26

THE GRIDBUS MIDDLEWARE FOR MARKET-ORIENTED COMPUTING

RAJKUMAR BUYYA, SRIKUMAR VENUGOPAL, RAJIV RANJAN, AND CHEE SHIN YEO

26.1 INTRODUCTION

Grids aim at exploiting synergies that result from the cooperation of autonomous distributed entities. The synergies that result from Grid cooperation include the sharing, exchange, selection, and aggregation of geographically distributed resources such as computers, databases, software, and scientific instruments for solving large-scale problems in science, engineering, and commerce. For this cooperation to be sustainable, participants need to have economic incentives. Therefore, "incentive" mechanisms should be considered as one of the key design parameters for designing and developing end-to-end Grid architectures. Although several studies have investigated market-oriented management of Grids, they were limited mostly to specific aspects of the system design such as service pricing or price-aware scheduling. This chapter presents architectural models, mechanisms, algorithms, and middleware services developed by the Gridbus project for end-to-end realization of market-oriented Grid computing.

Grid technologies such as Globus provide capabilities and services required for the seamless and secure execution of a job on heterogeneous resources. However, to achieve the complete vision of Grid as a utility computing environment, a number of challenges need to be addressed. They include designing Grid services capable of distributed application composition, resource brokering methodologies, policies and strategies for scheduling different Grid application models, Grid economy for data and resource management, application service specification, and accounting of

Market-Oriented Grid and Utility Computing Edited by Rajkumar Buyya and Kris Bubendorfer

resource consumption. The application development and deployment services need to scale from desktop environments to global Grids and support both scientific and business applications.

The Gridbus project is engaged in the design and development of service-oriented cluster and Grid middleware technologies to support e-science and e-business applications. It extensively leverages related software technologies and provides an abstraction layer to hide idiosyncrasies of heterogeneous resources and low-level middleware technologies from application developers. In addition, it extensively focuses on the realization of the utility computing model scaling from clusters to Grids and to the peer-to-peer computing systems. It uses economic models that aids in the efficient management of shared resources and promotes commoditization of their services. Thus, it enhances the tradability of Grid services according to their supply and demand in the system. Gridbus supports the commoditization of Grid services at various levels:

- Raw resource level (e.g., selling CPU cycles and storage resources)
- Application level (e.g., molecular docking operations for drug design application)
- Aggregated services (e.g., brokering and reselling of services across multiple domains)

The computational economy methodology helps in creating a service-oriented computing architecture where service providers offer paid services associated with a particular application and users, on the basis of their requirements, would optimize by selecting the services that they require and can afford within their budgets. Gridbus hence emphasizes the end-to-end quality of services driven by computational economy at various levels—clusters, peer-to-peer (P2P) networks, and the Grid—for the management of distributed computational, data, and application services.

Gridbus provides software technologies that spread across the following categories:

- Enterprise grid middleware with service-level agreement (SLA)-based resource allocation (Aneka)
- Grid economy and virtual enterprises (Grid Market Directory)
- Grid trading and accounting services (GridBank)
- Grid resource brokering and scheduling (Gridbus Broker)
- Grid workflow management (Gridbus Workflow Engine)
- Grid application development tools (Visual Parametric Modeller)
- Grid portals (Gridscape)

26.2 ARCHITECTURE

The Gridbus project aims to develop software frameworks and algorithms to realize a market-driven Grid computing environment, an example of which is illustrated in

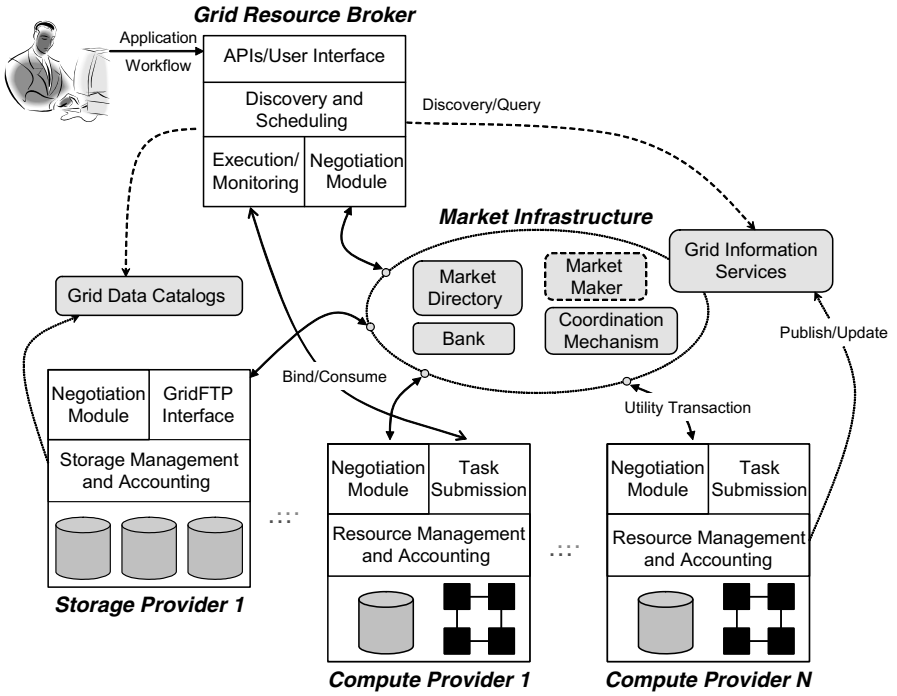


Figure 26.1 Architectural elements of market-based Grid computing.

Figure 26.1. The resource providers offer various resources, and are driven by the twin motivations of maximizing their profit and resource utilization. The requirement of the user is to execute her application given her requirements, such as accessing specific datasets for processing and/or a deadline for its completion. The user is constrained by her budget for accessing resources, and possibly by other factors such as access restrictions on certain storage resources and computing environments that can execute her application. The user operates through a Grid resource broker that, given user requirements and constraints, discovers appropriate resources, negotiates with them for access, executes the application, and returns the results to the user. The interface between the broker and the providers is enabled through the market infrastructure that provides functionalities such as directory of providers, and accounting and banking. In the following paragraphs, we will look at each participant and the related Gridbus components.

