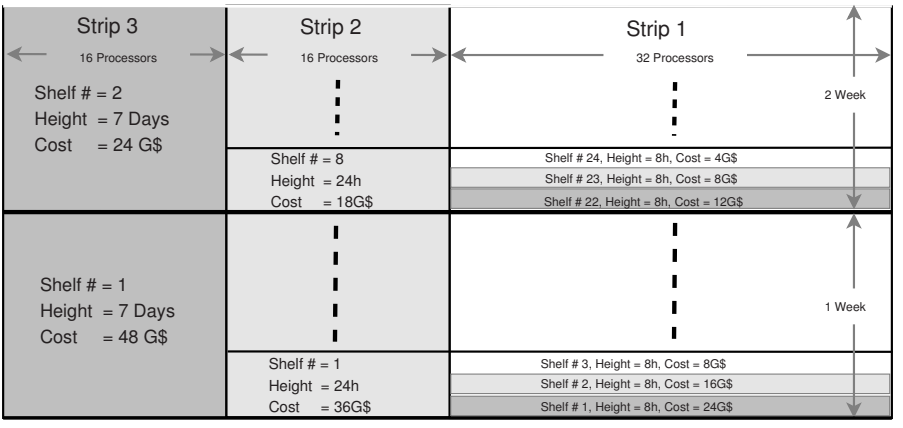


Fig. 6.5. A possible Capacity Management model for VSHSH with three Sub-Strips having different Capacity, Shelf Height, and Cost Models.

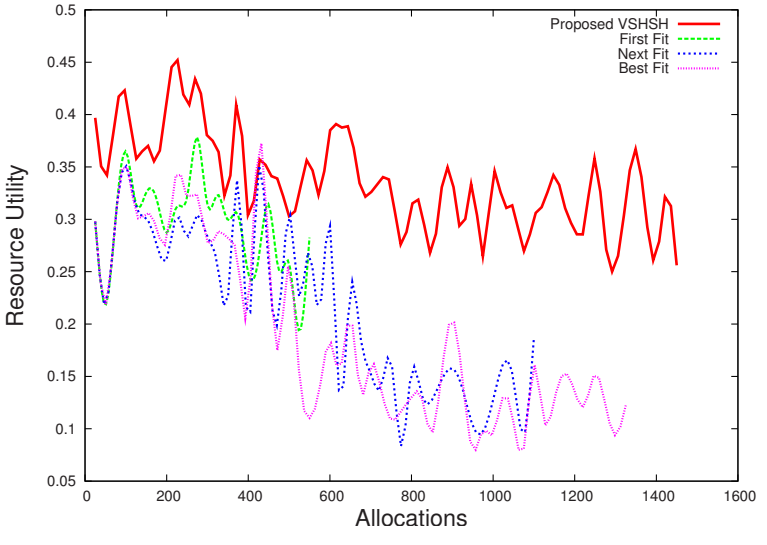


Fig. 6.6. Average Resource Utility with Allocations.

Table 6.2. Average response time per transaction.

Concurrent Users	2	4	6	8
Response Time (ms)	96	124	148	168

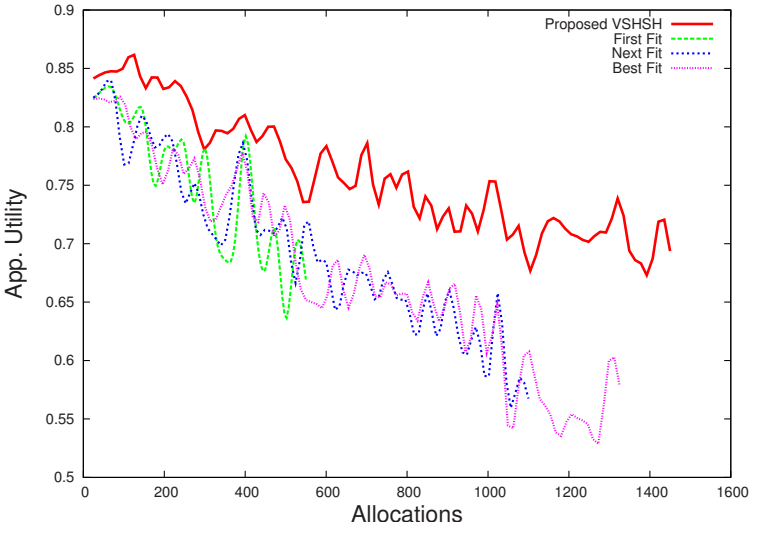


Fig. 6.7. Average Application Utility with Allocations.

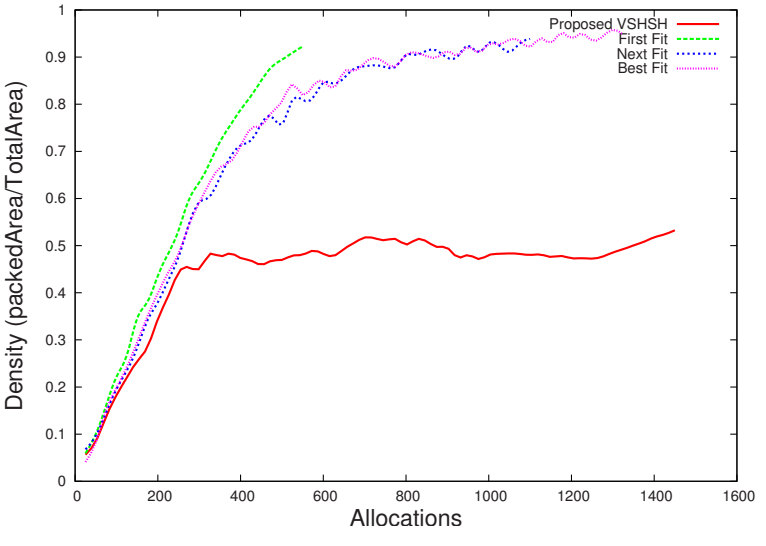


Fig. 6.8. Resource utilization (density).

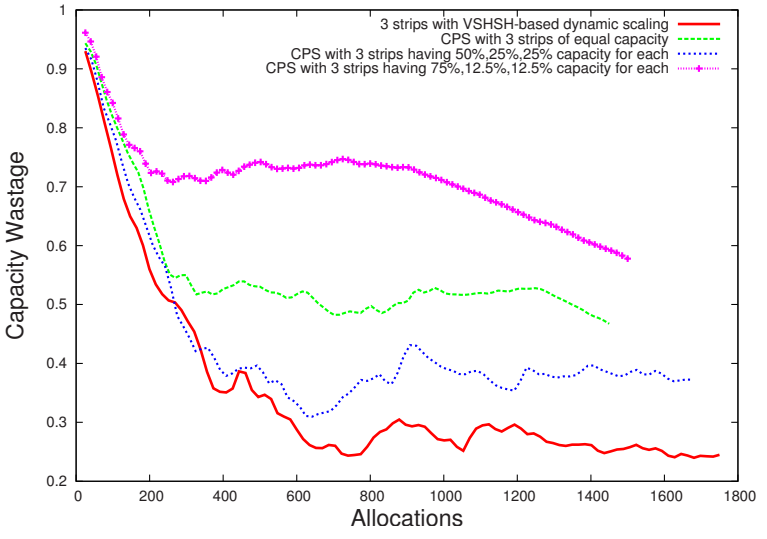


Fig. 6.9. The Resource Wastage with Different Capacity Planning Strategies (CPS).

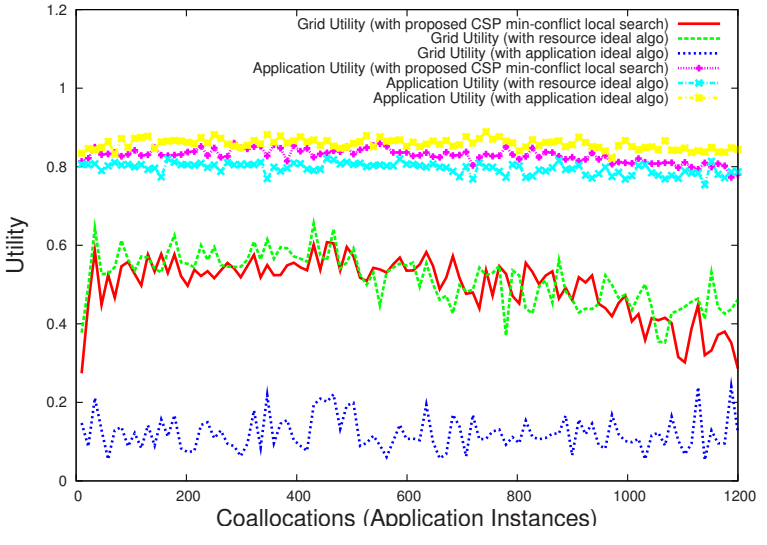


Fig. 6.10. Comparison of Co-Allocation Algorithm with Resource-Ideal and Application-Ideal Allocation Algorithms Showing Expected and Worse Performance.

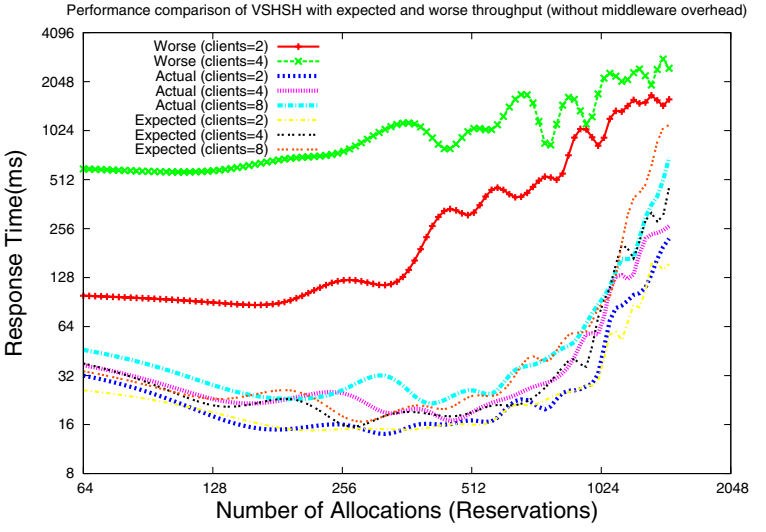


Fig. 6.11. Performance (Response Time per Transaction) of the VSHSH Compared with Expected and Worse Possible with Different Concurrent Clients but Without Grid Middleware Overhead.

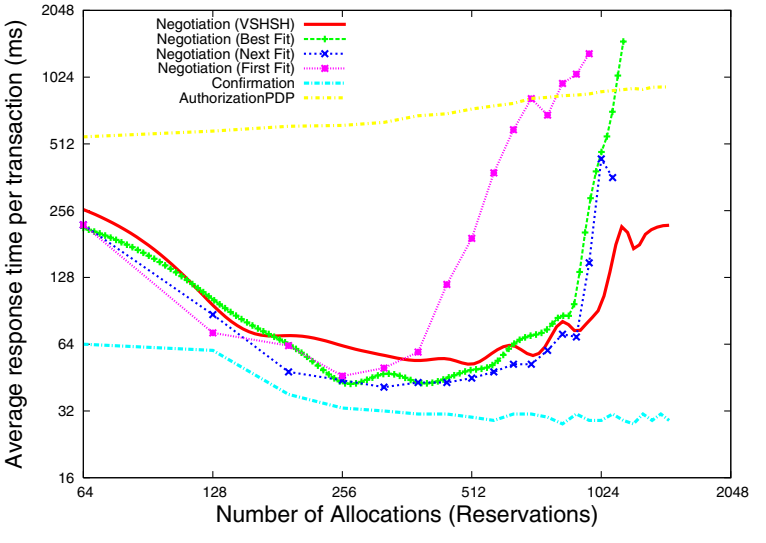


Fig. 6.12. Response time with different allocations.

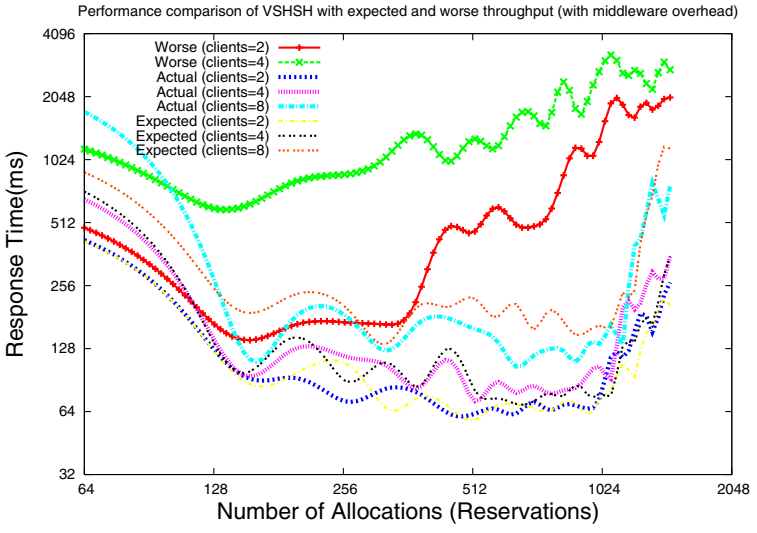


Fig. 6.13. Performance of the VSHSH along with middleware overhead compared with expected and worst possible with different concurrent clients along with Grid middleware overhead.

VSHSH maximizes the resource utility. Best-fit is slightly better than other legacy approaches but still VSHSH is much better. Interestingly, the behavior of VSHSH is similar to the Best Fit, but with higher utility. In addition, it can be observed that resource utility scales well, independently of the number of allocations.

Figure 6.7 compares application utility by comparing VSHSH with the legacy shelf algorithm approaches. It can be observed that VSHSH maximizes the utility. The main reason that VSHSH demonstrates better performance than other approaches is that VSHSH makes both top and bottom justified allocations possible.

Figure 6.8 depicts average density of allocated area, packed with varying number of allocations. The VSHSH shows lower density due to the capacity planning strategy which achieves better application and resource utility. We can observe that VSHSH maintains a density of 50% between 200 and 1300 allocations. In addition, VSHSH continues accepting more requests but the density grows linearly after 1300 requests.

Figure 6.9 compares average capacity wastage (unused total area) with different capacity planning strategies. We have set different capacity values to the strips in order to see how wastage can be reduced. It can be observed that setting equal capacity to each strip is worse than our base settings (50%,25%,25%) used for the previous measurements, and that increasing the first strip capacity (75%,12.5%,12.5%) does not reduces the wastage. This is due to the multiple QoS constraints which effects the trade-off between

resource utility and application utility. The strategy with dynamically scalable strips shows minimum resource wastage but it depends on the history of resource allocation requests.

Figure 6.10 depicts comparison of the proposed co-allocation offer generation algorithm that optimizes overall Grid and application utilities. The measurements were performed using 4 allocator instances. We compare the proposed algorithm with two other algorithms: (a) a *Grid-ideal* that maximizes Grid utility (i.e. the aggregated resource utilities), and (b) an *application-ideal* that maximizes application utility. We can observe that the application ideal algorithm reduces the resource utility significantly, whereas the resource ideal introduces small reduction to the application utility. In the case of our approach, the results are quite encouraging: the Grid utility is closer to the Grid-ideal, and application utility is closer to the application-ideal algorithm.

According to Figure 6.12 an average response time for a single *negotiate-reserve* session with varying number of allocations is initially similar for all algorithms. However, afterward in contrast to existing algorithms, the VSHSH response time increases linearly instead of exponentially with allocations.

Figure 6.11 and 6.13 show the response time of the VSHSH compared with expected and worst possible performance with different number of concurrent clients, i.e. 2, 4, and 8, with and without Grid middleware overhead, respectively. The relative values of worse performance are taken with the first two weeks are fully packed (reserved), and approximately 23000+ allocations were already made. The values for the expected performance correspond to the maximum utility, i.e. minimum response time required to perform the allocation, and to send the result to the client. The actual value corresponds to the response time of our VSHSH algorithm. We can observe that the performance of VSHSH is close to the expected performance, and that response-time grows linearly after approximately 1024 allocations. This is because of the increase in number of data structures (agreement documents) needed to store the allocations instances. Furthermore, the Grid middleware overhead as shown in Figure 6.13 is much higher due to communication, WSRF, and container overheads. In addition, there is an observable startup overhead (until 128 reservations), because of the initialization of the middleware and the data objects. It is obvious from these depictions that the main overhead lies in the Grid middleware rather than in the proposed system.

We have developed a graphical application tool as part of GridARM console, that can be used to specify multiple terms and conditions (constraints), visualize generated allocations and allocation offers and utility of the generated offers (explained in next chapter) that shows comparison of offers. Figure 6.14 shows reservations and allocation offers made in the entire Grid as well as made for a single node. It is depicted that allocations are made in a sequence and segmentation of unused slots is minimized that results in improved utilization. Lower part shows a dialog that is used to specify or browse multiple terms and conditions for allocation requests or received offers.

Fig. 6.14. Visualization of reservations (red) and allocation offers (green) for a single nodes as well as for the entire Grid. Lower part shows a dialog to compose multiple terms and conditions.

6.5 Related Work

Capacity planning has been addressed in several fields, for instance *yield management* in airline reservation system, where perishable resources are advertised and sold in a way to maximize overall profit [123]. However, capacity planning is rather new for the Grid. Negotiation-based advance reservation for optimized QoS delivery has been a subject of numerous studies [92, 189, 186, 123]. A few researchers are investigating the negotiation based advance reservation for the Grid.

GARA [73] and *DUROC* [54] defined a basic architecture and simple API for the manipulation of advance reservation of Grid resources. However, they concentrate mainly on its applicability for job management. On the other hand, *SNAP* [40] proposes a negotiation protocol for a distributed model in which resource interactions are mapped onto a set of Service Level Agreements (SLAs). *SNAP* has been replaced by the OGF proposed standard [87] on which we based our 3-layered negotiation protocol for agreement and capacity management.

The usefulness of advance reservation is theoretically presented in [115]. Modelling and solving resource allocation problems with soft constraint techniques is discussed in [199]. Performance impact of advance reservations on workflows is depicted in [159], and effects of different models to support advance reservations are presented in [162]. However, none of them has discussed capacity planning. An algorithm described in [142] supports fuzziness in a limited set of parameters and applies speedup models to propose flexible allocations. Co-reservations are proposed as virtual resources. However, dynamic Grid behavior, reservation of logical resources, and optimization from the perspective of the Grid had not been considered.

ICENI [115] uses a two tier system for reserving resources on the Grid. It exposes reservation capability of underlying resource managers such as SGE [168] without considering dynamic behavior of the Grid. No flexible negotiation mechanism that considers multiple QoS parameters other than the timeframe is available.

Gridbus [27] provides a broker service for the Grid resources that takes into account the fact that deadline and budget are specified, and then optimizes the usage of resources only by considering the current state of the resources but without any planning horizon. Buyya et. al. proposed in [167] a time optimization algorithm in auction-based proportional share systems with multiple VOs, in which a user broker periodically adjusts a bidding price in order to meet the deadline and minimize the cost.

A resource co-allocation across multiple nodes is presented in [24], they studied co-allocation in multicluster systems with both analytic means and with simulations for a wide range of parameters based on their previous work on influences of various parameters, such the job structure and size.

KOALA [117] Grid scheduler supports co-allocation with a limited planning to synchronize start-times of tasks to be co-allocated on multiple nodes. It focuses only on the make span of an application.

Many heuristics [48, 179] have been proposed for allocation optimization but they optimize a single objective, for instance minimizing execution time. Several heuristics have been proposed to address allocation problems based on multiple QoS constraints such as budget and deadline. The approaches given in [148] adjust a schedule generated by a time optimized heuristic and a cost optimized heuristic to meet users' budget constraints respectively. GRIA (Grid Resources for Industrial Applications) [82] provides various resource allocation strategies for workflow execution based on QoS requirements.