A layered view of its realization within the Gridbus middleware is shown in Figure 26.2. The Gridbus software stack is primarily divided into five layers: Grid applications layer, user-level middleware layer, core Grid middleware layer, Grid fabric software layer, and Grid fabric hardware layer. The notion of Grid economics is prevalent at each of these layers. At the *Grid applications layer*, the Gridbus project contributes through its monitoring and application composition Grid portals. These Grid portals have the capability to seamlessly interact with services running at the

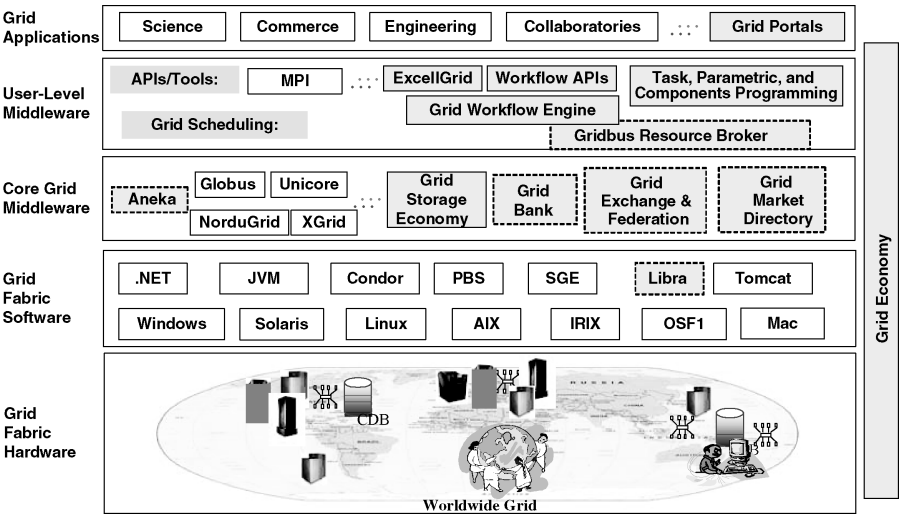


Figure 26.2 Gridbus software stack. Gridbus components are shown in colored background.

user-level middleware layer, including the Gridbus resource broker and workflow engine. At the *core Grid middleware layer*, Gridbus has developed software services for enterprise desktop Grid integration (Aneka), accounting (GridBank), cooperative resource management (Grid-Federation), and resource discovery (Grid Market Directory). The Libra system, which operates at the *Grid fabric software layer*, supports market-based allocation of cluster computing resources.

The Grid fabric hardware layer includes different kinds of computing, data, and storage facilities that belong to different Grid resource-sharing domains. There can be different types of providers offering different kinds of services to users. In Figure 26.1, we have depicted compute and storage providers, as these are the two basic resources required by any application. A compute provider leases highly capable computational resources such as supercomputers or clusters to the Grid environment. Such resources are generally managed by a queue-based scheduling system that allocates jobs to processors or nodes. However, most cluster management systems aim to improve system-centric metrics such as utilization. In contrast, Libra is an economy-based cluster scheduler that focuses on improving the quality of service (QoS) on a per-user basis. In addition, the resource may provide the ability to reserve nodes or processors in advance. The advance reservation is conducted through the negotiation interface that also enables the provider to participate in the market. These capabilities are provided in Aneka [10], a .NET-based enterprise Grid framework, in addition to the traditional cluster resource manager functions such as job submission and management. While the description so far relates to a compute provider, a storage provider would have similar components, except that the resource management would be replaced by storage management functions. Providers also track resource usage through accounting mechanisms to bill the users for their execution.

The Gridbus Grid resource broker [5] functions as a user agent in the market-oriented environment shown in Figure 26.1. The broker uses the user's requirements to discover appropriate Grid resources by querying Grid information services such as Globus' *Grid information indexing service* (GIIS) [12]. Market information such as prices and offers are queried from the market directory. Using this information, the broker identifies suitable providers and either carries out one-to-one negotiations or participates in auctions for resource shares. It then schedules user jobs over the acquired resource shares such that the deadline requirements of the user are met.

The primary components of the current market infrastructure are the Grid Market Directory (GMD) [6] and GridBank [7]. The GMD enables providers to advertise their services to the users through a registry service. Brokers can query the GMD to locate required services and query their attributes such as service addresses, pricing, and input methods. Other information services such as GIIS can also be considered as part of the market infrastructure as they allow the broker to discover capabilities and status of services, which, in turn, determine their value. GridBank is a accounting and micropayment service that provides an infrastructure for secure payments between the users and providers. GridBank can also be used as an accounting and authorization mechanism wherein only users with requisite credit in their accounts can enter into contracts with providers. It is important to note that there may be more than one instance of these components present in a Grid. As the scale of providers, brokers, and market components increases, it becomes necessary to connect these entities on the basis of a decentralized and scalable network model. Furthermore, these entities need to coordinate their activities in a scalable manner to achieve the desired systemwide objective functions. One such mechanism is the Compute Power Market [14], built using the JXTA infrastructure from Sun Microsystems, which allows the trading of computational power over peer-to-peer networks. Another more recent advancement with respect to coordinated Grid resource management has been the GridFederation [18] model, which encapsulates decentralized protocols and algorithms for efficient discovery and coordinated provisioning of resources in federated Grid and peer-to-peer systems.

26.3 GRID RESOURCE BROKER

The Gridbus broker is an advanced service-oriented metascheduler for compute and data Grids, with support for a wide range of Grid middleware and services. It accommodates many functions that Grid applications require, including discovering the right resources for a particular user application, scheduling jobs in order to meet deadlines, and handling faults that may occur during execution. In particular, the broker provides capabilities such as resource selection, job scheduling, job management, and data access to any application that requires distributed Grid resources for execution. The broker handles communication with the resources running different Grid middleware, job failures, varying resource availability, and different user objectives such as meeting a deadline for execution or limiting execution within a certain budget.

26.3.1 Architecture

The design of the Gridbus broker follows a layered architecture consisting of interface, core, and execution layers that together provide the capabilities shown for the market-oriented broker in Figure 26.1. The interface layer consists of application programming interfaces (APIs) and parsers for the input files through which external programs and users communicate with the broker, respectively. Resource discovery and negotiation, scheduling, and job monitoring are carried out in the core layer. The job execution is carried out through the execution layers in which middleware-specific adapters communicate with the target resources.

26.3.2 Input

There are many ways to specify the user requirements to the broker. Figure 26.3 shows a user application specified using the broker's own XPML (Extended Parametric Modeling Language) format. The `qos` tags enclose the user's QoS requirements, which, in the example, specify the deadline by which the job must be executed and the budget available for execution. The user wants the execution completed with the least

```
<xpml>
        <qos>
               <deadline value="2007-ll-10T19 :30 :" />
               <budget value="l0000.0" />
               <optimisation value="COST" />
        </qos>
        <parametar name="X" type=" integer" domain=
           "range" >
               <range from="1" to="10" interval="l" />
        </parameter>
        <parameter name="time_value" type="integer"
           domain=" single">
               <single value="3000" />
        </parameter>
        <job-requirements>
               <property name="estimatedTime"
                  value="60.00" />
        </job-requirements>
        <task>
               <execute>
                      <command value="calc" />
                      <arg valuer=" $X " />
                      <arg value=" $time_value "/>
               </execute>
        </tasK>
</xpml>
```

Figure 26.3 User requirement specification using XPML.

expense, which is indicated by the optimization value. XPML is used for specifying parameter sweep applications in which a single application is executed over a range of parameters. The `parameter` tags indicate the parameters, and the `task` tags specify the application to be executed.

Of interest to a market-oriented Grid is the QoS section. The values provided by the user in this section form the basis for the broker's resource discovery and scheduling mechanisms. While the only parameters recognized by the broker at present are the deadline, budget, and optimization values, the number of such inputs is limited only by the capabilities of the schedulers in the broker.

26.3.3 Discovery, Negotiation, and Scheduling

The broker queries resources for their capabilities and availability. Information about the resource costs is queried from the Grid Market Directory (GMD). Once the resources are identified, the broker may carry out one-to-one negotiations with them. The Gridbus broker has the ability to conduct bilateral negotiations with the resources by using the Alternate Offers Protocol [1]. The negotiation consists of the broker exchanging proposals with counter-proposals from the resource until both of them converge on an acceptable agreement, or one of them quits the process.

Figure 26.4 shows an extensible Markup Language (XML)-based negotiation proposal for reserving nodes in advance on a resource (with the values shown in bold). The broker creates this proposal according to the requirements given by the user. The reward field indicates the provider's gain for supplying the required number of

```
<xml - fragment xmlns: ws =" http: // www.gridbua.org/
   negotiation/ws">
   <ws : Reward>200.0</ws: Reward>
   <ws : Penalty>50.0</ws: Penalty>
   <ws: Requirements>
      <ws:ReservationRecordType>
         <ws:ReservationStartTime>
         2008-04-01T18:22:00.437+11:00
         </ws:ReservationStartTime>
       <ws:Duration>750000.0</ws:Duration>
       <ws:NodeRequirement>
         <ws:Count>4</ws:Count>
       </ws:NodeRequirement>
       <ws:CpuRequirement>
         <ws: Measure>Ghz</ws :Measure>
         <ws: Speed>2.5</ws:Speed>
       <ws:CpuRequirement>
      </ws:ReservationRecordType>
   </ws:Requirements>
</xml-fragment>
```

Figure 26.4 Negotiation proposal format.

resources. The penalty field denotes the penalty to be paid if the provider accepted the proposal but did not supply the required resources.

The requirements section here asks for four nodes with a minimum CPU speed of 2.5 GHz each for duration of 750 s starting from 6:22 p.m. on April 1, 2008. The provider (or resource) can, in turn, create a counterproposal by modifying sections of the broker's proposal and send that as a reply. The offers and counteroffers continue until one of the parties accepts the current proposal, or rejects it altogether. At present, the broker can negotiate only with Aneka [10], the resource management system covered in Section 26.6.

The broker enables different types of scheduling depending on the objectives of the user and type of resources. At present, the broker can accommodate compute, storage, network, and information resources with prices based on time (1 Grid dollar for 1 second), or capacity (1 Grid dollar for 1 MB). It can also accommodate user objectives such as the fastest computation within the budget (time optimization), or the cheapest computation within the deadline (cost optimization) for both compute and data-intensive applications. The compute-intensive algorithms are based on those developed previously in Nimrod/G [2]. A cost–time-minimizing algorithm for data-intensive applications is described in the following paragraphs. This algorithm was published and evaluated previously [3].

A distributed data-intensive computing environment consists of applications that involve mainly accessing, processing and transferring data of the order of gigabytes (GB) and upward. These operations are conducted over resources that are geographically distributed, and shared between different users. Therefore, the impact of data access and transfer operations on the execution time of the application and resource usage is equal to, if not more than, that of the compute-intensive processing operations. Transferring large volumes of data through the network can be very costly, and so can be processing it at an expensive compute resource. Therefore, the total cost can be defined as the sum of the processing cost, the data transfer (network) cost, and the storage cost. Likewise, the total time for execution is the sum of the job completion time and the data transfer time. A simple scheduling heuristic to reduce the total execution cost of the application can be expressed as follows:

1. Repeat for every scheduling interval while there are unprocessed jobs.
2. For every job, find the data file(s) that it is dependent on and locate the data hosts for those files.
3. Find a data-compute set (a set consisting of one compute resource for the execution and one data host for each file involved) that guarantees the minimum cost for that job.
4. Sort the jobs in order of increasing cost.
5. Assign jobs from the sorted list starting with the least expensive job until either all the jobs are allocated or all the compute resources have been allocated their maximum jobs.

Although this list shows only cost minimization, the same heuristic was followed in the case of time minimization except that the criterion in step 2 was changed to the *minimum execution time* required.

This scheduling algorithm was evaluated on resources distributed around Australia, listed in Table 26.1. The network connections between the compute resources were assigned artificial costs as given in Table 26.2. We used a synthetic application that transferred and processed large data files. These files were evenly distributed on the resources and were registered in a replica catalog [4]. The broker located the files by querying the catalog. For this experiment, we had 100 files, each 30 MB in size. Each job depended on one of the files, thus creating 100 jobs.

TABLE 26.1 Resources Used for Evaluation of Cost-Based Data-Intensive Scheduling

Organization[a]	Machine Details	Role	Cost [G$/(CPU·s)]	Total Jobs Executed	
				Time	Cost
Dept. Computer Science, Univ. Melbourne (UniMelb CS)	belle.cs.mu.oz.au; IBM eServer, 4 CPU, 2 GB RAM, 70 GB HD, Linux	Broker host, data host, NWS server	NA (not used as a compute resource)	—	—
School of Physics, Univ. Melbourne (UniMelb Physics)	fleagle.ph.unimelb.edu.au; PC, 1 CPU, 512 MB RAM, 70 GB HD, Linux	Replica catalog host, data host, computer resource, NWS sensor	2	3	94
Dept. Computer Science, Univ. Adelaide (Adelaide CS)	belle.cs.adelaide.edu.au; IBM eServer, 4 CPU (only 1 available), 2 GB RAM, 70 GB HD, Linux	Data host, NWS sensor	NA (not used as a compute resource)	—	—
Australian National Univ., Canberra (ANU)	belle.anu.edu.au; IBM eServer, 4 CPU, 2 GB RAM, 70 GB HD, Linux	Data host, computer resource, NWS sensor	4	2	2
Dept. Physics, Univ. Sydney (Sydney Physics)	belle.physics.usyd.edu.au; IBM eServer, 4 CPU (only 1 available), 2 GB RAM, 70 GB HD, Linux	Data host, compute resource, NWS sensor	4	72	2
Victorian Partnership for Advanced Computing, Melbourne (VPAC)	brecca-2.vpac.org; 180-node cluster (only head node used), Linux	Compute resource, NWS sensor	6	23	2

[a]This column lists abbreviations used in Table 26.2.