In [196], authors have developed algorithms based on the genetic algorithms to minimize either execution cost or time. In their extended work in [198] they have proposed a workflow execution planning approach, which optimize two objectives. The planner can generate a set of alternative options if the optimization objectives are conflicted. Alternative options provides more flexibility to users to choose a desired option based on their QoS requirements. It applies Multi-objective evolutionary algorithms (MOEAs) for the workflow execution planning problem. The goal is to simultaneously minimize two

conflicting objectives-*execution time* and execution cost while meeting users' maximum time constraint (deadline) and price constraint (budget).

Different reservation-aware schedulers such as [11, 168] do not consider flexibility of different constraints, thus resulting in lack of negotiability. Maui/Silver [111] is a job scheduler for cluster/Grid systems that makes it possible to allocate local resources in the future. On a Grid node, it can be used in combination with the other LRMs. However, an extention in the Grid job submission service is required. Also, it does not support negotiation and capacity planning.

In contrast to these existing works, we focus on optimizing multi-constrained allocations with effective capacity planning and management. A 3-layer negotiation-based advance reservation is supported in order to make flexible allocations. Generalized and practical solutions for priority provision and agreement enforcement are introduced to deal with the dynamic Grid behavior.

6.6 Summary

The Grid possesses a great potential for capacity planning and management along with advance reservation. In this paper, we have addressed several challenges and introduced Grid capacity planning with negotiation-based advance reservation for optimized QoS. Advance reservation enables a Grid resource manager to deliver resources on-demand with significantly improved QoS. A new 3-layer negotiation protocol with a smart offer generation mechanism makes it possible to optimally generate multi-constrained allocations and to efficiently reach an agreement. For an effective capacity planning, we model resource allocation problem as on-line strip packing problem and introduce a new algorithm to solve it. The new algorithm called VSHSH significantly improves Grid resource utilization. Contention elimination and open reservations are supported to deal with dynamic Grid behavior, whereas, a practical solution for agreement enforcement is provided based on the state-of-the-art Grid technologies.

A prototype of the proposed system has been implemented to examine the effectiveness of our approach. We have demonstrated that with proper capacity planning using negotiation-based advance reservation, the utilization can be maximized independently of the number of requested allocations. In addition, the proposed approach better deals with the dynamic nature of the Grid and generates more optimal allocations compared to existing heuristics used for NP-hard resource allocation problems. Furthermore, it is observed that the proposed approach does not add any significant overhead to the existing Grid middleware services.

7

Semantics in the Grid: Towards Ontology-Based Resource Provisioning

Automatic Grid resource discovery and brokerage shields the Grid middleware complexities from the Grid users and leads towards an invisible, easy to use, and robust Grid runtime environment. Realizing this vision requires a machine understandable resource description and intelligent resource matching mechanisms. Semantic technologies like ontologies provide vocabularies with explicit definition, unambiguous machine-interpretable meanings which make automatic resource brokerage possible. We propose an ontology-based resource description, discovery, and correlation mechanism. For the resource description model, it is proposed to replace the classical symmetric attribute based resource description model with an extensible asymmetric resource description model. This model provides a foundation for a flexible and extensible resource discovery and resource matching mechanism.

7.1 Introduction

With the emergence of the Grid and increase in its resources, resource management with automatic brokerage gains importance. Automatic Grid resource brokerage is a challenging task; it has to provide a mechanism in which the Grid resources can be advertised by resource providers, automatically discovered by a resource manager, and allocated to the resource requesters on-demand. The discovery and allocation of resources is part of resource management (Chapter 3). In addition, the resource management also provide negotiation mechanism between resource provider and requester (Chapter 5).

Different types of resources, including physical resources such as *nodes* (Definition 11), and logical resources such as *activities* (Definition 6), can provide similar capabilities but with varying degrees of quality of service. This highlights that resource capabilities are required to be presented in such a way that consumers can easily discover resources matching their requirements, by following sophisticated patterns of resource discovery, matchmaking, and

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 157–177, 2010.

negotiation. A powerful discovery mechanism can be built on expressive description mechanisms. This means, it is necessary to explicitly, precisely, and unambiguously describe Grid resources and specify various constraints over resource descriptions. The description should be automatically interpretable and understandable by services (Definition 9) operating as part of the *Grid runtime environment* (Section 2.4).

Attribute based resource description (Section 3.3.1), used mostly in state-of-the-art Grid middlewares, such as UniCore [70] and Condor [128], has several short comings: they lack power of representations, capability to dynamically update, and independence between the resource provider and resource consumer for expressiveness. This resource description mechanism is *symmetric*, i.e. both resource provider and consumer have to agree on a certain syntax or schema.

This chapter proposes an ontology-based resource description and matching mechanism. The *resource matching*, also refers to as matchmaking, is a process of creating associations between resource requests, the available resources, and their characteristics at a specific point in time. Resource matching is a central part, however we use it as a more general concept. An ontology provides vocabularies with explicitly defined and machine understandable meanings. We propose Grid resource ontologies in the form of OWL-DL classes and concepts for describing resources in such a way that they can be unambiguously interpreted and automatically understood by the management system. Our discovery mechanism is based on a simple but expressive request-response mechanism in which clients express requests based on OWL Query Language SPARQL (Section 2.5.3) or (OWL-QL) [66].

Our use of the Semantic Web technologies, such as (OWL) [180], SPARQL [182] and Jena APIs [32] for semantics based resource management, is in the line of the Semantic Grid vision [47, 43, 46] and its service-oriented architecture [36].

Resource ontologies are asymmetrically extensible so that resource providers can easily extend them without loosing the semantic soundness. We do not have to agree on a certain terminology while extending the ontological concepts. For instance, a term `Unix` defined as a class of `OS` in our base ontology of computing resources, then one can asymmetrically extend the term `Unix` to `Linux` and `Linux` to `ScientificLinux`. After this, a reasoner can automatically and unambiguously infer that `ScientificLinux` is a specialization of `Linux` as well as `Unix`.

7.2 Describing Resources with Semantics

The popular approach being used in the *semantic web* is the use of description logic with ontologies (Section 2.5.2). A commonly used definition of ontologies is that they are formal explicit specifications of a shared conceptualization

[90]. An ontology is a set of hierarchical description of important concepts in a domain, along with descriptions of properties of each concept.

In OWL (Section 2.5.2), an ontology is a set of definitions of classes and properties and the constraints on the way those classes and properties are employed. Using the powerful expressiveness and fact stating ability of OWL, one can declare classes as taxonomy of concepts and organize them in a subsumption hierarchy; classes can be expressed as a logical combination of other classes. Properties can also be organized in a sub-property hierarchy. Applying different kinds of *restrictions* on classes and properties leads towards introduction of specialized classes of new concepts.

In the context of the *Semantic Web* [181], ontologies play an important role in automatizing processes to access information. For this, ontologies provide structured and extensible vocabularies that demonstrate the relationships between different terms allowing intelligent agents to flexibly and unambiguously interpret their semantics. For example, a computing resource ontology might include the information that the terms `Pentium` and `Celeron` are `Intel` Processors, that `Intel` is not `AMD` or `SPARC`, and that an `Intel` system includes a `Pentium` or `Celeron` processor. This information allows the term `"Computer with Pentium or Celeron Processor"` to be unambiguously interpreted (e.g. by a resource broker) as a specialization of `"Intel System"`. In the Description Logics, the fundamental reasoning of `concept expression` is a subsumption [109], which checks whether one concept is a subset (or superset) of an other concept.

The matching of a request to the available resources depends on their proper and accurate description. Different languages, based on different logical formalism, can be used for resource descriptions. One can use *Description Logics* [16], *Logical Programming*, and *First Order Logic* as a logical formalism. Since in our case we need to propose alternative option as well, that is possible with hierarchical relationships between classes of resource and request descriptions, we have considered a *Description Logics* based language, i.e. OWL-DL. It provides a rich set of modeling primitives with powerful expressiveness for our requirements. By using the OWL-DL we attain the following:

- Grid resource management and brokerage takes a large step towards compatibility within the *Semantic Web* and the Grid resource descriptions become web-understandable.
- The resource provider can describe resources with different levels of complexity and completeness by extending existing concepts.
- The XML-schema data types and structures can be exploited for the property ranges in resource descriptions.
- A subsumption relationship enables us to perform complex resource matching, that otherwise could be very difficult if not impossible.

- A more natural conceptual definition of resources based on the restriction over the resource attributes is possible and a semantics based agreement between resource provider and consumer can be achieved.
- After having the conceptual resource descriptions, the resource provisioner can categorize sets of resources and generate alternative options for requesters.
- Most promisingly, clients can express complex requests in a simple-human as well as machine-understandable format.
- The management system can fulfill more resource requests or improve provisioning capability. For instance, in a legacy system without ontology-based description, a request for a *Unix* system fails if the term *Unix* is not specified. However, this might be successful while using an ontology.
- Spelling or typing errors in descriptions and requests can be prevented by using a controlled vocabulary.

The OWL-DL supports a reasonable set of tools for creation and manipulation of ontologies such as Protege ontology editor [177], Jena ontology APIs [32] along with different implementations of reasoners, such as the RACER [178] and Pellet [160].

7.2.1 Concept Description

We use OWL-DL for the description of resources and their components. However, the syntax of OWL is rather verbose and complex. In order to hide the complexity and simplify the expressibility, we introduce a very simple concept description language, that is automatically translated into OWL concepts and knowledge base. The purpose of introducing a very simple yet a new language is that it enables us to update dynamic information at runtime. For instance, in case of *node* (Definition 11), the *totalCPUs* is a static attribute whereas *freeCPUs* is dynamic. A resource provider can hardcode *totalCPUs* while describing resources but *freeCPUs* needs to be updated periodically. Different *information providers (IP)* can be configured as part of concepts and invoked dynamically to populate properties of a concept. The simplified concept description language can be defined as:

```
concepts name : [Node|Application|Goal] {
  concept name [: baseType+]? [as baseProperty] {
    [property name [: type+]? = value]*
    [meta property name [: type+]? value]*
    [constraint [constraint]* op [bounds]?]*
    [concept]*
  }
}
```

This shows that one concept can be defined by other concepts. A concept can be derived from a set of other concepts. Enclosing concepts possess *has-a* or

part-of relationships. Similarly, a property can also be inherited and multiple properties can be added to a specific concept. A standard XML schema type can be used as a range of a property. The presence of 'as' in a concept definition indicates name of 'has-a' relationship. Meta property is not considered as part of the concept but can be used during prepossessing. Following text shows an example description of a node using concept language:

Example 22.

```
concept node : Node {
  meta ontology = http://dps.../~mumtaz/glare/segrid_2.0.owl
  meta conceptBuilder = org.askalon.gridarm.segrid.ConceptBuilder
  meta reasoner = http://kreusspitze.dps.uibk.ac.at:8081
  concept Host : Cluster {
    name = ${HOSTNAME}
    concept CoreDuo : Itanium {
      clockSpeed = cpu.clockSpeed()
      vendor = Intel
      model = Centrino Duo
      meta cpu : infoProvider =
               org.askalon.gridarm.segrid.ip.CPUInfoProvider
      concept IA64 : Architecture { }
    } concept Redhat : Linux {
      version = os.version()
      release = os.release()
      meta os:infoProvider = /home/mumtaz/.../libexec/osinfo
    }
  }
  concept CE : ComputingElement {
    totalCPUs = 16
    freeCPUs = ce.freeCPUs()
    meta ce:infoProvider = org.askalon.....GlueCEInfoProvider
    concept GRAM : Gatekeeper { ... }
  }
}
```

□

This example depicts a description of a *node* with a *Host* and a *CE* that is converted in OWL concepts. The *Host* is further defined in terms of CoreDuo processor and Redhat operating system, whereas *CE* is defined in terms of *GRAM* gatekeeper and properties *totalCPUs* and *freeCPUs*. Each concept may have some information providers (IP). An IP can be either a Java class or a shell script that is invoked dynamically to retrieve and substitute the actual values of associated properties at runtime. However, the values of static properties may be hardcoded as well. Besides, IPs, meta properties can also be used, for example, to specify URI of the foundation ontology, URL of a reasoner, and a concept builder class that transforms specified concepts into OWL classes and properties.

The concepts defined by the Grid resource requester or provider are then automatically translated into OWL, and a description logic reasoner such as pellet is used for inference. The translation or ontology building is implemented using Jena Ontology APIs [32]. Jena allows us to create new ontology model using a set of concepts. Furthermore, new individuals are created dynamically once a resource is available. In ontological terms, an *individual* refers to as a concrete description of a resource.

Furthermore, we propose the same concept language to define high-level user requirements or goals as a request that is transformed into SPARQL query:

Example 23.

```
concepts query : Goal {
  meta select = node totalCPUs
  concept node : Node {
    concept host : Host {
      constraint hasOperatingSystem {
          bounds range=Redhat
      }
    }
    concept ce : ComputingElement {
      constraint and {
        constraint totalCPUs {
          bounds min=4
        } constraint freeCPUs {
          bounds min=2
        }
      }
    }
  }
}
```

This query of type *Goal* can be described in plain English as:

> Select a *node* and its *totalCPUs* if the *node* has `Redhat` as operating system and a computing element with at least 4 *totalCPUs* and at least 2 *freeCPUs*.

This query can be transformed into SPARQL format that looks as follows:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX seg: <http://segrid.gridarm.askalon.org/segrid_2.0#>
SELECT ?node ?totalCPUs
WHERE {
  ?node rdf:type                seg:Node .
  ?node seg:hasHost             ?host .
  ?node seg:hasComputingElement ?ce .
  ?host rdf:type                seg:Host .
```

```
  ?ce rdf:type                    seg:ComputingElement .
  ?ce seg:totalCPUs               ?totalCPUs .
  ?ce seg:freeCPUs                ?freeCPUs .
  FILTER ((?totalCPUs>8)&&(?freeCPUs>8))
}
```

Here `seg`, `rdfs` and `rdf` are prefixes referring to namespaces of our foundation ontology, RDF, and RDFS respectively. □

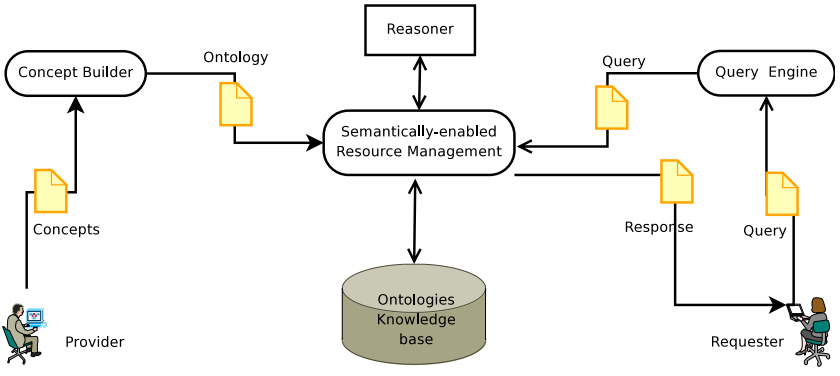


Fig. 7.1. Concept description to resource provisioning flow.

Resource providers can define and register their resources with the resource manager that can be contacted by the users/requesters in order to make requests, and, to get semantically generated matching offers. Figure 7.1 depicts an information flow starting from concept description by resource provider to semantically enabled resource provisioning by resource manager.

7.3 Architectural Extension

As described in Chapter 3, *GridARM* consists of a set of services loosely coupled in a service-oriented fashion. As shown in Figure 7.2, three core services include a *broker* (Chapter 3) that performs matchmaking of physical resources such as computers, the *GLARE* (Chapter 4) that is a Grid application registration, deployment, and provisioning framework, an *allocator* that provides agreement management, negotiation, and capacity planning for computing resources (Chapter 5, Chapter 6).

This chapter introduces the semantics based extension to *GridARM* in which the core services are extended with semantically-enabled services loosely coupled to enable coordinated resource sharing and provisioning with semantics. In such an environment, both kind of services can co-exist and coordinate

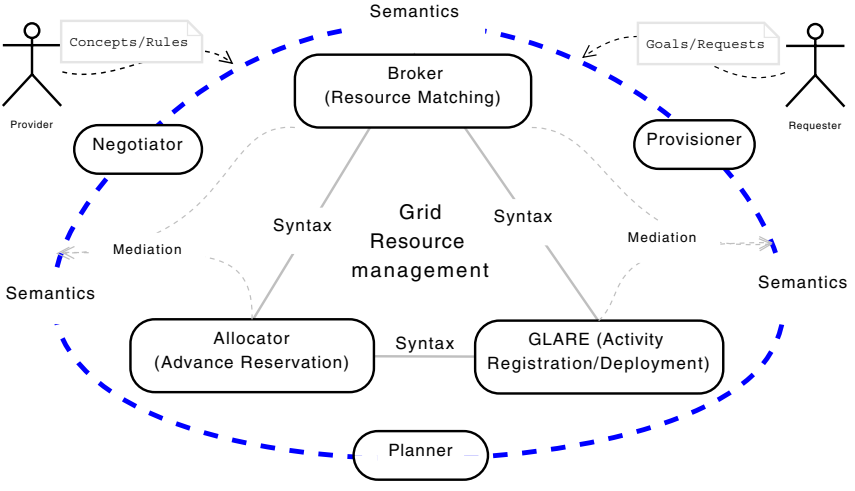


Fig. 7.2. Grid Resource Management Architecture.

with each other according to the vision of proposed architecture for the Semantic Grid. Semantically-enabled services can interact with multiple shared reasoners [178, 160] for entailment of ontological concepts advertised by service providers. For instance, one reasoner can be used to policy entailment, whereas other can be used for resource entailment. The main services include:

- a `provisioner` that performs resource matching with semantics covering both physical (computers) and logical (activities) resources. This service mainly coordinate with resource broker (Chapter 3) and GLARE (Chapter 4),
- a `planner` service extends planning capability of allocator (Chapter 5) with semantics for optimal resource allocation. For example, a planner can take two major responsibilities; planning for optimal allocation, and planning for optimal execution. The later is responsibility of scheduler therefore we focus on planning for optimized resource allocation;
- a `negotiator` is a semantic extension of reservation and negotiation mechanism (Chapter 6) provided by the legacy allocator.

7.4 Resource Ontologies

Physical and logical resource ontologies are proposed in the form of OWL-DL concepts that is collectively referred to as *base ontology* or *foundation ontology*. A subset is shown in the Figure 7.3. The physical resource ontology describes concepts related to the Grid nodes such as *processor*, *operating system*, whereas logical resource ontology represents concepts associated with logical resources such as activities (Definition 6), agreement (Definition 34),

configuration etc. Based on these concepts, new entailment may also be inferred by the resource manager/provisioner. For instance, a specific purpose *resource ensemble* with aggregated power of physical resources or synthesized capabilities of logical resources.

Resource providers and requesters use these ontologies to describe their resources and *resource requests* in an ontological format. The resource matching mechanism uses these ontological concepts in reasoning while matching a resource request with available resources.

We created a basic ontology in OWL-DL [146] using Protege [1]. It defines fundamental concepts of Grid nodes and activities. Advertisement of resources is done by providing semantics descriptions either by using a simple concept description mechanism (Section 7.2.1), or automatically transforming from syntactical descriptions generated by resource discoverer (Section 3.3.1) as node descriptions or registered in *GLARE* as an XML-based internal representation of activities. Each activity is modelled as an extension of basic concepts defined in the foundation ontology.

7.4.1 Physical Resource Ontology

The physical resource ontology provides a vocabulary that enables resource providers to describe their resources in a more expressive way. The basic ontological model can be extended to add more domain specific concepts while describing a specific resource.

It is proposed to represent the concepts related to the Grid resources as OWL-DL classes in a hierarchical way. The resource description is defined as the boolean combination of a set of constraints over the resource concepts and properties. The *constraints* can be expressed either through OWL restrictions or XML schema restrictions. The resources can be described by using different classes and properties and also by importing domain specific ontologies. In order to use the ontology model more effectively, a resource request is also considered as a hypothetical resource and can be subsumed in the latest model of the resource ontology available at the time of request.

The base ontology of the physical resources consists of classes and properties that describe `nodes`, `Network`, `Storage`, like `Computing element`, `Cluster`, `SubCluster`, and `Host`. These classes includes `Policy`, `Processor`, `OperatingSystem`, `Architecture`, `Filesystem`, `Memory`, `State`, etc. A `Computing element` of a resource consists of concepts like resource `Info`, `State`, `Jobs` and `Policy`. Each class defines the most generic prospects of the concept being modelled. In order to achieve this model using OWL, each class is defined to be a subclass of a set of anonymous classes and each class restricts some of its properties. For instance, the `Node` class shown in Figure 7.3 is defined as subclass of several anonymous classes that each of which restricts one of the class properties such as `hasFileSystem`, `hasArchitecture`, `hasNetworkAdapter`, etc.

[1] An ontology editor available at http://protege.stanford.edu

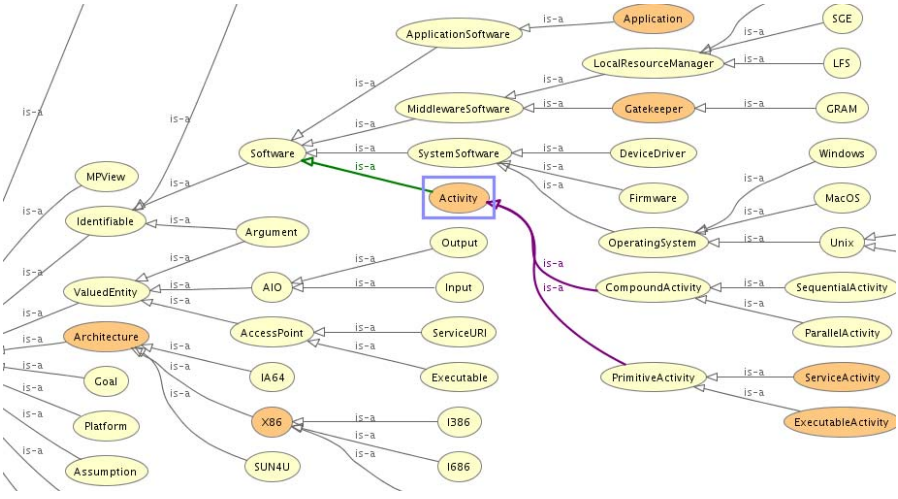


Fig. 7.3. An incomplete class hierarchy of different concepts of the Grid resources.

The vocabularies are extensible. A user can extend them asymmetrically without loosing their semantics. For example, we can add a new term *NetGear* as a specialization of `GigabitEthernet` which is not compatible with `Solaris` operating system. The new term added independently by a user can be inferred by a reasoner as it `is-a` NetworkAdaptor. Also integrity of requests can be verified by the reasoner with request satisfiability check, e.g. if someone requests that:

> 'I need a computing resource with `Solaris` OS and with `NetGear` Ethernet adapter'

The reasoner can easily identify that this request cannot be fulfilled since `NetGear` is incompatible with the OS `Solaris`.

7.4.2 Resource Ensembles

The resource ensemble ontology deals with the conceptual grouping of Grid resources. A resource provisioner generates different resource ensembles based on the concepts and restrictions given under the foundation ontology. The resources in a resource ensemble share some common features; either they collectively provide a new and complex capability or a more powerful aggregated capability. For instance, a set of all nodes that share a specific network filesystem, or a set of nodes, in which each node is connected to other nodes through a specific/common fast interconnect.

A resource ensemble ontology may also group nodes in ensembles in which enclosed nodes collectively achieve a certain minimum number of `Mflops` as part of the ensemble. In this way a resource ensemble provides a required

number of `MFlops` which is otherwise not possible by a single node. This kind of resource ontology enables a resource provisioner to accept requests in a more generic form that is closer to a natural language.

7.4.3 Logical Resource Ontology

Similar to physical resource ontology, logical resource ontology covers concepts related to activities (Definition 6): software components with functional and non-functional properties and arguments, and workflow applications (Definition 10). Using a basic vocabulary of concepts defined as part of the foundation ontology, an application provider can add new concepts asymmetrically. That means, an application provider doesn't have to restrict itself to a specific terminology, concept, or property names and/or their values. For instance, an activity can be categorized as *PosixActivity* and *ServiceActivity*. A PosixActivity can be further categorized according to different versions, supporting architectures, and other QoS based categories. Similarly, *input/output* arguments can be defined in terms of primitive and complex arguments: a *FileArgument* can be further categorized as *ImageDescription*, *DocumentArgument*, and *MediaFile* etc.

Following example shows a description of an application with *DVI2PDFConverter* as a *PosixActivity*, whereas *DVIFile* as an *input DocumentArgument* and *PDFFile* as an *output DocumentArgument*.