TABLE 26.2 Network Costs between Data Hosts and Compute Resources (in G$/MB)[a]

Data Node Compute Node	ANU	UniMelb Physics	Sydney Physics	VPAC
ANU	0	34.0	31.0	38.0
Adelaide CS	34.0	36.0	31.0	33.0
UniMelb Physics	40.0	0	32.0	39.0
UniMelb CS	36.0	30.0	33.0	37.0
Sydney Physics	35.0	33.0	0	37.0

[a]See Table 26.1, "Organization" column, for abbreviations used in this table.

TABLE 26.3 Summary of Evaluation Results

Scheduling Strategy	Total Time Taken (min)	Compute Cost (G$)	Data Cost (G$)	Total Cost (G$)
Cost minimization	71.07	26,865	7560	34,425
Time minimization	48.5	50,938	7452	58,390

Table 26.3 summarizes the results that were obtained. As is expected, cost minimization scheduling produces minimum computation and data transfer expenses, whereas time minimization completes the experiments in the least time. The graphs in Figures 26.5 and 26.6 show the number of jobs completed against time for the two

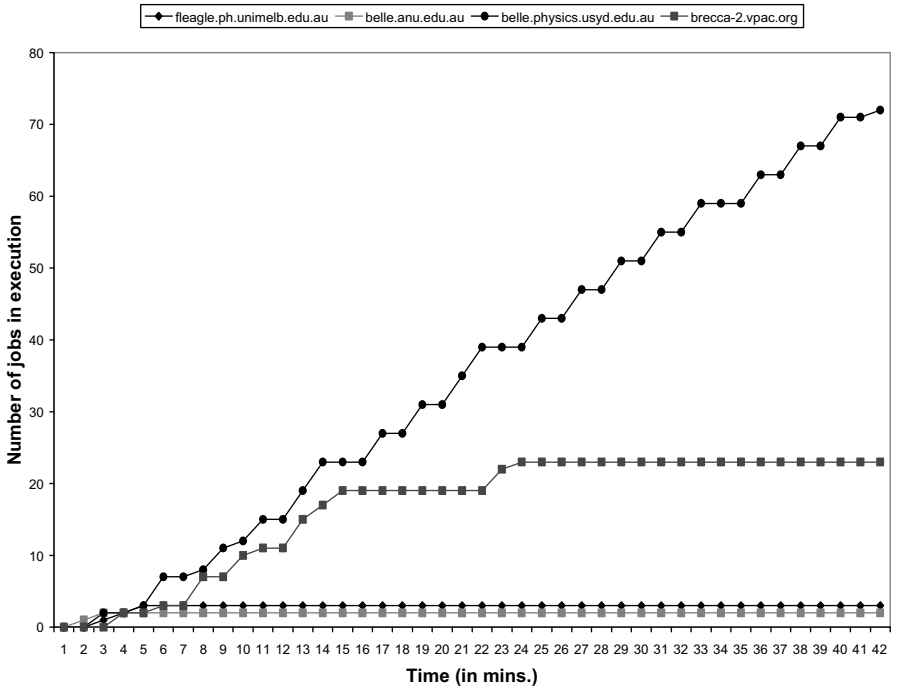


Figure 26.5 Cumulative number of jobs completed versus time for time minimization scheduling in data Grids.

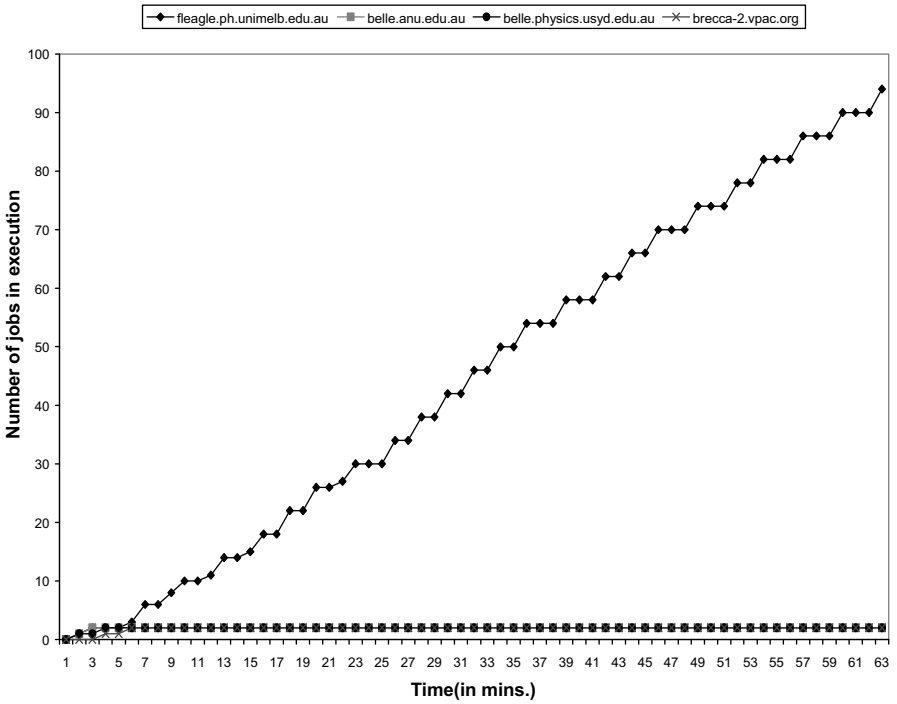


Figure 26.6 Cumulative number of jobs completed versus time for cost minimization scheduling in data Grids.

scheduling strategies. It can be seen that these mirror the trends for similar evaluations conducted with computational Grids [2]; that is, time minimization used the more expensive but faster resources to execute jobs, whereas cost minimization used the cheaper resource most to ensure a lower overall expense.

26.4 GRID MARKET DIRECTORY (GMD)

It has been envisioned that Grids enable the creation of virtual organizations (VOs) [11] and virtual enterprises (VEs) [13] or computing marketplaces [14]. In a typical marketbased model VO/VE, Grid service providers (GSPs) publish their offerings in a market directory (or a catalog), and Grid service consumers (GSCs) employ a Grid resource broker (GRB) that identifies GSPs through the market directory and utilize the services of suitable resources that meet their QoS requirements (see Fig. 26.7).

To realize this vision, Grids need to support diverse infrastructure/services [11], including an infrastructure that allows (1) the creation of one or more Grid market place (GMP) registries, (2) the contributors to register themselves as GSPs along with their resources/application services that they wish to provide, (3) GSPs to publish themselves in one or more GMPs along with service prices, and (4) Grid

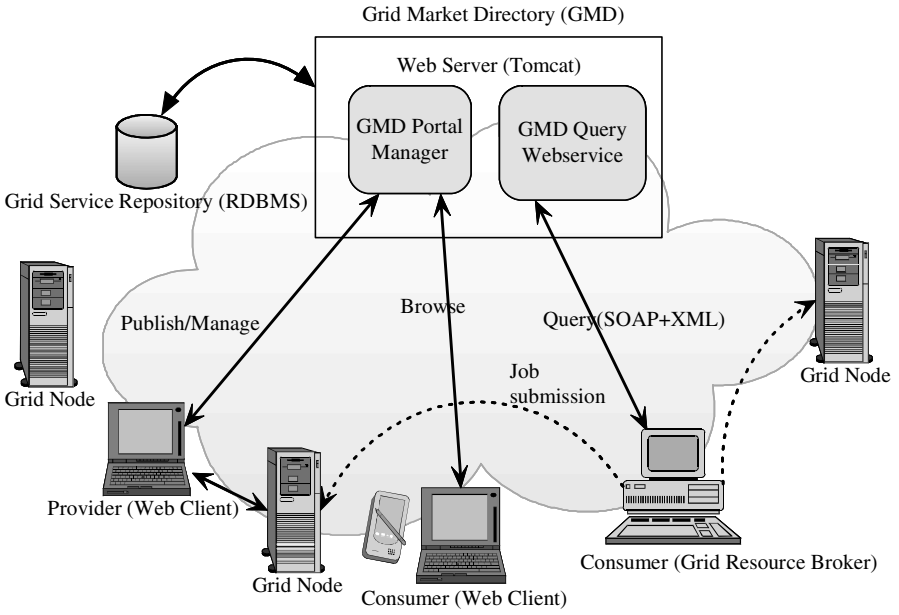


Figure 26.7 Grid Market Directory (GMD) architecture.

resource brokers to discover resources/services and their attributes (e.g., access price and usage constraints) that meet user QoS requirements. In this section, we describe a software framework called the *Grid Market Directory* (GMD) that supports these requirements.

The GMD [6] serves as a registry for high-level service publication and discovery in virtual organizations. It enables service providers to publish the services that they provide along with the costs associated with those services. Next, it allows consumers to browse the GMD for finding the services that meet their QoS requirements. The key components (refer to Fig. 26.7) of the GMD are

- *GMD portal manager* (GPM), which facilitates service publication, management, and browsing. It allows service providers and consumers to use a Web browser as a simple graphical client to access the GMD.
- *GMD query Web service* (GQWS), which enables applications (e.g., resource broker) to query the GMD to find a suitable service that meets the job execution requirements (e.g., budget).

Both components receive client requests through a HTTP server. Additionally, a database (GMD repository) is configured for recording the information of Grid services and service providers.

The GMD is built over standard Web service technologies such as Simple Object Access Protocol (SOAP) and XML. Therefore, it can be queried by programs

irrespective of their operating environment (platform independent) and software libraries (language-independent). To provide with an additional layer of transparency, a client API has been provided to enable programs to query the GMD directly, so that the developers need not concern themselves with SOAP details. The Gridbus resource broker interacts with the GMD to discover the testbed resources and their high-level attributes such as access price.

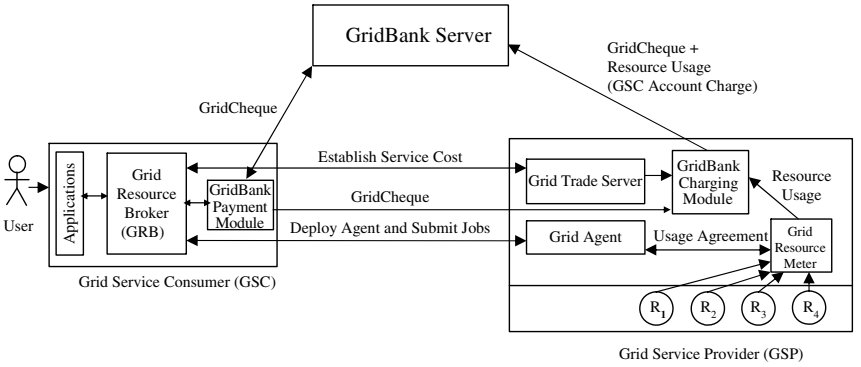
26.5 GridBank

The early efforts in Grid computing and usage scenarios were mostly academic or exploratory in nature and did not enforce the Grid economy mechanisms. With the more recent move toward a multiinstitutional production-scale Grid infrastructure such as the TeraGrid facility [8], the need for Grid economy and accounting is being increasingly felt. In order to enable the sharing of resources across multiple administrative domains, the accounting infrastructure needs to support unambiguous recording of user identities against resource usage. In the context of the Gridbus project, an infrastructure providing such a service is called the *GridBank* [7].

GridBank is a secure Grid-wide accounting and (micro)payment handling system. It maintains the users' (consumers and providers) accounts and resource usage records in the database. It supports protocols that enable its interaction with the resource brokers of GSCs and the resource traders of GSPs. It has been envisioned to provide services primarily for enabling Grid economy. However, we also envision its usage in e-commerce applications. The GridBank services can be used in both cooperative and competitive distributed computing environments.

GridBank can be regarded as a Web service for Grid accounting and payment. GridBank uses SOAP over Globus toolkit's sockets, which are optimized for security. Clients use the same user proxy/component to access GridBank as they use to access other resources on the Grid. A *user proxy* is a certificate signed by the user that is later used to repeatedly authenticate the user to resources. This preserves the Grid's single-signin policy and avoids the need to repeatedly enter the user password. Using existing payment systems for the Grid would not satisfy this policy.

The interaction between the GridBank server and various components of Grid is shown in Figure 26.8. GSPs and GSCs first open an account with GridBank. Then, the user submits the application processing requirements along with the QoS requirements (e.g., deadline and budget) to the GRB. The GRB interacts with GSP's Grid Trading Service (GTS) or Grid Market Directory (GMD) to establish the cost of services and then selects a suitable GSP. It then submits user jobs to the GSP for processing along with details of its chargeable account ID in the GridBank or GridCheque purchased from the GridBank. The GSP provides the service by executing the user job, and the GSP's Grid resource meter measures the amount of resources consumed while processing the user job. The GSP's charging module contacts the GridBank with a request to charge the user account. It also passes information related to the reason for charging (resource usage record).