```
concepts PDFTools {
  concept DVI2PDFConverter : PosixActivity {
    ...
  }
  concept DVIFile : DocumentArgument as input { ...
  }
  concept PDFFile : DocumentArgument as output { ...
  }
  concept Reliability : QualityOfService {
    property value : xsd:float qos.reliability()
    .....
  }
}
```

Using a subsumption hierarchy of concepts, an application requester can describe its high-level requirements. These requirements are mapped to concrete descriptions after reasoning with the latest ontological model. In this way multiple alternative options can be generated and proposed matching user requirements.

7.5 Discovering Resources with Semantics

As described in Section 2.5.3, both SPARQL and OWL-QL can be used for querying ontology models and knowledge bases. SPARQL is centered around

RDF whereas OWL-QL is specifically designed for OWL(-DL). The OWL-QL can be used to describe a resource request in a simple and very expressive format by using a collection of the OWL *facts and axioms*. An OWL-QL query includes a query pattern as a collection of OWL sentences and a list of *must-bind*, *may-bind*, and *don't-bind* variables. These different types of variable bindings distinguish OWL-QL from other query languages and help in accessing alternative or close matches. Furthermore, a query optionally includes a query premise, an answer pattern and a reference to *answer knowledge base*. For example, a client can request for nodes with the following query pattern:

Example 24.

```
Query:
  ("Which nodes have 64bit Solaris operating system?")
  Query Pattern: {(hasOperatingSystem ?node ?os)
                  (type               ?os   Solaris)
                  (hasArchitecture    ?os   64Bit)}
 Must-Bind Variables List : (?node)
 May-bind Varables List   : ()
 Don't-bind Variable List : ()
 Answer Pattern           : {(?node)}
Answer:
  ("altix1.uibk.ac.at" "hcma.uibk.ac.at")
```

□

The OWL-QL is designed for answering queries of the form *"What URIrefs and literals from the answer knowledge base and OWL denote objects that make the query pattern true?"* [66].
In the OWL-QL queries, URLs of answering servers and references to the answer knowledge bases can be specified. This feature can be exploited in making a distributed *resource matching* framework. For example, a resource ontology and a usage policy knowledge base can be installed on different nodes and a requester can specify them dynamically while making a request. The use of human readable surface syntax for queries and answers are very useful in a *Semantic Grid* context. It could be possible to devise a translation mechanism in which a simple request in the form of a formal natural language syntax could be translated into a query pattern. For instance, in the above example the *modal verbs* can be replaced with a variables to be found/bound (e.g. *'which node'* with *?node*), objects with OWL classes, and so on. An equivalent query in SPARQL, of OWL-QL example shown above, looks like:

Example 25.

`SELECT` a node `?node` having `Solaris` operating system with `64Bit` architecture.

```
SELECT ?node
WHERE {
  ?node seg:hasOperatingSystem ?os .
  ?os   rdf:type               ?Solaris .
  ?os   seg:hasArchitecture    ?arch .
  ?arch rdf:type               64Bit .
}
```

□

Example 26. This example shows that a client can request for an activity with the following SPARQL query:

> `SELECT` an activity `?activity` that converts a DVI document into a PDF document and has reliability greater than 0.5.

```
SELECT ?activity
WHERE {
  ?activity seg:input  ?input .
  ?input    rdf:type   seg:DVIFile .
  ?activity seg:output ?output .
  ?output   rdf:type   seg:PDFFile .
  ?activity seg:hasReliability ?reliability .
  ?reliability seg:hasValue ?rvalue .

  FILTER (?rvalue > 0.5)
}
```

□

A query premise can also be added like `Linux` and `IA64` as shown in the following SPARQL example below:

Example 27.

> If a node has `Linux` operating system and `IA64` architecture then what is the available memory of that node?"

```
SELECT ?node ?freeMemory
WHERE {
  ?node rdf:type               seg:Node .
  ?node seg:hasOperatingSystem ?os .
  ?os   rdf:type               Linux .
  ?node seg:hasArchitecture    ?arch .
  ?arch rdf:type               IA64 .
  ?node seg:hasMemory          ?mem .
  ?mem  seg:freeMemory         ?freeMemory .
}
```

```
Answer: node                        | freeMemory
--------------------------------------
hcma.uibk.ac.at           | 2.4GB
```

□

SPARQL provides a mechanism to add filters in the query.

Example 28.

Select a set of nodes with `ScientificLinux` as operating system, at least $3GB$ free memory, at most 64 total CPUs, and free CPUs not less than 16.

```
SELECT ?node ?freeMemory ?totalCPUs ?freeCPUs
WHERE {
  ?node rdf:type                seg:Node .
  ?node seg:hasOperatingSystem  ?os .
  ?os   rdf:type                seg:ScientificLinux .
  ?node seg:hasMemory           ?mem .
  ?mem  seg:freeMemory          ?freeMemory .
  ?node seg:hasComputingElement ?ce .
  ?node seg:totalCPUs           ?totalCPUs .
  ?ce   seg:freeCPUs            ?freeCPUs .
  FILTER
   ((?totalCPUs<=64)&&(?freeCPUs>15)&&(?freeMemory>3000))
}
Answer:
?node                      |?freeMemory |?totalCPUs |?freeCPUs
-------------------------------------------------------------
hcma.uibk.ac.at            |   3.4GB    |   64      |   20
altix1.jku.austriangrid.at|   3.1GB    |   48      |   16
```

□

7.6 Subsumption-Based Resource Matching

The resource matching mechanism is introduced based on the subsumption of concepts and roles represented by the *description logics* formalisms. *Description logics* techniques are employed to classify the Grid resource descriptions. The resource descriptions are maintained in a hierarchy of concepts or classes, where classes are linked through roles established by using the restrictions on classes and properties. Each resource description is embedded in the hierarchy persistently once it is satisfied. A request for the resource(s) could also be made similar to the resource description and subsumed in the taxonomy.

This mechanism enables the Grid resource provisioner to generate an exact or a close match. Based on the subsumption of the request in the resource

ontology model, the provisioner can easily suggest alternative options that can be exercised if an exact match can not be found. This kind of alternative offering is hard to achieve without an ontological model. The resource subsumption in the taxonomy and the request matching is performed by considering the matching concepts based on the following propositions:

- $\texttt{Request} \sqsubseteq \texttt{Resource} \vee \texttt{Request} \equiv \texttt{Resource}$
  This shows that the `Request` is a sub concept or an equivalent concept of the `Resource`, which means an exact and ideal match satisfying all necessary and sufficient conditions.
- $\texttt{Request} \sqsupseteq \texttt{Resource}$
  Represents a `Request` as a super concept of `Resource`. Resources belonging to the super-concept do not fulfill all constraints set by the request, but are considered as the best possible match and used as an alternative option.
- $\neg\,(\texttt{Request} \sqcap \texttt{Resource} \sqsubseteq \bot)$
  The intersection of both `Resource` and `Request` concepts is satisfiable and considered as a least suitable option as an alternative.
- $(\texttt{Request} \sqcap \texttt{Resource} \sqsubseteq \bot)$
  This means that there is no match possible.

These concept matching propositions clearly organize the relationships in a well defined discrete scale and provides a solid ground for a *Description Logics* reasoner to be used for the classification of the `Request` to compute its subsumption relationship with all registered resources.

7.7 Evaluation

The foundation or basic ontology is created by using Protege ontology editor with OWL plugin [177]. Then our simple concept language (Section 7.2.1) converter, developed in Jena [32], is used to dynamically generate new concepts and populate knowledge base. The Pellet reasoner is used to see the classification of a resource request in the knowledge base. The subsumption of resource requests in the model proves the propositions given above and matches with OWL-QL query result. According to our initial findings, the time needed for inference and subsumption grows significantly with increasing number of ontological concepts. Although this overhead matters, but in the Grid, the number of instances of the concepts is more important.

We compared throughput of proposed semantics-based asymmetric resource matching with syntax-based symmetric resource match in *GLARE* and WS-MDS. We performed following tests with activity (Definition 6) entries of real world applications deployed in the Austrian Grid [33]. We use the simple concept description language for semantically describing activity concepts, and Jena [32] APIs for translating concepts into ontology model. Same set of activities were registered in WS-MDS and GLARE as described in Section 4.5.

Figure 7.4 and 7.5 demonstrate comparison of semantically enabled resource matching with conventional asymmetric attribute-based approaches adapted by *GLARE* and WS-MDS (Section 2.3.3). Both figures represent throughput. Figure 7.4 depicts with varying number of concurrent clients whereas Figure 7.5 depicts with varying number of entries respectively. It is shown that semantics has a greater overhead resulting in less throughput. This is because of: the expensive mechanism required for the reasoning, and XML-based OWL that is very verbose and thus takes longer for parsing. However, the semantics based resource matching mechanism is adapted for accurate provisioning of resources instead of performance.

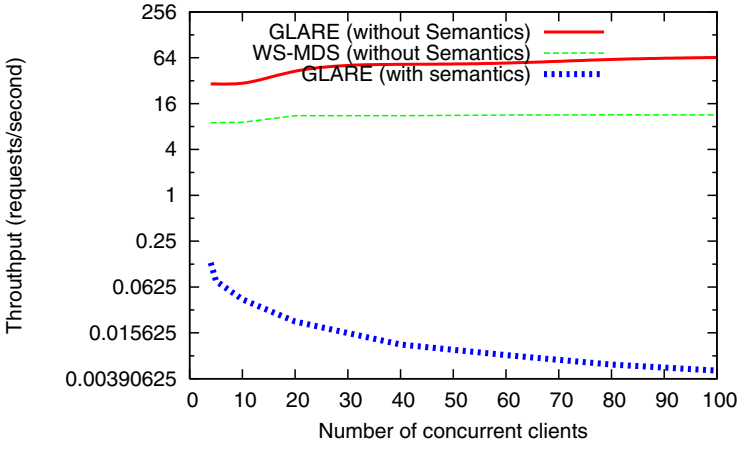


Fig. 7.4. Comparison with varying number of concurrent clients.

7.7.1 Subsumption: An Example

The following example demonstrate the usefulness of the taxonomy subsumption and OWL-QL. It contains a query to search for nodes with *Solaris* operating system and with *SGE* as *local resource manager (LRM)*.

Example 29.

```
SELECT ?node
WHERE {
  ?node rdf:type                 seg:Node .
  ?node seg:hasOperatingSystem   ?os .
  seg:Solaris  owl:subClassOf    ?ost .
  ?os   rdf:type                 ?ost .
  ?node seg:hasComputingElement  ?ce .
  ?ce   seg:hasLRM               ?lrm .
```

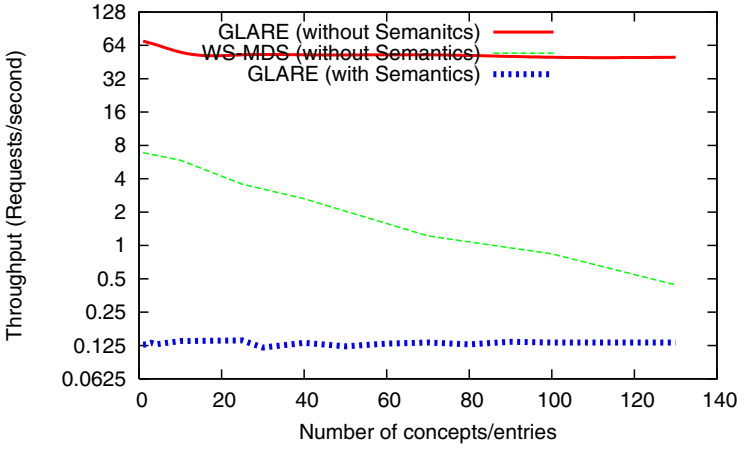


Fig. 7.5. Performance comparison with varying number of entries/concepts.

```
  ?lrm  rdf:type                ?SGE .
}
Answer: ("Node4, Node5, Node1")
```

This query is performed at the time when the resource ontology model as shown in the Figure 7.6 with solid lines was available. The presence of `'Solaris subClassOf ?ost'` statement indicates that the provisioner can return not only the exact match but close matches as well.

The example ontology shown in Fig. 7.6 explains how the resource matching is achieved. At the time of request we have a subtree of the main subsumption tree in the system with seven Grid nodes. Nodes are individuals of concepts organized in a hierarchy in which resource description in each lower level is a specialization of the resource concept given in the upper levels. We added the request to the knowledge base as follows:

```
concepts Request : Goal {
  concept node : Node {
    concept host : Host {
      constraint hasOperatingSystem {
          bounds range=Solaris
    }
  }
    concept ce : ComputingElement {
      constraint hasLRM {
        bounds range=SGE
      }
    }
  }
}
```

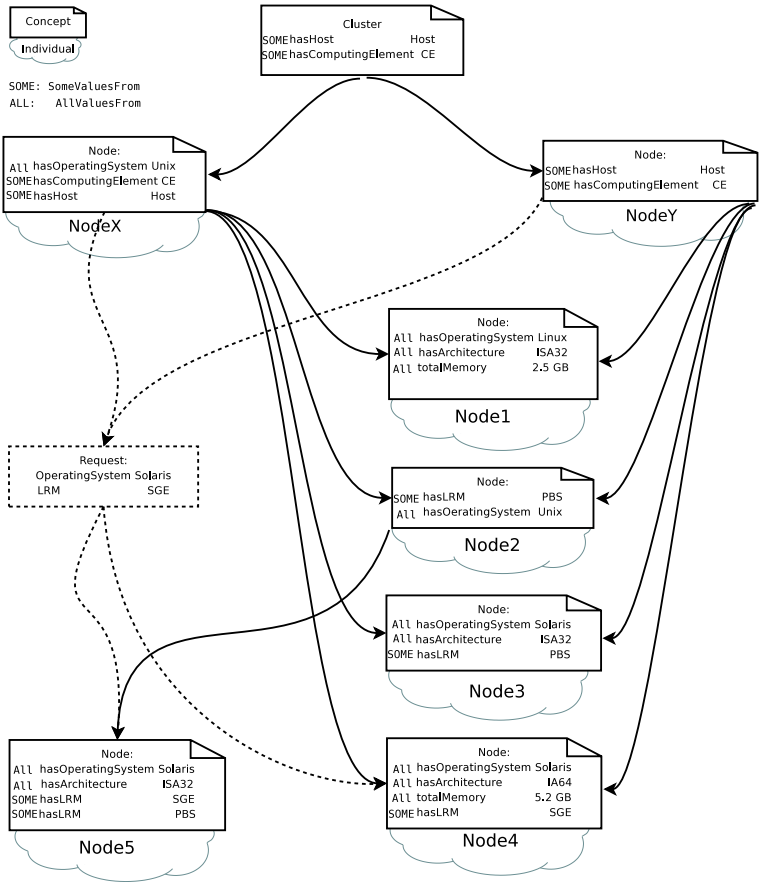


Fig. 7.6. Subsumption of a `Request` in the existing resource taxonomy.

The request is transformed in OWL concepts and then classified in the subsumption tree by the reasoner as shown in the subsumption tree as `'Request'` surrounded by the dotted rectangle and its relation with other nodes is depicted as dotted lines. It is observed that nodes `Node4` and `Node5` are specialized concepts of the `Request` and as such are marked as `exact` matches. There is no equivalent concept. If we look for the super concepts of the `Request` up to the root, then nodes `NodeX` and `NodeY` would be marked as a match. By employing the third proposition as specified in Section 7.6, node `Node1` is found compatible with the request. All other nodes are declared inconsistent as the restrictions over the properties do not match.

7.8 Related Work

The most prominent information services in the Grid and Web communities are the Meta-computing Directory Service (MDS) [39] and UDDI [126] respectively. These services support a simple query language for the resource and service selection but lack in descriptive expressiveness. Thereby, there is no sophisticated resource matching mechanism available. In the traditional Grid resource management systems, different syntax-based synchronous resource matching mechanisms are used. These mechanisms provide expressiveness to some extent but the drawback is that they still require symmetric attribute-based description mechanisms.

The Condor [128] system is one example in which a symmetric syntax-based matchmaking is performed for the resource allocation in the Grid infrastructure. For this purpose, a classified advertisement-like matchmaking framework has been developed. In this framework, resources and requests are described in the form of attribute *name-value* pairs and the resource consumers and providers specify their matching constraints. These constraints are then evaluated to determine a match for each *request* with available resources. The drawback of this matchmaking is that it works only if both request and resource descriptions use the same attribute names and agreed upon attribute values, it fails otherwise. A sample *Request ClassAd* can be specified as:

Example 30.