1) GRB negotiates service cost per time unit (e.g., $ per hour)
2) GridBank Payment Module requests GridCheque for the GSP whose service GSC wants to use. GridBank issues GridCheque provided GSC has sufficient funds.
3) GridBank payment module forwards GridCheque to GridBank Charging Module.
4) GRB deploys Grid Agent and submits jobs for execution on the resource.
5) Grid resource meter gathers resource usage records from all resources used to provide the service, optionally aggregates individual records into one resource usage record and forwards it to the GridBank charging module. Grid resource meter optionally performs usage check with grid agent.
6) GridBank charging module contacts GridBank and redeems all outstanding payments. It can do so in batches rather than after each transaction.

Figure 26.8 GridBank.

26.6 ANEKA: SLA-BASED RESOURCE PROVISIONING

This section describes how a service-oriented enterprise Grid platform called *Aneka* can implement SLA-based resource provisioning for an enterprise Grid using advanced reservations. An enterprise Grid [9] harnesses unused computing resources of desktop computers connected over an internal network or the Internet within an enterprise without affecting the productivity of their users. Hence, it increases the amount of computing resources available within an enterprise to accelerate application performance.

26.6.1 Design of Aneka

Aneka [10] is a .NET-based service-oriented platform for constructing enterprise Grids. It is designed to support multiple application models, persistence and security solutions, and communication protocols such that the preferred selection can be changed at any time without affecting an existing Aneka ecosystem. To create an enterprise Grid, the resource provider only needs to start an instance of the configurable Aneka container hosting required services on each selected desktop node. The purpose of the Aneka container is to initialize services, and to act as a single point for interaction with the rest of the enterprise Grid.

Figure 26.9 shows the design of the Aneka container on a single desktop node. To support scalability, the Aneka container is designed to be lightweight by providing the bare minimum functionality needed for an enterprise Grid node. It provides the base infrastructure that consists of services for persistence, security (authorization,

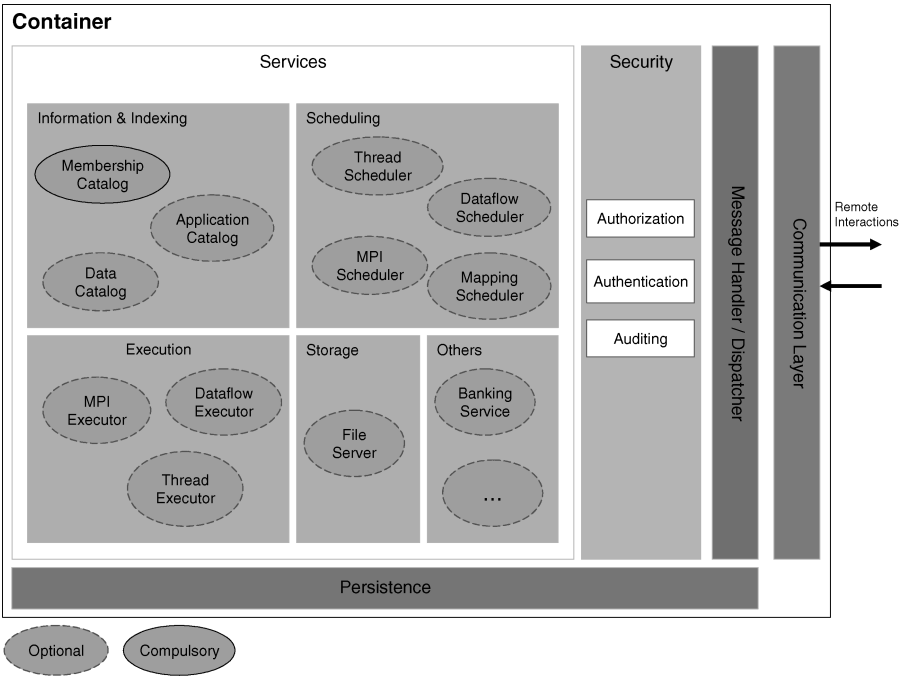


Figure 26.9 Design of Aneka container.

authentication, and auditing), and communication (message handling and dispatching). Every communication between Aneka services is treated as a message, handled and dispatched through the message handler/dispatcher that acts as a frontend controller. The Aneka container hosts a compulsory *membership catalog service*, which maintains the resource discovery indices (such as a .NET remoting address) of services currently active in the system.

The Aneka container can host any number of optional services that can be added to augment the capabilities of an enterprise Grid node. Examples of optional services are indexing, scheduling, execution, and storage services. This provides a single, flexible, and extensible framework for orchestrating different kinds of Grid application models.

To support reliability and flexibility, services are designed to be independent of each other in a container. A service can interact with other services only on the local node or other nodes through known interfaces. This means that a malfunctioning service will not affect other working services and/or the container. Therefore, the resource provider can seamlessly configure and manage existing services or introduce new ones into a container.

Aneka thus provides the flexibility for the resource provider to implement any network architecture for an enterprise Grid. The implemented network architecture depends on the interaction of services among enterprise Grid nodes since each Aneka container on a node can directly interact with other Aneka containers reachable on the

network. An enterprise Grid can have a decentralized network architecture peering individual desktop nodes directly, a hierarchical network architecture peering nodes in the hierarchy, or a centralized network architecture peering nodes through a single controller.

26.6.2 Resource Management Architecture

Figure 26.10 shows the interaction between the user/broker, the master node, and execution nodes in an enterprise Grid with centralized network architecture. *Centralized network architecture* means that there is a single master node connecting to multiple execution nodes. To use the enterprise Grid, the resource user (or broker acting on its behalf) has to first make advanced reservations for resources required at a designated time in the future.

During the request reservation phase, the user/broker submits reservation requests through the reservation service at the master node. The reservation service discovers available execution nodes in the enterprise Grid by interacting with the allocation service on them. The allocation service at each execution node keeps track of all reservations that have been confirmed for the node and can thus check whether a new request can be satisfied.

By allocating reservations at each execution node instead of at the master node, computation overheads that arise from making allocation decisions are distributed across multiple nodes and thus minimized, as compared to overhead accumulation at a single master node. The reservation service then selects the required number of execution nodes and informs their allocation services to temporarily lock the reserved timeslots. After all the required reservations on the execution nodes have been temporarily locked, the reservation service feeds back the reservation outcome and its price (if successful) to the user/broker.

The user/broker may confirm or reject the reservations during the confirm reservation phase. The reservation service then notifies the allocation service of selected execution nodes to lock or remove temporarily locked timeslots accordingly.

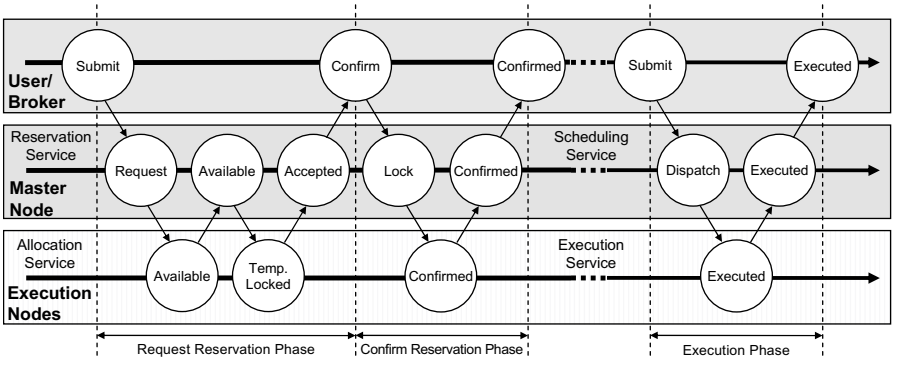


Figure 26.10 Interaction of enterprise Grid nodes.

We assume that a payment service is in place to ensure that the user/broker has sufficient funds and can successfully deduct the required payment before the reservation service proceeds with the final confirmation.

During the execution phase when the reserved time arrives, the user/broker submits applications to be executed to the scheduling service at the master node. The scheduling service determines whether any of the reserved execution nodes are available before dispatching applications to them for execution; otherwise applications are queued to wait for the next available reserved execution nodes. The execution service at each execution node starts executing an application after receiving it from the scheduling service and updates the scheduling service of changes in execution status. Hence, the scheduling service can monitor executions for an application and notify the user/broker on completion.

26.6.3 Allocating Advanced Reservations

Figure 26.11 shows that the process of allocating advanced reservations occurs in two levels: the allocation service at each execution node and the reservation service at the master node. Both services are designed to support pluggable policies so that the resource provider has the flexibility to easily customize and replace existing policies for different levels and/or nodes without interfering with the overall resource management architecture.

The allocation service determines how to schedule a new reservation at the execution node. For simplicity, the allocation service at each execution node can

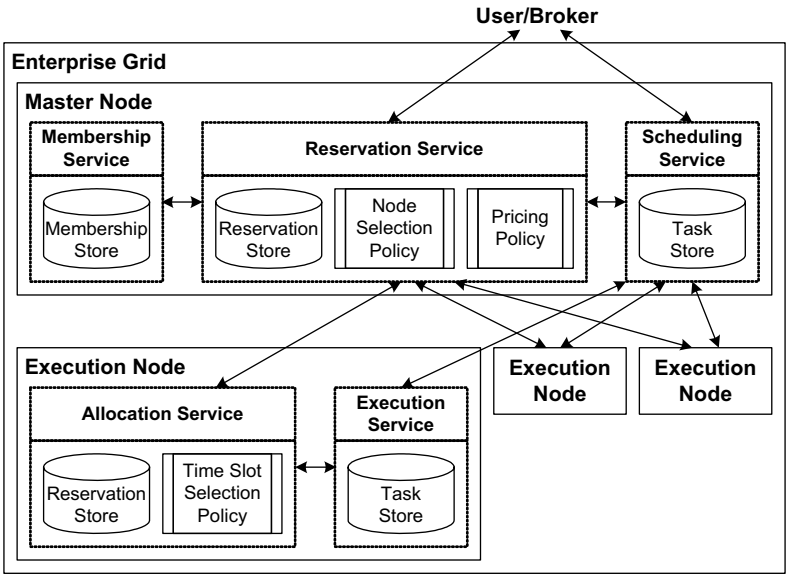


Figure 26.11 Interaction of services in enterprise Grid.

implement the same timeslot selection policy. The allocation service allocates the requested timeslot if the slot is available. Otherwise, it assigns the next available timeslot after the requested start time that can meet the required duration.

The reservation service performs node selection by choosing the required number of available timeslots from execution nodes and administers admission control by accepting or rejecting a reservation request. It also calculates the price for a confirmed reservation on the basis of the implemented pricing policy. Various pricing policies may be implemented. Available timeslots are selected with respect to the application requirement of the user.

The application requirement considered is the task parallelism to execute an application. A sequential application has a single task and thus needs a single processor to run, while a parallel application needs a required number of processors to concurrently run at the same time.

For a sequential application, the selected time slots need not have the same start and end times. Hence, available timeslots with the lowest prices are selected first. If there are multiple available timeslots with the same price, then those with the earliest start time are selected first. This ensures that the cheapest requested timeslot is allocated first if it is available. Selecting available timeslots with the lowest prices first is fair and realistic. In reality, reservations that are confirmed earlier enjoy the privilege of cheaper prices, as compared to reservation requests that arrive later.

However, for a parallel application, all the selected timeslots must have the same start and end times. Again, the earliest timeslots (with the same start and end times) are allocated first to ensure that the requested time slot is allocated first if available. If there are more available timeslots (with the same start and end times) than the required number of timeslots, then those with the lowest prices are selected first.

The admission control operates according to the service requirement of the user. The service requirements examined are the deadline and budget to complete an application. We assume that both deadline and budget are hard constraints. Hence, a confirmed reservation must not end after the deadline and cost more than the budget. Therefore, a reservation request is not accepted if there is an insufficient number of available timeslots on execution nodes that end within the deadline and if the total price of the reservation costs more than the budget.

26.6.4 Performance Evaluation

Figure 26.12 shows the enterprise Grid setup used for performance evaluation. The enterprise Grid contains 33 personal computers (PCs) with 1 master node and 32 execution nodes located across three student computer laboratories in the Department of Computer Science and Software Engineering, The University of Melbourne. Synthetic workloads are created by utilizing trace data. The experiments utilize 238 reservation requests in the last 7 days of the SDSC SP2 trace (April 1998–April 2000) version 2.2 from Feitelson's Parallel Workloads Archive [15]. The SDSC SP2 trace from the San Diego Supercomputer Center (SDSC) (USA) is chosen because it

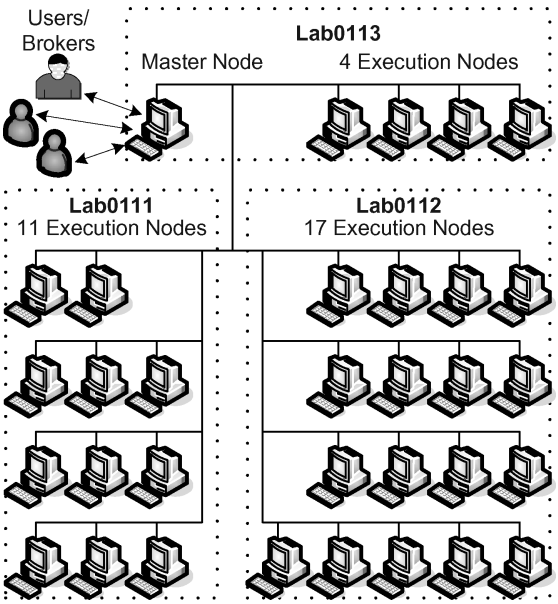


Figure 26.12 Configuration of Aneka enterprise Grid.

has the highest resource utilization (83.2%) among available traces to ideally model a heavy-workload scenario.