```
Request JobClassAd:
[
      Type = "Job";
      Owner="mumtaz";
      Constraint =
          type == "Machine" &&
          Arch == "Intel"   &&
          Disk >= 20000     &&
          OpSys =="Linux260";
]
```

This example shows a request for a machine with `Intel` architecture, `Linux` operating system, and at least 20GB disk space. The *Resource ClassAd*s are described in a similar way by using the similar syntax. The `constraint` clause of request `classAd` is matched with the `constraint` clauses of resource `classAds` to find the solution. The disadvantage of this system is that both the resource provider and the requester have to agree on a unique syntax and they cannot extend the terms or concepts independently. In contrast, this kind of asymmetric extension of concepts, that can be performed independently, is possible in our proposed resource matching mechanism. For instance, as shown in Figure 7.7, the term `Intel` can be extended to `Pentium` and `Xeon` without coordination between the participants and the provisioner can automatically understand the semantics of the new terms.

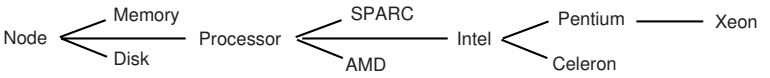


Fig. 7.7. Processor Class hierarchy.

The work described in [170] is the first effort for ontology-based Grid resource matching devised based on semantic web technology. In this work, an asymmetric description of resources and requests are modeled and described separately. Instead of a syntax-based resource matching, a semantics based matchmaking is proposed. Due to the *asymmetric description*, no coordination between resource providers and consumers is required before adding a new vocabulary. A simple example of a *Job Request* is given below:

Example 31.

```
JobRequest.Name "request1"
JobRequest.Owner "mumtaz"
JobRequest.JobType "MPI"
JobRequest.RequestResource.ResourceType
        "ComputerSystem"
JobRequest.RequestResource.RequiredOS.OSType
        "Unix"
```

Based on the domain background knowledge and rules specified in TRIPLE [67], the matchmaker concludes that `Linux` and `SunOS` can be used as a `Unix` operating system, and the requested MPI job can run on any tightly-coupled machine like Linux cluster or a shared memory system. The system is based on RDF-Schema for an ontology description. The domain background knowledge and matching rules, described in the TRIPLE rule language, are explicitly required to perform resource matching. The TRIPLE rules are first compiled into *XSB rules*, which are then compiled into instructions for *XSB virtual machine*. TRIPLE/XSB evaluates rules and finds the best match for the request with the help of background knowledge and ontologies. The disadvantage of this system is the overhead of explicit rules definition when the semantics vocabulary increases. Furthermore, the recursive rules and two-phase compilation is time consuming. The underlying ontology language of this system, i.e. RDFS, supports a smaller set of semantic vocabulary (W3C approved axioms and constructors) as compared to the OWL-DL which is the language of our proposed system. An extension to the standard RDFS is possible but it leads to the requirement of a non-standard specialized reasoner and query constructs. Our proposed system has no such limitation.

Although one can extend standard RDFS semantics to make it more expressive by introducing new constructs, but then one has to implement a specific reasoner and have to construct specialized queries in order to take advantage of new constructs. This approach is against standardization efforts being done by the Semantics Web community.

7.9 Summary

This chapter introduces semantic enhancement of resource provisioning and management for the Grid. The semantics are introduced by proposing an ontology-based resource description and resource matching mechanism which is used to make Grid resources available on-demand. It is shown that OWL is a powerful language that suits our needs. It is demonstrated that SPARQL can be used as the request-response mechanism to find exact or close matches.

A simple concept description language is introduced that can be used by requesters and providers to represent `resource/request` descriptions. Resources can be described asymmetrically. These descriptions are then automatically translated into OWL concepts and knowledge base and requests are translated into SPARQL query. A taxonomy subsumption mechanism is introduced that can be exploited to generate alternative offers if exact match is not possible.

The advantages of the proposed semantics-based resource matching mechanism is highlighted by providing several examples. It is observed that on-demand provisioning of the Grid resources can be improved significantly with the possibility of semantics-based alternative offer generation process.

8

Semantics-Based Activity Synthesis: Improving On-Demand Provisioning and Planning

In the previous chapter, the possible role of semantics in Grid resource matching and brokerage is discussed. This chapter introduces semantics-based synthesis of activities for automatic workflow generation and improving on-demand resource provisioning. On-demand synthesis of Grid activities plays a significant role in automatic workflow composition and in improving service quality of a Grid resource provisioner. However, in the Grid, synthesis of activities has not been considered due to limited expressiveness of the representation of activity capabilities and the lack of adapted resource management means to take advantage of such activity synthesis. This chapter introduces a new mechanism for automatic synthesis of available activities for the Grid by applying ontology rules. Rule-based synthesis combines multiple primitive activities to form new compound activities. The synthesis process generates new compound activities that can be provisioned as new or alternative options for negotiation and advance reservation. This is a major advantage compared to other approaches that only focus on resource matching. The newly generated compound activities provide aggregated capabilities that otherwise may not be possible; this leads towards an automatic generation of complex workflow applications. Furthermore, we introduce semantics in capacity planning for improving optimization in resource allocation. We demonstrate advantages of semantic-based automatic synthesis of Grid activities.

8.1 Introduction

The Grid enables resource sharing and coordinated problem-solving across computers and humans in a distributed and heterogeneous environment [144]. In such a complex and dynamic environment, some pf the challenges lie in coupling resources with components of potential applications which may be executed across matching resources as workflows. Enabling scientific workflow applications for the Grid has been identified as an important research topic [58, 197, 63, 112, 157] and is an ongoing challenge for the research

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 179–198, 2010.

community working in the area of workflows [62]. A *Grid workflow* (Definition 10) as a collection of *activities* (software components) may be executed across multiple Grid nodes in order to achieve a common goal [136]. Abstract activities are mapped to concrete deployments at runtime in order to deal with the dynamicity and heterogeneity of Grid resources (Chapter 4). The workflow mapping and execution has been automated [62, 158] but the workflow composition is still manual. Decoupling of abstract activity descriptions from concrete deployments has been successfully done as described in Chapter 4. This enables a Grid user to manually compose an abstract workflow independently from the dynamic Grid environment. The manually generated workflow can be mapped dynamically to Grid resources at runtime.

Automatic synthesis of Grid activities plays a significant role in automatic composition of workflows and in improving provisioning quality of the Grid resource manager. The *synthesis* involves combining or integrating multiple activities into a single complex activity. Performing such a synthesis is useful in: 1) improving usability of the Grid by aggregating capabilities of underlying resources, 2) generating multiple options built on scarce resources (activities) to be provisioned on-demand by the Grid resource manager, and 3) optimizing resource allocation with adapted planning mechanism.

However, synthesis of activities in the Grid has been largely ignored due to limited representation of their capabilities and lack of adapted resource management mechanisms to take advantage of the newly generated activities.

This chapter introduces a new mechanism for automatic synthesis of activities by applying the ontology rules [183]. In the extendable foundation ontology each primitive activity is modelled as a separate concept in terms of its *inputs*, *outputs*, *usage assumptions* and *after effects* [180]. Ontology rules can be used to define new concepts and applying them to existing ontology results in new entailment. We introduce a set of new rules for synthesis of activities which integrate primitive activities to form new synthesized activities. These rules are defined by following a set of well defined workflow patterns and applied to the activity *knowledge base* with the help of rule-based reasoning tools such as Pellet [160] or Racer [178]. Rules-based semantics gives the declarative descriptions of activities a well-defined meaning by specifying ontological foundations and by showing how such foundations are realized in practice. Activities are described obtrusively, advertised dynamically, and provisioned on-demand according to user goals.

The extension in the semantics work, presented in previous chapter, with activity synthesis enables a resource manager to become a smart provisioner that automatically generates a set of complex activities according to defined rules and delivers them on-demand. These complex activities can be treated as standalone workflows or can be used as building blocks of new workflows. A workflow can be generated as a side effect in the form of an abstract workflow. Askalon's high-level *Abstract Grid Workflow Language (AGWL)* [62], execution and scheduling services, and a set of monitoring and prediction tools are used to:

- visualize workflows,
- map their components to the concrete deployments [158],
- execute onto the Grid [62].

Since synthesized activities provide new or aggregated capability of its founding constituents, provisioner becomes a better capacity planner and negotiator. After the synthesis, a provisioner gets more to offer possibly with different service qualities than originally it could have been left with. The provisioner accepts user goals and generates workflows based on user goals leaving the user to focus on its problem.

Furthermore, semantics are introduced not only for resource matching but for the capacity planning as well. The role of semantics in optimizing resource allocation is demonstrated.

The resource descriptions may be queried with abstract goals, may foresight required capabilities, or may be checked to avoid inconsistencies in the declarations. Rules-based management builds a rigorous approach towards giving the declarative descriptions of resources a well-defined meaning by specifying ontological foundations and by showing how such foundations may be realized in practice. Semantic description and rule-based synthesis of primitive resources enable a provisioner to accept user goals, generate complex workflow and make it available for execution. This enables a Grid user to focus on its problems and let the provisioner to help in achieving its goals.

8.2 Motivation

The Semantic Grid is considered as an extension of the current Grid in which information and services are given well-defined and explicitly represented meaning with the power to enable better cooperation [46]. Resource representations are enriched with semantics using ontologies [146] and discovered based on high-level user goals. A goal is a specific measurable and time targeted objective. On a personal level, goal setting is a process that allows people to specify the work towards their own objectives. In the Grid, a goal may be a data set that is to be generated by using some input. The generation may involve a series of activities and aggregated power of computing resources. In the Grid, aggregated capability of activities can be utilized to achieve a goal that otherwise may not be possible especially with needed quality of service.

Provision of aggregated computing power (in MFlops) of various Grid nodes is one example of aggregated capacity. However, aggregating capabilities of software components or activities is rather challenging. The activities deployed across multiple computers may provide aggregated capability that leads to utilize aggregated *computing power*. Since activities represent self-contained autonomous software components, the aggregation is possible with synthesis.

Lets assume we want to solve a problem that needs an aggregated power of computers distributed in a Grid. We can opt for one of the following:

1. design and develop a new application from scratch that is fully integrated in the Grid and can be executed on multiple nodes in a distributed fashion,
2. create an association of a set of legacy activities that are already deployed on various nodes in the Grid and that can collectively achieve the same goal.

The later option is similar to writing a shell script by using available commands in a shell instead of developing a new application from scratch.

Analogous to a shell script, in which each command with well defined interfaces is a self contained activity, a Grid script i.e. workflow also comprises independent (legacy) applications that collectively achieve a single goal. Th shell commands are registered in its path whereas activities in the Grid are registered in a registry or information service such as *GLARE* described in Chapter 4.

This model can be explained with a concrete example. Consider a client wants to render a movie based on a textual description of its scenes. It is likely that there is no (free) tool available that can perform such task, and even if there is one, it is possible that the tool is not Grid-aware and is not able to distribute the rendering of different scenes of the movie across the Grid among different computers. Note that the movie rendering is a compute-intensive task. What we can assume is the existence of some freely available tools that collectively can perform this task. For instance, three tools can be selected as:

1. *POVray* i.e. *Persistence of Vision Raytracer* [133] is a high-quality tool for creating stunning three-dimensional graphics used not only by hobbyists and artists, but also in biochemistry research, medicine, architecture and mathematical visualization. *POVray* renders a scene description into a series of PNG files (pictures),
2. *Png2yuv* that pipes an archive of PNG frames generated by POVray, to `stdout` as a *YUV4MPEG2* stream,
3. *ffmpeg* tool that converts a stream into MPEG format or display it on a computer screen.

To perform the movie rendering on the Grid, there are following options:

1. assuming that all required tools are available on a single Grid node, a user builds a job script and submits to a Grid node by using a middleware operating environment. This is trivial and can be achieved easily e.g. by using the Globus Toolkit.
2. if all tools are available but distributed across the Grid on different nodes, then a user composes a workflow and submits to a Grid workflow execution system. This can be done with the Askalon [62] workflow composition and runtime environment and the POVray workflow is successfully executed on the Austrian Grid testbed [122]. Figure 8.1 shows a possible POVray wokflow, where different primitive or aggregated activities can be executed on different Grid sites.

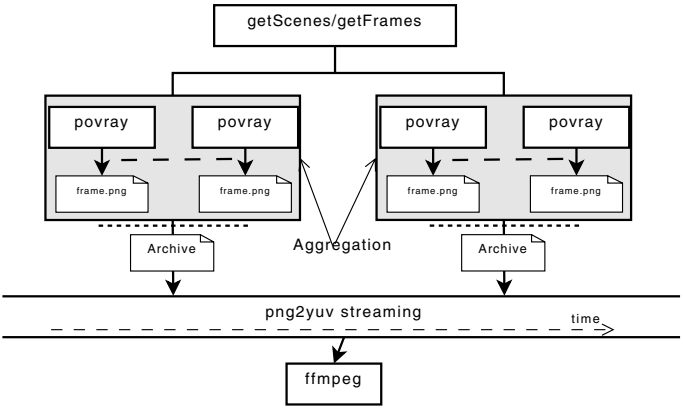


Fig. 8.1. A POVray workflow application.

POVray is known to be a very time consuming and parallelizable process; one can model `POVray` rendering scenario as a workflow depicted in Figure 8.1. According to this scenario, the description of a movie can be separated in several scenes, where each scene is composed of several frames that can be rendered as parallel activities on the Grid. Finally, all the frames are merged into a *.mpg* movie using a *png2yuv* followed by a *ffmpeg* activity.

Both options described above not only need manual steps for composition, but also require a user to have additional knowledge of each component. For instance, specific usage of each tool and whether or not several instances of the tool can be executed in parallel etc. The manual process becomes even harder if a user knows its goals but does not know how to efficiently generate a movie with the POVray description by executing some tools on the Grid. Even a graphical interface may not be helpful without consulting a POVray and a Grid expert. Furthermore, there may be several options that could do the same task with varying QoS. For instance, same task can be done with an executable or with a Web service. This further confuses a domain specialist and makes even graphical composition a non-trivial task.

However, with the help of our proposed semantic-based activity synthesis, the workflow can be generated automatically. A client needs to specify its goals, e.g. *'to generate a movie in a specific format based on textual description of scenes'*. However, it may not need to provide input descriptions, instead, the provisioner can generate multiple options with different types of possible inputs. Since each activity of a workflow can be performed by different tools, this may lead to the generation of multiple alternative options. The provisioner can select the best option based on a user or system defined criteria.

Our proposed solution gives explicit meanings to activities and makes them understandable for machine processing. Synthesis rules correlate input and output arguments of activities and combine them to form complex activities that provide aggregated or new capabilities. For instance, `png2yuv` and

`ffmpeg` can be combined to make a compound activity `png2mpeg` whose input becomes input of `png2yuv` and output becomes output of `ffmpeg`. $\mathcal{O}_{png2yuv} = \mathcal{I}_{ffmpeg} \Longrightarrow$ `png2mpeg` where $\mathcal{I}_{png2mpeg} = \mathcal{I}_{png2yuv}$ and $\mathcal{O}_{png2mpeg} = \mathcal{O}_{ffmpeg}$ This activity synthesis process iteratively generates all possible combinations and ultimately ends up with a workflow that fulfills the user's goals. In case of `POVray`, the input of complex resource is a textual description of scenes and output is a `MPEG` movie. The ontology rules for iteratively generating complex workflows (compound activities) are explained in Section 8.3.1.

A provisioner with multiple options becomes a better negotiator for the resource. It can offer alternative options with varying QoS. For instance, a provisioner may offer the following alternative options that independently fulfill a similar goal:

1. a single application or service that is available on a Grid node at a certain timeframe for which a client needs to make advance reservation (Definition 21).
2. a set of tools that can collectively perform the same task. The provisioner can generate a workflow and offer it to the client. The client then contacts the workflow enactor service to execute the generated workflow onto the Grid.
3. a provisioner can propose to generate a generic or tailor-made service using a wrapper generator [83, 93, 50]. This could be relatively expensive but the trade-off is that a client does not care about additional steps of advance reservation or workflow scheduling as the generated wrapper service works as a dedicated self-contained workflow execution service.

8.3 Synthesis Model

The foundation ontology includes concepts that are associated with logical resources such as activity, input/output argument, assumption, effect, primitive and complex activities, application, software, gatekeeper etc. Agreement concepts represents a special kind of logical resources including agreement, temporal concepts, reservation etc. The ontologies are asymmetrically extensible thus activity providers can easily extend fundamental concepts without loosing their semantic soundness.

8.3.1 Ontology Rules

Although a reasoner can derive additional ontological entailment, based on property and class hierarchies i.e. subsumptions, sometimes it is necessary to infer pertinent information that cannot be determined otherwise. This emphasizes the need of rules [32, 183]. Since rules are based upon OWL Lite and DL therefore they may have bindings to the underlying ontology. Ontology rules play a vital role in the proposed activity synthesis. Foundation ontology also

includes a set of rules that are fired iteratively and are applied to generate complex activities by combining legacy primitive activities as basic building blocks in a well defined pattern. The synthesized activities either provide a new capability or aggregated capability of its basic building blocks.

8.3.2 Activity Synthesis Problem

The synthesis of activities is a problem of combining multiple activities by correlating output of one activity to the matching input of an other activity in order to form a new synthesized activity. Let $\mathcal{A}$ be a set of activities involved in the synthesis, $\mathcal{M} = \mathcal{I} \cup \mathcal{O}$ a set of arguments (like files, integers etc.), then a synthesized activity α is modelled as a triple:
$\alpha = \{\mathcal{I}_\alpha, \mathcal{O}_\alpha, \mathcal{A}_\alpha\}$ where $\alpha \cap \mathcal{A}_\alpha = \oslash \wedge \mathcal{I}_\alpha \cap \mathcal{O}_\alpha = \oslash$ and $\mathcal{I}_\alpha \subseteq \mathcal{M}$ is a set of input arguments of α, $\mathcal{O}_\alpha \subseteq \mathcal{M}$ is a set of output arguments of α and $\mathcal{A}_\alpha \subseteq \mathcal{A}$ is a set of activities which are combined during synthesis to form α. Let $\mathcal{I}_\beta = \bigcup_{\forall \delta \in \mathcal{A}_\alpha} \mathcal{I}_\delta$ is a set of input arguments of all activities $\in \mathcal{A}_\alpha$, $\mathcal{O}_\beta = \bigcup_{\forall \delta \in \mathcal{A}_\alpha} \mathcal{O}_\delta$ is a set of output arguments of all activities $\in \mathcal{A}_\alpha$, and $\mathcal{M}_\beta$ is a set of arguments that are produced and consumed internally by activities $\in \mathcal{A}_\alpha$

$$\mathcal{M}_\beta = \mathcal{I}_\beta \cap \mathcal{O}_\beta \text{ then } \mathcal{I}_\alpha = \mathcal{I}_\beta - \mathcal{M}_\beta \text{ and } \mathcal{O}_\alpha = \mathcal{O}_\beta - \mathcal{M}_\beta.$$

The synthesis can be explained with help of example activities shown in Figure 8.2. A set $\mathcal{A}$ of primitive activities is given as $\mathcal{A} = \{\texttt{a,b,c,d,e,f,g,h,i}\}$. Activity `'a'` is defined as $\{\{5\},\{6\},\{\oslash\}\}$: a primitive activity with argument 5 as input and argument 6 as output. Similarly, activity `'b'` is defined as $\{\{6\},\{7\},\{\oslash\}\}$: a primitive activity with argument 6 as input and argument 7 as output. Since the output of activity `'a'` is same as the input of activity `'b'`, therefore both activities can be combined to form a new synthesized activity `'A'`, such that $\texttt{A} = \{\{5\},\{7\},\{\texttt{a,b}\}\}$: a compound activity with argument 5 as input, argument 7 as output, and `a,b` as internal primitive activities of `'A'`.

Similarly, a compound activity `'C'`, as shown in Figure 8.2, is defined as $\texttt{C} = \{\{15,16\},\{18,19\},\{\texttt{h,i}\}\}$. This activity is formed by combining two primitive activities $\texttt{h} = \{\{16\},\{17,18\},\{\oslash\}\}$ and $\texttt{i} = \{\{15,17\},\{19\},\{\oslash\}\}$. Since one output argument of `'h'` matches one input argument of `'i'`, therefore the matching argument i.e. 17 is consumed internally, whereas the unmatched output argument of `'h'` (i.e. 18) becomes part of output arguments of `'C'`, and unmatched input argument of `'i'` (i.e. 15) becomes part of input arguments of `'C'`.

A compound activity `'B'`, as shown in Figure 8.2, is defined as $\texttt{B} = \{\{1\},\{11\},\{\texttt{c,d,e,f,g}\}\}$: an activity that is formed based on a workflow pattern described in the following section.

8.4 Applying Patterns for Activity Synthesis

Activity synthesis follows workflow patterns, and the resulting synthesized activity either provides new functionality or gives an aggregated capability of combined activities. We provide a set of synthesis rules that follow data flow pattern for primitive activities as shown in Figure 8.2. These rules are fired iteratively resulting in automatic generation of complex activities. From data flow perspective, both sequential and parallel flows are very important workflow patterns.

8.4.1 Sequential Flow Patterns

This is one of the basic workflow patterns in which two activities are combined with each other such that the output (a set of arguments) of one activity is correlated or mapped to the input of second activity. Formally, if we have two activities $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{A}$ such that

$$\alpha = \{\mathcal{I}_\alpha, \mathcal{O}_\alpha, \mathcal{A}_\alpha\} \quad \beta = \{\mathcal{I}_\beta, \mathcal{O}_\beta, \mathcal{A}_\beta\} \quad \mathcal{O}_\alpha = \mathcal{I}_\beta \\ \Longrightarrow \gamma = \{\mathcal{I}_\alpha, \mathcal{O}_\beta, \{\alpha, \beta\}\}$$

A new synthesized activity γ is generated that pipes output $\mathcal{O}_\alpha$ of activity α to input $\mathcal{I}_\beta$ of activity β. This is depicted in Figure 8.2 as compound activity '`A`' that is formed by synthesis of '`a`' and '`b`', that is

$$\texttt{A} = \{\{5\}, \{7\}, \{\texttt{a,b}\}\}$$

We transform this formal description into ontology concept by introducing following rule:

[`SynthesisRule1` :
(`?a` *input* `?x`)(`?a` *output* `?y1`)
(`?b` *input* `?y2`)(`?b` *output* `?z`)
(`?y1` *rdf:type* `?T`)(`?y2` *rdf:type* `?T`) `makeTemp(?c)`
$\Longrightarrow$ (`?c` *rdf:type* `PipedActivity`)
(`?c` *input* `?x`)(`?c` *output* `?z`)]

According to this rule a new activity c is generated; `makeTemp` is a *built-in* [32] that is used to create a new concept in the underlying ontology. `PipedActivity` is a basic concept defined in foundation ontology and `rdf`, `rdfs` and `owl` represent namespace prefixes which are used as default namespace for RDF [34], RDFS [42] and OWL [146] in the Jena rule-based reasoner. A non-trivial form of sequential flow synthesis is the one in which output $\mathcal{O}_\alpha$ of an activity α matches with input $\mathcal{I}_\beta$ of an other activity β but at least one side always matches partially. In this case, partially matched arguments are consumed internally whereas un-matched arguments become part of input/output of resulting synthesized activity γ. Formally, let $\texttt{D} = \mathcal{O}_\alpha \cap \mathcal{I}_\beta$ then

$$\mathcal{O}_\alpha \cap \mathcal{I}_\beta \neq \oslash \ \wedge \ \mathcal{O}_\alpha \cap \mathcal{I}_\beta \neq \mathcal{O}_\alpha \cup \mathcal{I}_\beta \Longrightarrow$$
$$\gamma = \{\{\mathcal{I}_\alpha + (\mathcal{I}_\beta - \texttt{D})\}, \{\mathcal{O}_\beta + (\mathcal{O}_\alpha - \texttt{D})\}, \{\alpha, \beta\}\}$$

This defines that if the output of an activity α partially matches with the input of an other activity β then both activities can synthesized to form a third activity γ with the input and output arguments as shown above. Such kind of synthesis is very useful for generating complex workflows, the following rule is applied to make it possible:

```
[SynthesisRule2 :
 (?a rdf:type GridActivity)
(?b rdf:type GridActivity)
(?a owl:differentFrom ?b)
partialDataFlow(?a,?b) makeTemp(?c)
⟹ (?c rdf:type SequentialPairedActivity)
     partialDataFlow(?c, ?a, ?b)]
```

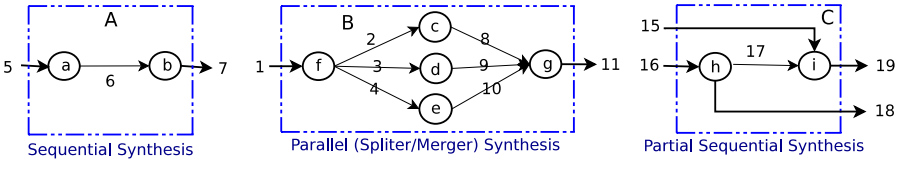


Fig. 8.2. Sequential and parallel flows patterns.

An example of this rule is depicted in Figure 8.2 as a compound activity `'C'` that is formed by synthesis of 'h' and 'i', that is

$$\texttt{C} = \{\{15, 16\}, \{18, 19\}, \{\texttt{h,i}\}\}$$

The `partialDataFlow` is a customized *builtin* implemented in Java and used in Jena library to simplify the reasoning that is otherwise non-trivial to provide as a rule. In the rule body it returns *true* if a and b has partial output and input match whereas in the rule head, it associates input/output arguments to newly created activity c accordingly.

8.4.2 Parallel Flow Patterns

Synthesis of sequential activities is enough for generating complex activities. However, the true essence of the Grid is to execute at least some of the activities in parallel on different nodes so that a speedup or significant improvement in QoS can be achieved. Achieving a speedup is crucial for scientific application. We define rules that result in the creation of complex activities that can execute in parallel, partially or as a whole, for instance, by following the

master/slave pattern. This is a common form of parallel flow in which one activity (say α) splits a task into sub tasks and distributes them among multiple instances of a slave activity (say γ). Then a third activity (say β) collects results from slaves and consolidates them. Since we assume that all activities are autonomous and self-contained therefore the splitter, merger, and slave activities (see activity `'B'` in Figure 8.2) are disjoint activities, that leads to:

$$\alpha = \{\mathcal{I}_\alpha, \mathcal{O}_\alpha, \mathcal{A}_\alpha\} \quad \beta = \{\mathcal{I}_\beta, \mathcal{O}_\beta, \mathcal{A}_\beta\} \quad \gamma = \{\mathcal{I}_\gamma, \mathcal{O}_\gamma, \mathcal{A}_\gamma\}$$
$$\mathcal{O}_\alpha = \mathcal{I}_\gamma \wedge \mathcal{I}_\beta = \mathcal{O}_\gamma \Longrightarrow \delta = \{\mathcal{I}_\alpha, \mathcal{O}_\beta, \{\alpha, \beta, \gamma\}\}$$

which can be translated to ontology rule as:

[`SynthesisRule3` :
(`?a` *output* `?x`)(`?x` *rdfs:subClassOf* `all(argument ?y1)`)
(`?b` *input* `?z`)(`?z` *rdfs:subClassOf* `all(argument ?y2)`)
(`?a` *input* `?ain`)(`?b` *output* `?bout`) (`?c` *input* `?y1`)
(`?c` *output* `?y2`)`makeTemp(?d)`
$\Longrightarrow$ (`?d` *rdf:type* `MasterSlaveActivity`)
(`?d` *splitter* `?a`) (`?d` *merger* `?b`)(`?d` *slave* `?c`)
(`?d` *input* `?ain`) (`?d` *output* `?bout`)]

In another form of parallel flow, any two activities α and β can be connected through a set of slave activities $\mathcal{A}_s$ so that $\mathcal{O}_\alpha$ is connected to the $\mathcal{I}_\beta$ by activities $\in \mathcal{A}_s$.

$$\alpha = \{\mathcal{I}_\alpha, \mathcal{O}_\alpha, \mathcal{A}_\alpha\} \quad \beta = \{\mathcal{I}_\beta, \mathcal{O}_\beta, \mathcal{A}_\beta\} \quad \mathcal{A}_s \in \mathcal{A}$$
$$\mathcal{O}_\alpha = \bigcup_{s \in \mathcal{A}_s} \mathcal{I}_s \ \wedge \ \mathcal{I}_\beta = \bigcup_{s \in \mathcal{A}_s} \mathcal{O}_s$$
$$\Longrightarrow \gamma = \{\mathcal{I}_\alpha, \mathcal{O}_\beta, \{\{\alpha, \beta\} + \mathcal{A}_s\}\}$$

This is a well known workflow pattern. As shown in Figure 8.2 *(parallel flow)*, activities `'f'` and `'g'` work as splitter and merger respectively. Each activity $c_i \in \{c_1, ..., c_n\}$ works as a slave and can be executed in parallel. According to this rule, a new compound activity `'B'` (Figure 8.2) is formed as:

$$\texttt{B} = \{\{1\}, \{11\}, \{\texttt{c,d,e}\}\}$$

If the output of an activity α is a collection of similar arguments ($|\mathcal{O}_\alpha| = 1$) and the input of another activity β is also a collection of similar arguments ($|\mathcal{I}_\beta| = 1$) but different from $\mathcal{O}_\alpha$, then a set $\mathcal{A}_s$ of slave activities is $\mathcal{A}_s = \{s\}$. They are instances of same activity and thus synthesized activity can be composed in a parallel loop, for instance `parallelFor` that is a high-level construct of AGWL. If an activity provider adds some information about activity usage, such as *Parallelizable* and *Iteratable*, then synthesis may lead to repeatable control flow activities.

Figure 8.3 depicts a set of primitive activities $\mathcal{A} = \{a - h\}$, a set of arguments $\mathcal{M} = \{1, ..., 13\}$. All primitive activities are independently defined. The flow of arguments is formed after synthesis. First of all primitive activities are combined in pairs (shown as shaded rectangles) and then more complex activities are formed (shown as rectangles). Synthesized activities are labeled as $\{A - H\}$. The most complex activity is I that transforms argument 1 into argument 13. After the activity synthesis, compound activities are formed as follows:

$$
\begin{aligned}
A &= \{\{1\}, \{5,6,7\}, \{a, E\}\} &&= \{\{1\}, \{5,6,7\}, \{a, b, c\}\} \\
B &= \{\{2,3\}, \{9,10\}, \{E, F\}\} &&= \{\{2,3\}, \{9,10\}, \{b, c, d, f\}\} \\
C &= \{\{5,6,7\}, \{13\}, \{D, h\}\} &&= \{\{5,6,7\}, \{13\}, \{d, e, f, g, h\}\} \\
D &= \{\{5,6,7\}, \{12\}, \{F, G\}\} &&= \{\{5,6,7\}, \{12\}, \{d, e, f, g\}\} \\
E &= \{\{2,3\}, \{5,6,7\}, \{b, c\}\} &&= \{\{2,3\}, \{5,6,7\}, \{b, c\}\} \\
F &= \{\{5,7\}, \{9,10\}, \{d, f\}\} &&= \{\{5,7\}, \{9,10\}, \{d, f\}\} \\
G &= \{\{6,9,10\}, \{12\}, \{e, g\}\} &&= \{\{1\}, \{13\}, \{e, g\}\} \\
H &= \{\{9,10,11\}, \{13\}, \{g, h\}\} &&= \{\{9,10,11\}, \{13\}, \{g, h\}\} \\
I &= \{\{1\}, \{13\}, \{A, C\}\} &&= \{\{1\}, \{13\}, \{a, b, c, d, e, f, g, h\}\}
\end{aligned}
$$

This shows that the activity synthesis significantly increases number of activities with different capabilities. The increase in activity search space may result in query hits that otherwise may not be possible.

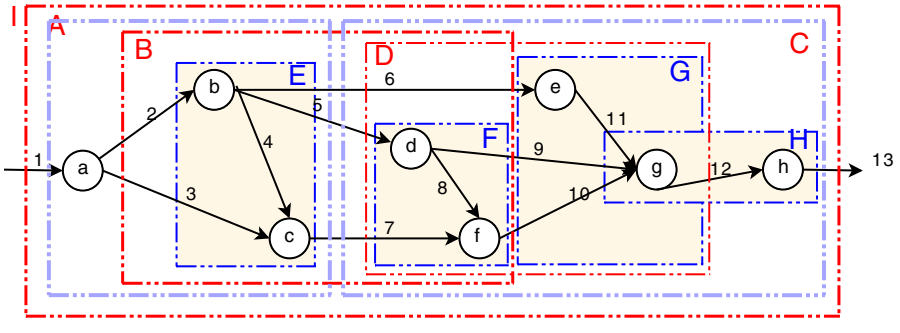


Fig. 8.3. A Sample Synthesis of activities.

8.5 On-Demand Provisioning

The provisioning is a way of matching user goals to available activities, especially complex activities or workflows. A user can specify its available input optionally accompanied by some quality of service parameters. We propose to use SPARQL [182] query language for defining goals as it is a proposed standard query language for ontologies and well supported by various tools and APIs, and can be used to define constraints along with rules. A goal in the form of SPARQL query can take the following form:

```
SELECT ?a
WHERE {?a rdf:type MovieRenderer .
    ?a outout ?out .
    ?out rdf:type MPEGFile .
}
```

In order to address multiple constraints while generating complex activities, we introduce constraints in ontology rules. For intstance, a complex activity should be generated only if average *reliability* (a QoS parameter) of primitive activities is at least 0.5%. This is however, not trivial with current rule language [183]. To overcome this limitation, we propose to use *built-ins* in rules.

8.5.1 Built-Ins and Constraints

Provision of *Built-in* constructs in a rule language is a modular approach of adding new entailment that otherwise is not possible with current reasoners. A set of *built-ins* are available for Jena that can be executed on underlying ontology model and knowledge-base. This set includes built-ins for mathematical calculations, comparisons, boolean evaluations, string and collection manipulations, etc. A set of custom *built-ins* is introduced that augments process of rule-based entailment. Customized *built-ins* can also be provided such as the following rule that generates an activity if and only if the average reliability of primitive activities is higher than 0.5.

```
[SynthesisRuleReliableActivity :
   (?a input ?x)(?a output ?y)(?b input ?y)
  (?b output ?z)(?a reliability ?i) (?b reliability ?j)
  average(?i,?j,?av) greaterThan(?av,0.5)
  makeTemp(?c)
⟹ (?c rdf:type ReliableActivity)
      (?c input ?x)(?c output ?j)
      (?c reliability ?avg)]
```

Similarly, user goals may also contain constraints in the form of filters as part of SPARQL query.

```
SELECT ?a
WHERE {?a rdf:type MovieRender  .
    ?a outout ?out  .
    ?out rdf:type MPEGFile  .
    ?a reliability ?reliability  .
    ?a animation ?animation  .
    ?animation rdf:type CyclicAnimation  .
    FILTER (?reliability >= 0.5)
}
```

A custom built-in called *agwl* is provided. Once user goals are mapped to at least one possible candidate activity then *agwl* is fired that transforms a matching synthesized activity into an AGWL document as a side effect.

The built-in *rpdp* is a reservation policy decision point that optionally works as a part of *assumptions* and interact with Askalon reservation service for user authorization to check if it has advance reservation. The advance reservation mechanism [156] is presented in detail in Chapter 5.

8.5.2 Assumptions and Effects

As described in Section 7.4, an activity or application provider may have some assumption about the user or a user may assume something about providers. Similarly, there could be some *after-effects* once an activity is executed. An assumption and effects described in primitive activities are aggregated in compound activities. The assumptions need to be true before activity execution and effects may be exercised after activity execution. For instance, an activity provider may assume that a user has advance reservation for activity execution on a certain Grid node and user account is charged as an after-effect.

[`SynthesisRuleResAssumption` :
... ...
(`?a` *rdf:type* `AGridActivity`)
(`?a` *hasAssumption* `?r`) (`?r` *rdf:type* `Reservation`)
(`?r` *owner* `?o`) `isCaller(?o)` $\Longrightarrow$ ...]

This rule defines an assumption that the client can initiate an action for an activity only if it has advance reservation. Similarly, the following rule defines an effect that a client account is charged if it has already executed a reserved activity.

[`SynthesisRuleChargingEffect` :
... ...
(`?u` *rdf:type* `sg:AGridUser`)
(`?u` *hasExecuted* `?a`) (`?a` *hasAssumption* `?r`)
(`?r` *rdf:type* `Reservation`) (`?r` *owner* `?o`)
$\Longrightarrow$ `charge(?o,?r)`]

Rules can be included in an ontology model as a file or URL. This enables application providers to update them dynamically as well as remotely.

8.6 Improving Capacity Planning

Beside improvement in on-demand provisioning and synthesis of activities to automatic generation of workflows, this chapter also introduces the use of ontologies in improving capacity planning. Chapter 6 introduces a new algorithm, called VSHSH (Section 6.3.1), for capacity planning and resource allocation with improved resources utilization. We also demonstrated that

the VSHSH improves *utility* (Definition 18) of both requester and provider of the Grid resources (Section 6.4). The utility is derived by aggregating the distances between ideal and real values of all QoS constraints.

The details of the distance formula, shown in Equation 6.1, is given in Section 6.2 as well as in [156]. This distance formula is inappropriate for lexicographical values. In order to deal with lexicographical values, a new approach is proposed based on semantics. This approach uses an ontology-based hierarchical structure of all possible values of a constraint in order to evaluate distance or deviation of offered value from the ideal or requested value.

8.7 Discussion and Experiments

We designed our foundation ontology using the Protege-OWL [177] editor, and implemented prototype of the proposed activity synthesis mechanism using Jena APIs [32]. Jena provides Java APIs for dynamically creating new ontological concepts, populating knowledge base, and reasoning based on ontology rules. Based on the foundation ontology, an activity (application) provider can add new concepts and provides semantic description of their resources. This can be done either using Protege ontology editor or our simplified configuration description mechanism as described in Section 7.2.1.

Workflow Generation: In order to demonstrate an automatic workflow generation with synthesis of activities, we independently register three tools as primitive activities as described below in our triple notation i.e. $\alpha = \{\mathcal{I}_\alpha, \mathcal{O}_\alpha, \mathcal{A}_\alpha\}$:

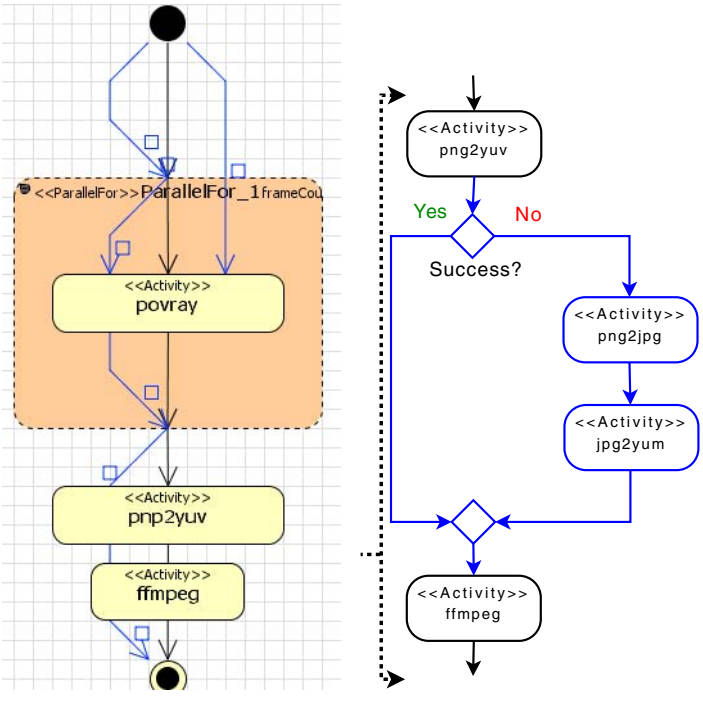
```
povray = { {InitializationFile,ArgumentFile,
 SceneDescriptionFile,StartFrameInt,
 FramesCountInt,TotalFramesInt}
 {PNGArchiveFile}{⊘} }

png2yuv = { {PNGArchiveFile}{YUVStream}{⊘} }

ffmpeg = { {YUVStream}{MPEGFile}{⊘} }
```

An ontology generator component reads the descriptions, generates ontologies and registers in the knowledgebase. All these concepts including activities and argument are defined as distinguished OWL concepts. After applying the synthesis rules and making a query with a goal such as:

```
SELECT ?a
WHERE {?a input ?x .
    ?a outout ?y .
    ?x rdf:type SceneDescriptionFile .
    ?y rdf:type MPEGFile .
}
```



(a) Generated AGWL (b) Control Flow

Fig. 8.4. POVray Workflow

we generate an *agwl* document that can be visualized in Askalon graphical workflow composition tool. The generated workflow is shown in Figure 8.4(a).

Provisioning Improvement: An interesting aspect of synthesis is that it generates more options for provisioner to select or negotiate for. For instance, consider three activities α, β and, γ as described below:

$$\alpha = \{\mathcal{I}_x, \mathcal{O}_y, \mathcal{A}_\alpha\}\ \beta = \{\mathcal{I}_y, \mathcal{O}_z, \mathcal{A}_\beta\}\ \gamma = \{\mathcal{I}_x, \mathcal{O}_z, \mathcal{A}_\gamma\}$$

By applying synthesis rules, beside others, a complex activity δ is generated where $\delta = \{\mathcal{I}_x, \mathcal{O}_z, \{\alpha, \beta\}\}$ is a combination of first two activities α and β since $\mathcal{O}_\alpha = \mathcal{I}_\beta$. As the synthesized activity δ is same as γ the provisioner gets two options to offer instead of just one as it was the case before synthesis. Furthermore, in case if there is no deployment of γ in the Grid, the provisioner may select/offer δ as an alternative option or vice versa, i.e. γ as a replacement of δ.

By following this approach, activity *png2yum* in Figure 8.4(a) can be replaced with following two activities:

```
 png2jpg={{PNGArchiveFile}{JPGArchiveFile}{∅}}
jpg2yum={{JPGArchiveFile}{YUVStream}{∅}  }
```

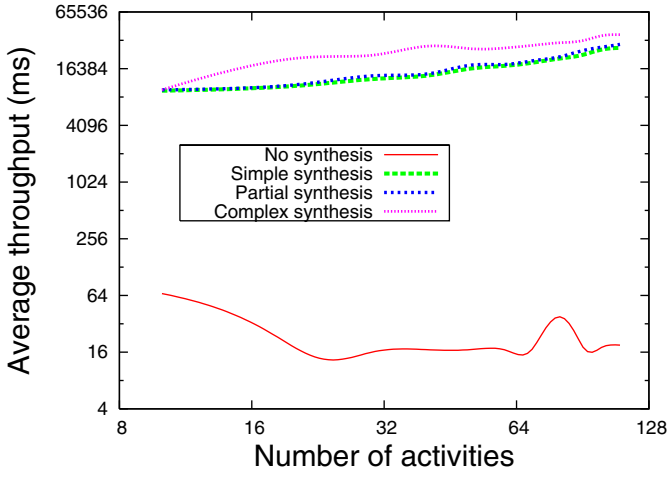


Fig. 8.5. Average response time per transaction (throughput).

Similar to *png2yuv* both *png2jpg* and *jpg2yum* transform *PNGArchiveFile* to the *YUVStream*(see Figure 8.4(b)). This may lead to the generation of a complex workflow with an alternative control-flow as shown in Figure 8.4(b), that means if *png2yuv* does not succeed then control moved towards *png2jpg* followed by *jpg2yum*.

Throughput: Figure 8.5 compares average throughput of different types of queries for available activities.

- No Synthesis: a query for a primitive activity without applying any rule for synthesis and reasoner involvement;
- Simple Synthesis: a query for a compound or synthesized activity generated by synthesizing with a simple combination of primitive activities with single input and output;
- Partial Synthesis: a query for a compound activity generated after complex combination of multiple $(2-4)$ activities with partially matching $(2-6)$ arguments;
- Complex Synthesis: a query for a compound activity generated after complex combination of multiple $(5-14)$ activities with partially matching (≥ 6) arguments.

It is depicted that overhead of queries for primitive activities (no synthesis) is very small. However, the overhead of queries for synthesized activities is significantly high because of the rule based reasoning. Simple synthesis is slightly better than partial synthesis and much better than complex synthesis.

Overhead: Figure 8.6 compares average overhead of synthesis rules when applied dynamically with the query. The overhead is about $10-30s$ for varying number of activities. This is due to the fact that rules are fired iteratively

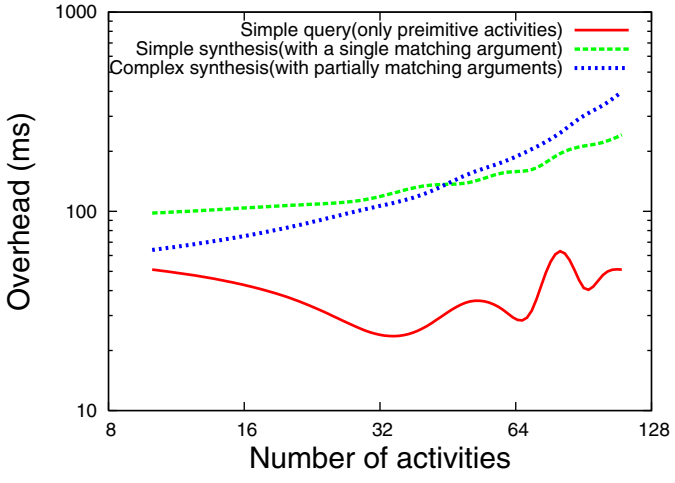


Fig. 8.6. Synthesis overhead.

for new entailment and process stops only when further entailment is not possible. Overhead of complex combinations is slightly higher than simple combinations. However, for a few hundred activities it is just under 30*s* that is rather negligible compared with manual composition time.

Figure 8.7 and 8.8 demonstrate improvement in resource and application utilities respectively. As described in Section 8.6, we applied semantics to our distance formula shown in Equation 6.2. Semantics significantly improves resource utility without compromising the client or application utility and thus

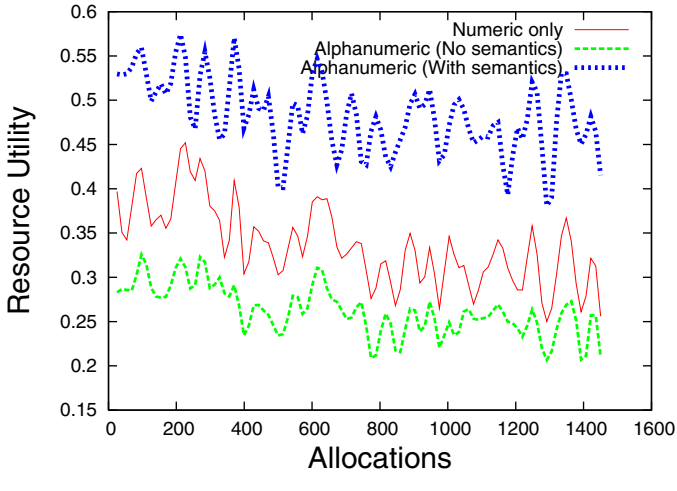


Fig. 8.7. Resource utility improvement with semantics.

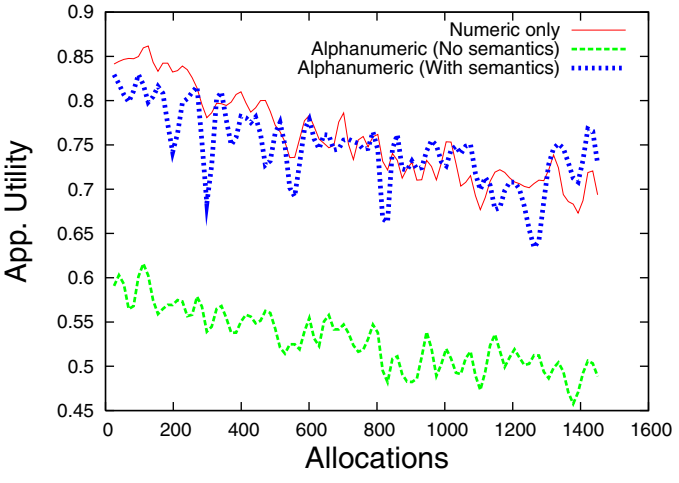


Fig. 8.8. Application utility

results in better capacity planning. See Chapter 6 for details about capacity planning, resource and application utilities, and optimization of resource allocations.

Firing rules: The introduction of a new rule in an ontology starts an iterative entailment process that stops when new entailment become impossible. This is called forward chaining and is considered an inefficient approach. However, iteration is necessary for determining interdependency of multiple rules and thus generating all possible combination of primitive and new activities. This results in faster subsequent queries as facts in the knowledge base are not required to be recalculated. Thus, user goals are efficiently mapped to an already generated workflow. A backward chaining is also possible in which only required entailment are generated. This is an efficient entailment process but the rules are fired for each new instance. This slowed down query response time.

8.8 Related Work

Synthesis and decomposition of processes in organizations has been addressed by various researchers, for instance, a work in [17] examines the synthesis and decomposition of processes in an organization by developing three metaphoric concepts such as full connectivity, independence, and redundancy and then deriving union and intersection of processes based on these concepts.

The work in [172] introduces synthesis of cost optimal workflow structure and identifies the mixed programming model with P-graph [173] based network algorithm.

Numerous researchers are working in the area of semantics Grid [46, 36, 86]. However, most of the work has focused on the problem of physical Grid resource matching [170]. The work in [23] provides a mechanism to map a small subset of Unicore resource descriptions with GLUE schema using ontologies.

ActOn (Active Ontology) [86] is an ontology-based information integration approach for EGEE that can be used to generate and maintain up-to-date metadata for a large scale distributed system.

Web service composition is similar to Grid workflow composition and synthesis. In the area of Semantic Web, a lot of conceptual work has been done but it does not focus on legacy applications. One such example is *WSMO* [45] that provides an ontology-based description mechanism only for Web services. The work described in [114] applies planning to the problem of web service composition.

In the area of Semantic Grid, very few researchers has been working on automatic composition of workflows. The work in [194] addresses problem of machine-assisted composition of workflows for E-Science data using deductive synthesis. The role of planning is discussed in [193] for Pegasus. The authors propose a workflow planner that uses heuristic control rules and searches a number of alternative pre-built complete plans in order to find better quality plan.

A semi-automated approach is introduced in [161] for knowledge evolution of ontology schemas that is applied to the process of new Grid service registration. In K-WfGrid project, knowledge evolution supporting automatic workflow composition has been studied using ontology alignment methods and Petri-net based Grid workflows [161].

In contrast, this book introduces a semantic-based on-demand synthesis of Grid activities to form complex activities. This leads to on-demand generation of abstract workflow applications that can be concretized automatically at runtime. The synthesized activities provide an entirely new capability, an alternative option of an existing activity and/or an aggregated capability of its basic building blocks. Furthermore, our approach exploits semantic-based synthesis of activities that improves provisioning and brokerage capability of resource management for the Grid.

8.9 Summary

In this chapter we formalized the problem of the Grid activity synthesis. Grid activities are software components (executables/services) which are represented in terms of abstract and concrete descriptions. We propose a rule-based synthesis mechanism that can be used to combine multiple activities to form compound activities. A set of custom *built-ins* is introduced that augments process of rule-based entailment. The synthesized activities either provide a new or an aggregated functionality of combined activities, or alternative options with different quality of service. This synthesis of activities leads to

automatic generation of Grid workflows that can be offered on-demand and mapped to the Grid dynamically at runtime.

Furthermore, the synthesis generates more (compound) activities thus increases activity search space and result in improved negotiability of a provisioner. That means a provisioner gets more options to offer than it originally may have.

We demonstrated the effectiveness of the synthesis with examples and experiments. Furthermore, role of semantics in improving capacity planning with optimization in resource allocations and better utilization of resource capabilities is presented.

9

Conclusion

In this monograph, we have addressed various research challenges: we have identified different research issues, proposed novel solutions and introduced new approaches in order to develop a resource management system for the Grid. The proposed system is implemented based on state-of-the-art Grid and Web technologies and deployed in a real world Grid infrastructure. We have demonstrated applicability and effectiveness of the new system through experiments. This chapter concludes this book by highlighting major contributions and future directions.

The contributions includes our research findings and development of new mechanisms that can be used by researchers working in the area of Grid computing in general and Grid resource management in particular. Following subsections describe our major contributions.

9.1 Resource Management Model

In this book we provide formal descriptions of various aspects of *the Grid* and *resource management* as part of the Grid middleware (Chapter 2).

- We have rendered the boundaries of various components of the Grid middleware covering both operating and runtime environments. These components include a Grid scheduler (Section 2.4.3), an enactor/executor (Section 2.4.4), and a resource manager (Section 2.6). The Grid scheduler is separated from resource manager in our proposed model. The scheduler focuses on execution planning and job control whereas resource manager focuses on on-demand resource provisioning.
- We have further identified characteristics of a resource manager (Section 2.6) that includes mechanisms such as provisioning, brokerage, deployments, activity registrations, activity synthesis, capacity planning and management.

M. Siddiqui and T. Fahringer: Grid Resource Management, LNCS 5951, pp. 201–206, 2010.

9.2 Towards Automatic Resource Management

We have designed and developed a new resource management system with automatic resource selection and brokerage. A client can specify its goals in the form of resource requests and the resource manager performs automatic resource brokerage and generates multiple options with matching resources. The main features of the newly developed resource management system are:

- *GridARM* (Section 3.3) is a distributed infrastructure in which services are loosely coupled and coordinate in a service-oriented fashion. Multiple instances of the resource manager can run autonomously while managing different sets of Grid resources. These resource managers can cooperate with each other in order to share expensive and/or idle resources;
- a flexible resource discovery mechanism (Section 3.3.1) in which multiple information services can be registered from which different information types can be retrieved and consolidated in a widely used GLUE schema format.
- a *candidate set generation* process is introduced (Section 3.3.2) in which matching of discovered resources with user goals takes place; a user can specify a selection criteria as part of the resource request and the ordered list of generated set of candidates is offered;
- a matching resource of highest rank can be selected automatically, where the rank is calculated according to default or user's specified selection criteria;
- in case of multiple contending applications or clients, the Grid resources are allocated according to the proportional share of resources in the Grid;
- the resource manager is customizable, it can be used to setup a dedicated experimental environment and a coarse-grained access control policy for the Grid resources;
- the resource management system is developed as a custodian of resource providers, it negotiates with clients to lease resources according to their proportional share in the Grid. As a result, a resource with more capacity gets more allocations without being unfair with low-share resources. This improves fairness and protects different sets of resources for different classes of users;
- *GridARM* is developed as a distributed scalable infrastructure based on the superpeer model (Section 3.4.2) that is a self-managing and fault-tolerant model. A new superpeer is re-elected automatically if existing superpeer stops working;
- we have evaluated proportional share-based load distribution and found that our approach is better as standard deviation of actual allocations.

9.3 Dynamic Registration and Automatic Deployment

In contrast to most of the existing resource management systems for the Grid, we have designed and developed a framework (Chapter 4) that covers logical resources such as *activities* (Definition 6), *applications* (Definition 10), and *service-level agreement*s (Definition 34). It provides dynamic registration, automatic deployment, and on-demand provisioning of the Grid activities that can be used in building Grid-enabled complex workflow applications. This framework, called *GLARE*, is implemented as an extension of *GridARM* with the following functionalities:

- *GLARE* separates activities in abstract and concrete descriptions known as *activity types* (Definition 7) and *activity deployments* (Definition 8) respectively. It clearly describes both representations of activities so that they can be advertised unambiguously and located automatically. *Activity types* are mapped to *activity deployments* and delivered on-demand;
- the distinguished feature of *GLARE* is that it provides automatic deployment of applications and activities on a manually or automatically selected node (Section 4.4.1). The deployment procedure is needed to be associated with the activity types by the activity providers. The procedure is executed automatically on the target node by *GLARE*;
- it enables an activity or application provider to perform undeployment of applications once they are utilized and no longer required;
- *GLARE* proposes a flexible mechanism to use different ways to perform automatic deployments. This includes expect-based deployment, GRAM-based deployment etc;
- *GLARE* enables application providers to perform registration and unregistration of deployed applications in order to control their visibility for different timeframes or for different users;
- we compared our superpeer model-based distributed framework with existing hierarchical Globus WS-MDS. We have found that our system is better in performance and is capable to handle more more registrations of activities.

9.4 Negotiation for Service-Level Agreement (SLA)

We have designed and developed a mechanism for allocation of Grid resources with negotiation-based advance reservation (Section 5.3) and a practical solution for agreement enforcement (Section 5.4.3). This mechanism enables a client to negotiate for required resources and to make an agreement with a better compromise between application requirements and resource capabilities. Major contributions in this context are given below:

- a flexible negotiation mechanisms has been developed by introducing different allocation offer generation algorithms to support different types of allocation strategies including fairsharing and optimization in resource utilization;
- a 3-layered protocol introduced for negotiation between the resource *requesters* and *providers* in order to reach and seal an agreement;
- a practical solution, based on the off-the-shelf Grid technologies, is provided for the enforcement of an agreement;
- mechanism for open reservations is introduced in order to deal with the dynamic Grid environment. This mechanism represents a priority provision, that is, a promise for allocation is made in advance but actual allocation of the resource deferred until runtime. The next available resource that fulfills user goals is allocated at runtime;
- we have demonstrated that advance reservation can have a major impact on execution time and can considerably increase predictability of the Grid environment.

9.5 Multi-Constrained Optimization and Capacity Planning

Beside negotiation-based advance reservation, we have introduced a mechanism for capacity management and planning that is developed based on advance reservations. The idea of capacity planning is new in the Grid, and it can be used in improving resource utilization while addressing concerns about under-utilization of Grid resources and reduction in quality of service (QoS). The proposed Grid capacity planning and management is performed with the help of advance reservation and multi-constrained allocation optimization. We have introduced a new algorithm called VSHSH (Algorithm 8) for optimized utilization of resources. The new algorithm models resource allocation as an on-line strip packing problem and provides a new mechanism that optimizes resource utilization and other QoS parameters while generating contention-free solutions. Major contributions of *GridARM* from the perspective of multi-constrained optimization and capacity planning are as follows:

- it provides a forward looking process in which allocations are made along a planning or time horizon;
- it exploits advance reservation for optimized resource allocations with service-level agreement (SLA). In this way, it proves usefulness of advance reservation and multi-constrained optimization;
- it provides a mechanism to plug-in different allocation offer generation algorithms that can be used flexibly according to the policy of the resource providers;

- an algorithm generates multiple options to be offered to the client. The options are generated as alternative offers based on multi-constrained optimization process for resource utilization;
- the allocation offers are generated in such a way that resource capacity is optimally utilized and capacity wastage is minimized. We have demonstrated the usefulness of our approach for capacity planning and management.

9.6 Semantics in the Grid

We have introduced an ontology-based resource description, discovery and selection mechanism. As a resource description model, we have proposed to replace the classic attribute-based symmetrical resource description model with an extensible ontology-based asymmetric model. This model provides foundation to a flexible and extensible discovery and correlation mechanism.

Furthermore, we have introduced a new mechanism for automatic synthesis of resources and software components by applying ontology rules. Rule-based synthesis combines multiple primitive resources to form new compound resources. Here are some of the main steps we have taken towards the Semantic Grid:

- we have introduced an asymmetric resource description mechanism with ontologies so that resource requesters and providers can flexibly describe their resources without having to agree on certain terms and their agreed upon values;
- we have proposed to exploit the subsumption of ontological concepts for resource selection that allows to propose alternative options if exact match does not exist;
- in contrast to the current trends in which semantics is used for matchmaking, we exploit semantics to synthesize description of Grid resources in order to generate multiple compound resources that can be provisioned as new or alternative options. The newly generated synthesized resources could provide aggregated capabilities that otherwise may not be possible;
- the synthesis has enabled the automatic generation of Grid workflow applications, and we have demonstrated it.

To summarize, we have designed and developed a new, coherent and consistent system which covers the most important aspects of Grid resource management, and demonstrated its usefulness for running real scientific applications on the Grid. *GridARM*, our Grid Resource Management system described in this book is a key component of the Askalon Grid runtime system (Section 2.4).

9.7 Future Research

We have addressed several research challenges in the context of resource management for the Grid. However, we believe it is a starting point and the research in this direction has a long way to go. We identify the following potential research directions that are either currently being considered or will be considered for future research:

- Physical resources possess great research potential for the lifecycle management with the help of virtualization technologies such as virtual machine systems, virtual LAN, and other state-of-the-art computing and network technologies.
- Porting of different kinds of legacy scientific and business application need to be addressed with more seriousness by applying different wrapping and integration patterns.
- We believe that configuration management of both physical and logical resources can be automatized. We intend to examine various possibilities and challenges in this dimension.
- The role of semantics in the Grid needs to be extended to cover not only resource descriptions but also configurations, policies, and agreement documents.
- Possible security loopholes, while performing automatic deployments of applications on nodes, need to be explored and the challenge of possible security threats is also a great topic of research.
- Fine-grained optimization techniques are to be studies for compilation of workflow applications in order to cement the process of automatic deployments.
- The allocation strategies and capacity planning with multiple planning horizons by considering more QoS parameters are to be examined.
- More complex ontology rules can be applied to generate more complex workflows with different types of control and data flows. Furthermore, the analysis of possible improvement in query-hits with resource synthesis can be done.
- A Grid portal technology with easy to use interfaces can be introduced for Web-based decentralized distributed management of the Grid.

A

Notations

Symbol	Description
$\mathcal{G}$	The Grid: a set of Grid nodes or Grid sites
$\mathcal{G}^c$	The Computational Grid
$\mathcal{G}^d$	The Data Grid
$\mathcal{G}^k$	The Knowledge Grid
$\mathcal{R}$	A set of Grid resources
$\mathcal{Q}$	A set of resource requests
$\mathcal{PR}$	A set of physical resources$\subseteq \mathcal{R}$
$\mathcal{LR}$	A set of logical resources$\subseteq \mathcal{R}$
$\mathcal{A}$	A set of Grid activities $\subseteq \mathcal{LR}$
$\mathcal{E}$	A set of *activity types*
$\mathcal{D}$	A set of *activity deployments*
$\mathtt{at}$	An *activity type* of an activity
$\mathtt{at}^{'}$	A generalized *activity type* of an activity
$\mathcal{D}_{\mathtt{a}}$	An *activity deployment* of an activity $\mathtt{a}$
$\mathcal{IS}$	A set of Grid Information Service
$\mathcal{IT}$	A set of information types associated with Grid resources
$\mathcal{T}$	A set of onstraints (terms and conditions)
$\mathcal{C}$	A set of clients
$\mathcal{M}$	A set of arguments
$\mathcal{V}$	Edges (Dependancies)
$\mapsto$	Function mapping
$\mathcal{P}$	A set of processors
$\mathcal{Y}$	Allocation function
$\mathcal{W}$	A workflow application $\in \mathtt{W}$
$\mathtt{W}$	A set of workflow applications
$\mathcal{B}$	A set of candidates
$\mathcal{S}$	A set of selections
$\mathcal{L}$	A set of allocations
$\mathcal{CL}$	A set of co-allocations

Symbol	Description
$\mathcal{I}$	A set of input arguments $\subseteq \mathcal{M}$
$\mathcal{O}$	A set of output arguments $\subseteq \mathcal{M}$
$\mathcal{U}$	Utilities and utility function
$\texttt{ref}(\texttt{r})$	A reference of a resource $\texttt{r}$
$\oslash$	Empty set
$\texttt{tcpu}(\texttt{g})$	Total CPUs of a node $\texttt{g} \in \mathcal{G}$
$\texttt{fcpu}(\texttt{g})$	Free CPUs of a node $\texttt{g} \in \mathcal{G}$
$\texttt{clock}(\texttt{g})$	Processor clock speed of a node $\texttt{g} \in \mathcal{G}$
$\texttt{rank}(\texttt{g})$	Rank of a node $\texttt{g} \in \mathcal{G}$
$\texttt{uptime}(\texttt{g})$	System *uptime* of node $\texttt{g} \in \mathcal{G}$
ς	*Candidate set generator* function
ξ	Resource selection function
$\top$	Maximized
$\bot$	Minimized
$\texttt{capacity}$	Capacity of a resource
$\texttt{duration}$	Time duration or interval between to points in time
Υ	Allocation function
δ	Discovery function
$\texttt{lookup}$	lookup function
$\texttt{agr}$	An agreement document
$\mathcal{OF}$	A set of offers
$\texttt{T}$	time dimension
$\texttt{dist}$	Distance function
ξ	Section function
$\texttt{startt}$	Start time
$\texttt{endt}$	End time
$\equiv$	Equivalent
$\mathbb{N}$	Set of natural numbers
$\mathbb{N}^*$	Set of positive natural numbers (non-zero)
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of positive real numbers
$\Longleftrightarrow$	If and only if (iff)
$\Longrightarrow$	Implication
$\forall$	For all
$\exists$	Exists
$\mid$	Set restriction
$\in$	Set membership
$\lvert S \rvert$	Cardinality of set S
$\mathcal{P}(S)$	Power set of S
$\times$	Cross product
$\wedge$	Logical conjunction
$\vee$	Logical disjunction
$\cup$	Set union
$\cap$	Set intersection

Symbol	Description
$\setminus$	Set difference
$\subset$	Subset of
$\prec$	Totally ordered set precedence
$\texttt{pred}(\texttt{a})$	Predecessor of a workflow activity `a`
$\texttt{succ}(\texttt{a})$	Successor of a workflow activity `a`
$\texttt{pred}^p(\texttt{a})$	Predecessor of rank p a workflow activity `a`
$\texttt{succ}^p(\texttt{a})$	Successor of rank p a workflow activity `a`
$\texttt{sched}$	Schedule function
$\sqsubseteq$	Sub concept or equivalent concept
$\sqsupseteq$	Super concept or equivalent concept
$\sum$	Summation
$\square$	End of an example, algorithm or proof

References

1. http://sourceforge.net/projects/smartfrog/
2. Adaptive Services Grid (ASG) project, http://asg-platform.org
3. Fathers of grid computing form start-up, http://articles.techrepublic.com.com/5100-22_11-5488342.html
4. Kepler project, http://kepler-project.org
5. The knowledge-based workflow system for grid applications (K-Wf Grid), http://www.kwfgrid.eu
6. LIGO Data Grid, http://www.lsc-group.phys.uwm.edu/lscdatagrid
7. Onto Grid, http://www.ontogrid.net
8. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of grid resource management systems for distributed computing. Softw. Pract. Exper. 32(2), 135–164 (2002)
9. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S.: Data management and transfer in high-performance computational grid environments. Parallel Computing 28(5), 749–771 (2002)
10. The Globus Alliance. Globus Toolkit, http://www.globus.org/toolkit
11. Altair. Portable Batch System (PBS) Professional 7.1, http://www.altair.com/software/pbspro.htm
12. Amdahl, G.M.: Validity of the single processor approachtoachieving large scale computer capabilities. In: AFIPS joint computer conference, vol. 30, pp. 483–485 (1967)
13. P. Anderson, A. Scobie: LCFG: The Next Generation. In: UKUUG Winter Conference. UKUUG, Bristol, July 4-7 (2002)
14. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Systems, S., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services (BPEL4WS). Specification version 1.1, Microsoft, BEA, and IBM (May 2003)
15. Augerat, P., Billot, W., Derr, S., Martin, C.: A scalable file distribution and operating system installation toolkit for clusters. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, May 21-24 2002, IEEE Computer Society Press, Los Alamitos (2002)
16. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook. Cambridge University Press, Cambridge (2003)

17. Basu, A., Blanning, R.W.: Synthesis and decomposition of processes in organizations. Info. Sys. Research 14(4), 337–355 (2003), doi:10.1287/isre.14.4.337.24901
18. Bays, C.: A comparison of next-fit, first-fit, and best-fit. Commun. ACM 20(3), 191–192 (1977), doi:10.1145/359436.359453
19. Berman, F., Wolski, R., Casanova, H., Cirne, W., Faerman, H.D.M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., Zagorodnov, D.: Adaptive Computing on the Grid Using AppLeS. IEEE Transactions on Parallel and Distributed Systems 14(4), 369–382 (2003)
20. BIRN: Biomedical informatics research network, http://www.nbirn.net
21. Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D., Luitz, J.: WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties. Institute of Physical and Theoretical Chemistry, Vienna University of Technology, Vienna (2001)
22. Braverman, A.M.: Father of the grid: Computer scientist ian foster has developed the software to take shared computing to a global level. University of Chigago Magzine 96(4) (2004)
23. Brooke, J., Fellows, D., Garwood, K., Goble, C.: Semantic matching of grid resource descriptions. In: Dikaiakos, M.D. (ed.) AxGrids 2004. LNCS, vol. 3165, pp. 240–249. Springer, Heidelberg (2004)
24. Bucur, A.I.D., Epema, D.H.J.: The maximal utilization of processor coallocation in multicluster systems. In: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, p. 60. IEEE Computer Society Press, Los Alamitos (2003)
25. Buyya, R., Abramson, D., Giddy, J.: NimrodG: An architecture for a resource management and scheduling system in a global computational grid. In: HPC ASIA 2000, China, IEEE Computer Society Press, Los Alamitos (2000)
26. Buyya, R.: Economic-based Distributed Resource Management and Scheduling for Grid Computing. PhD Thesis, p. 180. Monash University, Melbourne, Australia (2002)
27. Buyya, R., Venugopal, S.: The gridbus toolkit for service oriented grid and utility computing: An overview and status report. In: Proceedings of the First IEEE International Workshop on Grid Economics and Business Models, New Jersey, USA, April 23, 2004, pp. 19–36. IEEE Computer Society Press, Los Alamitos (2004)
28. CalTech: Montage: An astronomical image mosaic engine, http://montage.ipac.caltech.edu
29. Cannataro, M., Congiusta, A., Pugliese, A., Talia, D., Trunfio, P.: Distributed data mining on grids: services, tools, and applications. IEEE Transactions on Systems, Man, and Cybernetics: Part B 34(6), 2451–2465 (2004), http://dx.doi.org/10.1109/TSMCB.2004.836890
30. Cannataro, M., Talia, D.: The knowledge grid. Communications of the ACM 46(1), 89–93 (2003), http://doi.acm.org/10.1145/602421.602425
31. Global Grid Forum: CDDLM-wg. CDDLM – Configuration, Deployment and Lifecycle Management of grid services, http://forge.gridforum.org/projects/cddlm-wg
32. Hewlett-Packard Development Company: Jena: A Semantic Web Framework for Java, http://jena.sourceforge.net
33. The Austrian Grid Consortium: Austrian Grid, http://www.austriangrid.at

34. World Wide Web Consortium: Resource Description Framework (1998), http://www.w3.org/TR/WD-rdf-syntax
35. Cooper, K., Dasgupta, A., Kennedy, K., Koelbel, C., Mandal, A., Marin, G., Mazina, M., Mellor-Crummey, J., Berman, F., Casanova, H., Chien, A., Dail, H., Liu, X., Olugbile, A., Sievert, O., Xia, H., Johnsson, L., Liu, B., Patel, M., Reed, D., Deng, W., Mendes, C., Shi, Z., YarKhan, A., Dongarra, J.: New Grid Scheduling and Rescheduling Methods in the GrADS Project. In: International Parallel and Distributed Processing Symposium, Workshop for Next Generation Software, April 2004, IEEE Computer Society Press, Los Alamitos (2004)
36. Corcho, O., Alper, P., Kotsiopoulos, I., Missier, P., Bechhofer, S., Goble, C.: An overview of S-OGSA: A reference semantic grid architecture. Web Semantics: Science, Services and Agents on the World Wide Web 4, 102–115 (2006)
37. CrossGrid: http://www.crossgrid.org/
38. Csirik, J., Woeginger, G.J.: Shelf algorithms for on-line strip packing. Inf. Process. Lett. 63(4), 171–175 (1997)
39. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman., C.: Grid Information Services for Distributed Resource Sharing.. In: Tenth IEEE International Symposium on High- Performance Distributed Computing (HPDC-10), August 2001, p. 181. IEEE Computer Society Press, Los Alamitos (2001)
40. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland (July 2002)
41. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S.: A Resource Management Architecture for Metacomputing Systems. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1998, SPDP-WS 1998, and JSSPP 1998. LNCS, vol. 1459, pp. 62–82. Springer, Heidelberg (1998)
42. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0. w3c proposed recommendation (December 2003), http://www.w3.org/TR/rdf-schema/
43. De Roure, D., Jennings, N.R., Shadbolt, N.R.: The semantic grid: Past,present, and future. Proceedings of the IEEE 93, 669–681 (2005)
44. Dail, H., Sievert, O., Berman, F., Casanova, H., YarKhan, A., Vadhiyar, S., Dongarra, J., Liu, C., Yang, L., Angulo, D., Foster, I.: Scheduling in the grid application development software project. Grid resource management: state of the art and future trends, 73–98 (2004)
45. J.: d. Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, D. Fensel. The WSML Family of Representation Languages. Working draft, Digital Enterprise Research Insitute (DERI) (March 2005), Available from http://www.wsmo.org/TR/d16/d16.1/v0.2/
46. De Roure, D., Frey, J., Michaelides, D., Page, K.: The collaborative semantic grid. In: Proceedings of 2006 International Symposium on Collaborative Technologies and Systems, Las Vegas, USA, pp. 411–418. IEEE Computer Society Press, Los Alamitos (2006)
47. De Roure, D., Jennings, N.R., Shadbolt, N.R.: The semantic grid: A future e-science infrastructure. In: Berman, F., Fox, G., Hey, A.J.G. (eds.) Grid Computing - Making the Global Infrastructure a Reality, pp. 437–470. John Wiley and Sons, Chichester (2003)

48. Deelman, E.: Mapping abstract complex workflows onto grid environments. Journal of Grid Computing 1, 25–39 (2003)
49. Degermark, M., Koehler, T., Pink, S., Schelen, O.: Advance reservations for predictive service in the internet. Multimedia Syst. 5(3), 177–186 (1997), doi:10.1007/s005300050054
50. Delaittre, T., Kiss, T., Goyeneche, A., G., Terstyanszky, W.S., Kacsuk, P.: GEMLCA: Running legacy code applications as grid services, Journal of Grid Computing 3(1-2), 75–90 (2005)
51. Duan, R., Prodan, R., Fahringer, T.: DEE: A distributed fault tolerant workflow enactment engine for grid computing. In: Yang, L.T., Rana, O.F., Di Martino, B., Dongarra, J. (eds.) HPCC 2005. LNCS, vol. 3726, pp. 704–716. Springer, Heidelberg (2005)
52. Duan, R., Prodan, R., Fahringer, T.: Run-time optimization for Grid workflow applications. In: International Conference on Grid Computing, IEEE Computer Society Press, Los Alamitos (2006)
53. Dumitrescu, C., Foster, I.: Usage policy-based cpu sharing in virtual organizations. In: GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, Washington, DC, USA, pp. 53–60. IEEE Computer Society Press, Los Alamitos (2004)
54. Globus DUROC-team: The dynamically updated request online coallocator., `http://www-fp.globus.org/duroc/`
55. Elmroth, E., Tordsson, J.: A Grid Resource Broker Supporting Advance Reservations and Benchmark-Based Resource Selection. In: Dongarra, J., Madsen, K., Waśniewski, J. (eds.) PARA 2004. LNCS, vol. 3732, pp. 1061–1070. Springer, Heidelberg (2006)
56. Elmroth, E., Gardfjall, P.: Design and evaluation of a decentralized system for grid-wide fairshare scheduling. In: E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing, Washington, DC, USA, pp. 221–229. IEEE Computer Society Press, Los Alamitos (2005)
57. Ewa Deelman et. al.: Transformation Catalog Design for GriPhyN. Technical report griphyn-2001-17 (2001)
58. Deelman, E., et al.: Mapping abstract complex workflows onto grid environments. Journal of Grid Computing 1, 25–39 (2003)
59. EUROGRID: Application testbed for european grid computing, `http://www.eurogrid.org`
60. Expect, `http://expect.nist.gov/`
61. Fahringer, T.: ASKALON - A Programming Environment and Tool Set for Cluster and Grid Computing. Institute for Computer Science, University of Innsbruck, `http://dps.uibk.ac.at/askalon`
62. Fahringer, T., Prodan, R., Duan, R., Hofer, J., Nadeem, F., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., Wieczorek, M.: ASKALON: A Development and Grid Computing Environment for Scientific Workflows. In: Workflows for eScience, Scientific Workflows for Grids, p. 450
63. Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., Wieczorek, M.: ASKALON: A Grid Application Development and Computing Environment. In: 6th International Workshop on Grid Computing (Grid 2005), Seattle, Washington, USA, November 2005, IEEE Computer Society Press, Los Alamitos (2005)

64. Fahringer, T., Qin, J., Hainzer, S.: Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In: Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005), Cardiff, UK, May 9-12, 2005, IEEE Computer Society Press, Los Alamitos (2005)
65. Feitelson, D., Rudolph, L., Schwiegelshohn, U.: Parallel job scheduling – a status report. In: 10th Workshop on Job Scheduling Strategies for Parallel Processing, New-York (June 2004)
66. Fikes, R., Hayes, P., Horrocks, I.: OWL-QL: A Language for Deductive Query Answering on the Semantic Web. Technical Report KSL, Stanford University, Stanford, CA (2003)
67. Sintek, M., Decker, S.: *TRIPLE*–A query, inference, and transformation language for the semantic web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 364–378. Springer, Heidelberg (2002)
68. Enabling Grid for E-science (EGEE). Lightweight Middleware for Grid Computing, `http://cern.ch/glite`
69. Corporation for national Resource Initiatives. Handle System, `http://www.handle.net`
70. Unicore Forum. UNICORE: Uniform Interface to Computing Resources, `http://www.unicore.eu`
71. Foster, I., Gannon, D., Kishimoto, H.: OGSA: The Open Grid Services Architecture. The Global Grid Forum (November 2003), `https://forge.gridforum.org/projects/ogsa-wg`
72. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufmann, San Francisco (2004)
73. Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K., Roy, A.: A distributed resource management architecture that supports advance reservations and co-allocation. In: Proceedings of the International Workshop on Quality of Service (1999)
74. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications 15(3) (2001)
75. Foster, I.: What is the grid? a three point checklist. GRIDTODAY 1(6) (2002)
76. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)
77. Foster, I., Kesselman, C.: The grid in a nutshell. In: Grid resource management: state of the art and future trends, pp. 3–13 (2004)
78. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS-98), New York, November 3–5, 1998, pp. 83–92. ACM Press, New York (1998)
79. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: A Virtual Data System For Representing, Querying, and Automating Data Derivation. In: 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002) (September 2002)
80. Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., Darlington, J.: Optimisation of component-based applications within a grid environment. In: Proceedings of the 2001 ACM/IEEE conference on Supercomputing, November 10–16, 2001, ACM Press, New York (2001)

81. Global grid forum, http://www.ggf.org
82. Ghanem, M., Azam, N., Boniface, M., Ferris, J.: Grid-enabled workflows for industrial product design. In: E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, Washington, DC, USA, p. 96. IEEE Computer Society Press, Los Alamitos (2006)
83. Glatard, T., Emsellem, D., Montagnat, J.: Generic web service wrapper for efficient embedding of legacy codes in service-based workflows. In: Grid-Enabling Legacy Applications and Supporting End Users Workshop (GELA'06), Paris, France, June 2006, pp. 44–53 (2006), http://www.i3s.unice.fr/~johan/publis/GELA06.pdf
84. GLogin, http://www.gup.uni-linz.ac.at/glogin/
85. OGF GLUE-WG: Glue schema working group (GLUE-wg), http://forge.gridforum.org/sf/projects/glue-wg
86. Goble, C., Corcho, O., Xing, W.: ActOn: A Semantic Information Service for EGEE. In: The 8th IEEE/ACM International Conference on Grid Computing (Grid 2007), Austin, Texas (2007)
87. Global Grid Forum GRAAP-wg. GRAAP: Grid Resource Allocation Agreement Protocol (2006), https://forge.gridforum.org/projects/graap-wg
88. Semantic Grid: Semantic grid community portal, http://www.semanticgrid.org
89. GRUBER: A Grid Resouce uSLA-based Broker: http://people.cs.uchicago.edu/~cldumitr/GRUBER
90. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5(2), 199–220 (1993), doi:10.1006/knac.1993.1008
91. Habib, I.: Getting started with condor. Linux Journal 2006(149), 2 (2006)
92. Hafid, A., von Bochmann, G., Dssouli, R.: A quality of service negotiation approach with future reservations (NAFUR): a detailed study. Computer Networks and ISDN Systems 30(8), 777–794 (1998), citeseer.ist.psu.edu/article/hafid98quality.html
93. Hofer, J., Villazon, A., Siddiqui, M., Fahringer, T.: The Otho Toolkit: Generating tailor-made scientific grid application wrappers. In: German Society of Informatics (ed.) 2nd International Conference on Grid Service Engineering and Management (GSEM'05), Erfurt, Germany, September 19-22, 2005. Lecture Notes in Informatics (2005)
94. Modeling stateful resources with web services describes the ws-resource construct. ws-resource/ws-modelingresources.pdf, http://www-128.ibm.com/developerworks/library/
95. IBM: IBM solutions grid for business partners: Helping IBM business partners to grid-enable applications for the next phase of e-business on demand. PartnerWorld (2002)
96. IT Innovation: Workflow enactment engine (October 2002), http://www.it-innovation.soton.ac.uk/mygrid/workflow/
97. Jackson, D.B.: Grid scheduling with maui/silver. Grid resource management: state of the art and future trends, 161–170 (2004)
98. Jackson, D.B., Snell, Q., Clement, M.J.: Core Algorithms of the Maui Scheduler. In: JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing, London, UK, pp. 87–102. Springer, Heidelberg (2001)

99. Jasper, K.: Hydrological Modelling of Alpine River Catchments Using Output Variables from Atmospheric Models. Phd thesis, eth zurich, 2001 diss. eth no. 14385 (2001)
100. Globus Grid Forum JSDL-wg. Job Submission Description Language (JSDL), http://www.ggf.org/documents/GFD.56.pdf
101. Kim, K.H., Buyya, R.: Fair Resource Sharing in Hierarchical Virtual Organizations for Global Grids. In: 8th International Workshop on Grid Computing (Grid 2007), Austin, Texas, USA, September 2007, IEEE Computer Society Press, Los Alamitos (2007)
102. Kra, D.: Six strategies for grid application enablement, http://www-128.ibm.com/developerworks/grid/library/gr-enable/
103. LDAP: Lightweight directory access protocol, http://www.openldap.org
104. Linked environments for atmospheric discovery, https://portal.leadproject.org
105. Lim, H.N., Keung, C., Dyson, J.R.D., Jarvis, S.A., Nudd, G.R.: Performance Modelling of a Self-Adaptive and Self-Optimising Resource Monitoring System for Dynamic Grid Environments. In: AHM2005, Fourth All Hands Meeting, Nottingham (September 2005)
106. Liu, C., Yang, L., Foster, I., Angulo, D.: Design and evaluation of a resource selection framework for grid applications. In: In Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing (July 2002)
107. Liu, C., Foster, I.: A constraint language approach to matchmaking. In: RIDE '04: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04), Washington, DC, USA, pp. 7–14. IEEE Computer Society Press, Los Alamitos (2004)
108. Liu, C., Yang, L., Foster, I., Angulo, D.: Design and Evaluation of a Resource Selection Framework for Grid Applications. In: HPDC-11, the Symposium on High Performance Distributed Computing, Scotland (July 2002)
109. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
110. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D.A., Freund, R.F.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Heterogeneous Computing Workshop, p. 30 (1999)
111. Super clusters: Center for HPC Cluster Resource Management and Scheduling. Cluster Resources, Inc., http://www.supercluster.org/projects/
112. Mayer, A., McGough, S., Furmento, N., Lee, W., Newhouse, S., Darlington, J.: ICENI dataflow and workflow: Composition and scheduling in space and time. In: UK e-Science All Hands Meeting, Nottingham, UK, September 2003, pp. 627–634 (2003)
113. Mayer, A., McGough, S., Gulamali, M., Young, L., Stanton, J., Newhouse, S., Darlington, J.: Meaning and behaviour in grid oriented components. In: 3rd International Workshop on Grid Computing, Grid 2002, volume 2536 of Lecture Notes in Computer Science, Baltimore, USA (November 2002)
114. McDermott, D.: Estimated-regression planning for interactions with web services. In: Sixth International Conference in AI Planning Systems,, Telescience (2002)

115. McGough, A.S., Afzal, A., Darlington, J., Furmento, N., Mayer, A., Young, L.: Making the grid predictable through reservation and performance modelling. The Computer Journal 48(3), 358–368 (2005)
116. McGough, A.S., Lee, W., Darlington, J.: ICENI II Architecture. In: UK e-Science All Hands Meeting, Nottingham, UK (September 2005)
117. Mohamed, H.H., Epema, D.H.J.: Experiences with the koala co-allocating scheduler in multiclusters. In: CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), Washington, DC, USA, vol. 2, pp. 784–791. IEEE Computer Society Press, Los Alamitos (2005)
118. Nadeem, F., Yousaf, M.M., Prodan, R., Fahringer, T.: Soft benchmarks-based application performance prediction using a minimum training set. In: E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, Washington, DC, USA, p. 71. IEEE Computer Society Press, Los Alamitos (2006)
119. Natrajan, A., Humphrey, M.A., Grimshaw, A.S.: Grid resource management in legion. Grid resource management: state of the art and future trends, 145–160 (2004)
120. NCEDC: Northern california earthquake data center, `http://www.ncedc.org`
121. Nerieri, F., Prodan, R., Fahringer, T., Truong, H.L.: Overhead analysis of grid workflow applications. In: 7th IEEE/ACM International Conference on Grid Computing (Grid'06), Barcelona, Spain, IEEE Computer Society Press, Los Alamitos (2006)
122. Nerieri, F., Siddiqui, M., Hofer, J., Villazon, A., Prodan, R., Fahringer, T.: Using a heterogeneous service-oriented grid infrastructure for movie rendering. In: 1st Austrian Grid Symposium, AGS'05, Schloss Hagenberg, Austria, December 01-02 2005, To be published by OCG Verlag (2005)
123. S., Netessine, R.S.: Introduction to the theory and practice of yield management. INFORMS Transactions on Education 3(1), `http://ite.informs.org/Vol3No1/NetessineShumsky`
124. Novotny, J., Tuecke, S., Welch, V.: An online credential repository for the Grid: MyProxy. In: Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), August 2001, IEEE Computer Society Press, Los Alamitos (2001)
125. OASIS: Advancing open stanford for information society, `http://www.oasis-open.org`
126. OASIS: UDDI: The Universal Description, Discovery and Integration, `http://www.uddi.org/about.html`
127. Berkeley University of California: Search for extraterrestrial intelligence (SETI) `http://setiathome.berkeley.edu/`
128. The University of Wisconsin: Condor: High Throughput Computing, `http://www.cs.wisc.edu/condor/`
129. OGF: Open Grid Forum, `http://www.ogf.org`
130. Oram, A. (ed.): Peer-to-peer: Harnessing the Power of Disruptive Technologies. O'Reilly, Sebastopol, California (2001)
131. P-Grade: Parallel grid runtime and application developement environment, `http://www.lpds.sztaki.hu/pgrade/`
132. Platform. LSF, `http://www.platform.com/Products/Platform.LSF.Family`
133. POVray, `http://www.povray.org`

134. Poznanski, P., Melia, G.C., Leiva, R.G., Cons, L.: Quattor - a framework for managing grid-enabled large-scale computing fabrics. In: Workshop of the 4th Cracow Grid Workshop 2004 (December 2004)
135. Van Praag, A., Segal, B.: Potential hep applications of a new high performance networking technology. Cern, it/2000-007, August 21 (1999)
136. Prodan, R., Fahringer, T.: Grid Computing. LNCS, vol. 4340. Springer, Heidelberg (2007)
137. Prodan, R., Fahringer, T.: Overhead analysis of scientific workflows in grid environments. IEEE Transactions on Parallel and Distributed Systems (2007)
138. Invmod Wasim project: The water flow and balance simulation model (WaSiM-ETH), http://www.dps.uibk.ac.at/~marek/projects
139. Qin, J., Fahringer, T.: Advanced Data Flow Support for Scientific Grid Workflow Applications. In: Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (Supercomputing 2007, SC—07), Reno, NV, USA, November 2007, IEEE Computer Society Press, Los Alamitos (2007)
140. Raman, R., Livny, M., Solomon, M.: Policy driven heterogeneous resource coallocation with gangmatching. In: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, p. 80. IEEE Computer Society Press, Los Alamitos (2003)
141. ROADNet: Real-time observatories, and applications, and data management network, http://roadnet.ucsd.edu
142. Roeblitz, T., Schintke, F., Reinefeld, A.: Resource reservations with fuzzy requests. Concurrency and Computation: Practice and Experience (2006)
143. Roeblitz, T., Schintke, F., Wendler: Elastic grid reservations with user-defined optimization policies. In: Proceedings of the Workshop on on Adaptive Grid Middleware (2004)
144. Roure, D.D., Baker, M.A., Jennings, N.R., Shadbolt, N.R.: The evolution of the grid. In: Grid Computing - Making the Global Infrastructure a Reality, pp. 65–100 (2003), http://eprints.ecs.soton.ac.uk/6871
145. Russell, M., Allen, G., Goodale, T., Nabrzyski, J., Seidel, E.: Application requirements for resource brokering in a grid environment. Grid resource management: state of the art and future trends, 25–40 (2004)
146. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language 1.0 reference, W3C Proposed Recommendation, Available from http://www.w3.org/TR/owl-ref/
147. Sagal, B.: Grid computing: The european datagrid project. In: IEEE Nuclear Science Symposium and Medical Imaging Conference, Lyon France, October 2000, IEEE Computer Society Press, Los Alamitos (2000)
148. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.: Scheduling workflows with budget constraints. In: Gorlatch, S., Danelutto, M. (eds.) Integrated Research in GRID Computing (2007)
149. Schopf, J.M.: Ten actions when grid scheduling: the user as a grid scheduler. Grid resource management: state of the art and future trends, 15–23 (2004)
150. Schwiegelshohn, U., Yahyapour, R.: Attributes for communication between grid scheduling instances. Grid resource management: state of the art and future trends, 41–52 (2004)

151. Seidel, E., Allen, G., Merzky, A., Nabrzyski, J.: GridLab: A grid application toolkit and testbed. Future Generation of Computer Systems 18(8), 1143–1153 (2002)
152. Shang, Y., Wah, B.W.: A discrete lagrangian-based global-search method for solving satisfiability problems. J. of Global Optimization 12(1), 61–99 (1998)
153. Siddiqui, M., Fahringer, T.: GridARM: Askalon's Grid Resource Management System. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 122–131. Springer, Heidelberg (2005)
154. Siddiqui, M., Fahringer, T.: Semantically-enhanced on-demand resource provision and management for the grid. Journal of Multiagent and Grid Systems 3 (2007)
155. Siddiqui, M., Fahringer, T., Hofer, J., Toma, I.: Grid resource ontologies and asymmetric resource-correlation. In: Informatics, G.S.o. (ed.) 2nd International Conference on Grid Service Engineering and Management (GSEM'05), Erfurt, Germany, September 19-22, 2005. Lecture Notes in Informatics, pp. 205–218 (2005)
156. Siddiqui, M., Villazón, A., Fahringer, T.: Grid capacity planning with negotiation-based advance reservation for optimized QoS. In: SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida, p. 103. ACM Press, New York (2006), doi:10.1145/1188455.1188563
157. Siddiqui, M., Villazon, A., Fahringer, T.: Semantic-based on-demand synthesis of grid activities for automatic workflow generation. In: 3rd IEEE International Conference on e-Science and Grid Computing (e-Science), Banglore, India, December 2007, IEEE Computer Society Press, Los Alamitos (2007)
158. Siddiqui, M., Villazon, A., Hofer, J., Fahringer, T.: GLARE: A Grid Activity Registration, Deployment and Provisioning Framework. In: International Conference for High Performance Computing, Networking and Storage (SuperComputing), SC05, Seattle, Washington, USA, November 12-18, 2005, p. 52. ACM Press, New York (2005)
159. Singh, G., Kesselman, C., Deelman, E.: Performance impact of resource provisioning on workflows. Technical report 05-850, Information Sciences Institute, University of Southern California (2005), `http://www.cs.usc.edu/Research/TechReports/05-850.pdf`
160. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. In: Journal of Web Semantics (2006), Available as `http://www.mindswap.org/papers/PelletJWS.pdf`
161. Slota, R., Zieba, J., Kryza, B., Kitowski, J.: Knowledge evolution supporting automatic workflow composition. In: E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, Washington, DC, USA, p. 37. IEEE Computer Society Press, Los Alamitos (2006)
162. Smith, W., Foster, I., Taylor, V.: Scheduling with advanced reservations. In: 14th International Parallel and Distributed Processing Symposium, IPDPS'00 (2000)
163. Simple network management protocol, `http://www.snmplink.org/src/SNMPv3.html`
164. SPARQL: a candidate recommendation. sparql_is_a_candidate_recommendation, `http://www.w3.org/blog/SW/2007/06/15/`
165. SRB: SDSC Storage Resource Broker, `http://www.sdsc.edu/srb`

166. Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S., Gehrke, J., Plaxton, C.G.: A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. In: IEEE Real-Time Systems Symposium (December 1996)
167. Sulistio, A., Buyya, R.: A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems. In: SBAC-PAD '05: Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing, Washington, DC, USA, pp. 235–242. IEEE Computer Society Press, Los Alamitos (2005)
168. Sun Microsystems Inc.: Sun Grid Engine, http://www.sun.com/software/gridware/
169. Talia, D., Trunfio, P.: Toward a Synergy Between P2P and Grids. IEEE Internet Computing 7, 94–96 (2003)
170. Tangmunarunkit, H., Decker, S., Kesselman., C.: Ontology-based Resource Matching in the Grid–The Grid meets the Semantic Web.. In: Second International Semantic Web Conference, Sanibel-Captiva Islands, Florida, pp. 706–721 (2003)
171. The taverna project, http://taverna.sourceforge.net
172. Tick, J., Kovacs, Z., Friedler, F.: Synthesis of optimal workflow structure. Journal of Universal Computer Science 12(9), 1385–1392 (2006)
173. Tick, J.: P-graph-based workflow modelling. Acta Polytechnica Hungarica 4(1), 75–88 (2007)
174. The triana project, http://www.trianacode.org/about
175. Tschudin, C.: Open Resource Allocation for Mobile Code. In: First International Workshop on Mobile Agents, Berlin (1997)
176. In, J.-u., Avery, P., Cavanaugh, R., Ranka, S.: Policy based scheduling for simple quality of service in grid computing. In: 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 2004, IEEE Computer Society Press, Los Alamitos (2004)
177. Stanford Univesity: Protege: a free, open source ontology editor and knowledge-base framework, http://protege.stanford.edu
178. Haarslev, V., Möller, R.: RACER system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–705. Springer, Heidelberg (2001)
179. von Laszewski, G., Foster, I.T., Gawor, J.: CoG kits: a bridge between commodity distributed computing and high-performance grids. In: Java Grande, pp. 97–106 (2000), citeseer.ist.psu.edu/vonlaszewski00cog.html
180. W3C: OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S
181. W3C: Semantic Web Activity, http://www.w3.org/2001/sw/
182. W3C: SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
183. W3C: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL
184. W3C: Web Services Addressing (WS-Addressing), http://www.w3.org/Submission/ws-addressing
185. Waldspurger, C., Weihl, W.: Lottery scheduling: Flexible proportional share resource management. In: First Symposium on Operating System Design and Implementation (OSDI), USENIX (1994)

186. Lesser, V., Ortiz, C., Tambe, M. (eds.): Analysis of Negotiation Protocols by Distributed Search, pp. 339–361. Kluwer Academic Publishers, Dordrecht (2003), http://mas.cs.umass.edu/paper/249
187. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of Scientific Workflows in the ASKALON Grid Environment. ACM SIGMOD Record 35(3), 56–62 (2005), http://dps.uibk.ac.at/~marek/publications/acm-sigmod-2005.pdf
188. Wieczorek, M., Siddiqui, M., Villazon, A., Prodan, R., Fahringer, T.: Applying advance reservation to increase predictability of workflow execution on the grid. In: E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, Washington, DC, USA, p. 82. IEEE Computer Society Press, Los Alamitos (2006)
189. Wolf, L., Steinmetz, R.: Concepts for Resource Reservation in Advance (Special Issue on State of the Art in Multimedia Computing). Multimedia Tools and Applications 4(3), 255–278 (1997)
190. Wolski, R., Spring, N.T., Hayes, J.: The network weather service: a distributed resource performance forecasting service for metacomputing. Future Generation Computer Systems 15(5–6), 757–768 (1999), http://www.elsevier.com/gej-ng/10/19/19/30/21/35/abstract.html
191. The Globus Alliance WSRF: WSRF: Web Services Resource Framework, http://www.globus.org/wsrf
192. XQuery: An XML Query Language, http://www.w3.org/TR/xquery
193. Gil, Y., Deelman, E., Blythe, J., Kesselman, C., Tangmunarunkit, H.: Artificial intelligence and grids: workflow planning and beyond. Intelligent Systems 19(1), 26–33 (2004)
194. Yang, B., Bundy, A., Smaill, A., Dixon, L.: Deductive synthesis of workflows for e-science. In: SIGAW Workshop of CCGrid 2005, Cardiff (2005)
195. Younis, M.A., Saad, B.: Optimal resource leveling of multi-resource projects. Comput. Ind. Eng. 31(1-2), 1–4 (1996)
196. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming 14, 217–230 (2006)
197. Yu, J., Buyya, R.: A Taxonomy of Workflow Management Systems for Grid Computing. Technical report grids-tr-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia (Mar 2005), http://www.gridbus.org/
198. Yu, J., Kirley, M., Buyya, R.: Multi-objective planning for workflow execution on grids. In: 8th International Workshop on Grid Computing (Grid 2007), Austin, Texas, USA, September 2007, IEEE Computer Society Press, Los Alamitos (2007)
199. Zhang, W.: Modeling and solving a resource allocation problem with soft constraint techniques. Technical report wucs-2002-13, Department of Computer Science and Engineering, Washington University in St. Louis (May 2002)
200. Zhang, X., Schopf, J.M.: Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. In: Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Conference (IPCCC) (April 2004)

Index