The trace only provides the interarrival times of reservation requests, the number of processors to be reserved as shown in Figure 26.13a (downscaled from a maximum of 128 nodes in the trace to a maximum of 32 nodes), and the duration to be reserved as shown in Figure 26.13b. However, service requirements are not available from this trace. Hence, we adopt a similar methodology [16] to synthetically assign service requirements through two request classes: (1) low-urgency and (2) high-urgency. Figures 26.13b and 26.13c show the synthetic values of deadline and budget for the 238 requests, respectively.

A reservation request i in the *low-urgency* class has a deadline of high $\text{deadline}_i/\text{duration}_i$ value and budget of low $\text{budget}_i/f(\text{duration}_i)$ value. $f(\text{duration}_i)$ is a function representing the minimum budget required on the basis of duration_i. Conversely, each request in the *high-urgency* class has a deadline of low $\text{deadline}_i/\text{duration}_i$ value and budget of high $\text{budget}_i/f(\text{duration}_i)$ value. This is realistic since a user who submits a more urgent request to be met within a shorter deadline offers a higher budget for the short notice. Values are normally distributed within each of the deadline and budget parameters.

We evaluate the performance of seven pricing mechanisms as listed in Table 26.4 for high-urgency reservation requests (with short deadline and high budget) from sequential applications (requiring one processor to execute) in the enterprise Grid. The enterprise Grid charges users only for utilizing the computing

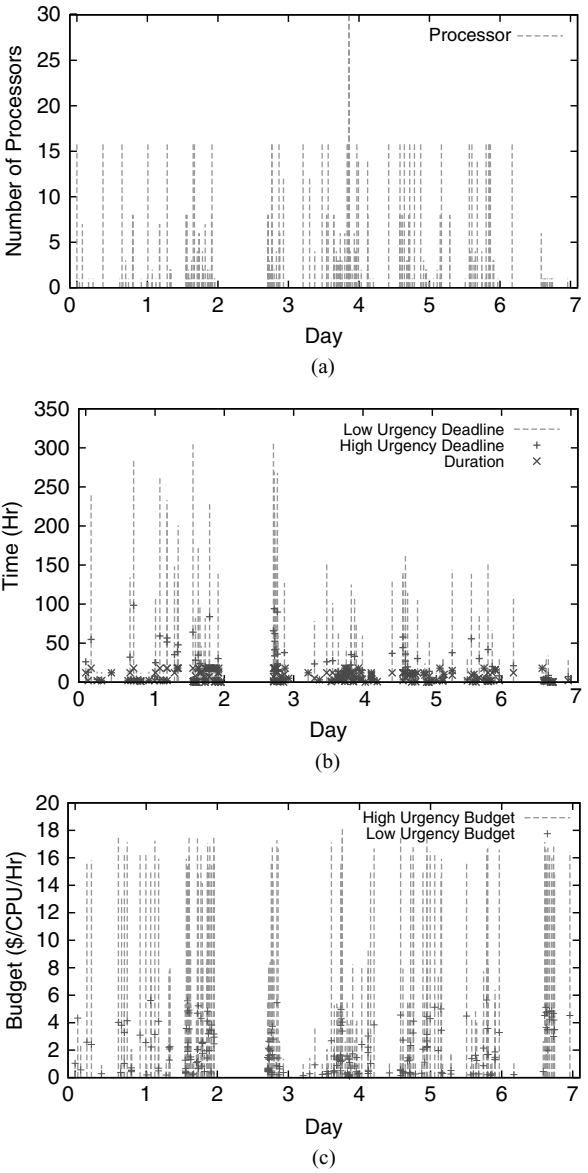


Figure 26.13 Last 7 days of SDSC SP2 trace with 238 requests: (a) number of processors (from trace); (b) duration (from trace) and deadline (synthetic); (c) budget (synthetic).

resource type on the basis of usage per processor (CPU) per hour (h). Thus, users are not charged for using other resource types such as memory, storage, and bandwidth. In addition, every user/broker can definitely accept another reservation timeslot proposed by the enterprise Grid if the requested one is not possible,

TABLE 26.4 Pricing Mechanisms

Name	Configured Pricing Parameters
`FixedMax`	\$3/(CPU·h)
`FixedMin`	\$1/(CPU·h)
`FixedTimeMax`	\$1/(CPU·h) (12 a.m.–12 p.m.)
	\$3/(CPU·h) (12 p.m.–12 a.m.)
`FixedTimeMin`	\$1/(CPU·h) (12 a.m.–12 p.m.)
	\$2/(CPU·h) (12 p.m.–12 a.m.)
`Libra + $Max`	\$1/(CPU·h) ($PBase_j$), $\alpha = 1$, $\beta = 3$
`Libra + $Min`	\$1/(CPU·h) ($PBase_j$), $\alpha = 1$, $\beta = 1$
`Libra + $Auto`	Same as `Libra + $Min`

provided that the proposed timeslot still satisfies both application and service requirements of the user.

The seven pricing mechanisms listed in Table 26.4 represent three basic types of pricing mechanism: (1) `Fixed`, (2) `FixedTime`, and (3) `Libra + $`. Table 26.4 lists the maximum and minimum types of each pricing mechanism, which are configured accordingly to highlight the performance range of the pricing mechanism. The `Fixed` mechanism charges a fixed price at all times. The `FixedTime` mechanism charges a fixed price for different time periods of resource usage where a lower price is charged for off-peak (12 a.m.–12 p.m.) and a higher price for peak (12 p.m.–12 a.m.).

`Libra + $` [17] uses a more fine-grained pricing function that satisfies four essential requirements for pricing of resources to prevent workload overload: (1) flexibility, (2) fairness, (3) being dynamic, and (4) being adaptive. The price P_{ij} for per unit of resource utilized by reservation request i at compute node j is computed as $P_{ij} = (\alpha * PBase_j) + (\beta * PUtil_{ij})$. The base price $PBase_j$ is a static pricing component for utilizing a resource at node \$j\$ that can be used by the resource provider to charge the minimum price so as to recover the operational cost. The utilization price $PUtil_{ij}$ is a dynamic pricing component that is computed as a factor of $PBase_j$ based on the utilization of the resource at node j for the required deadline of request i: $PUtil_{ij} = RESMax_j / RESFree_{ij} * PBase_j$. $RESMax_j$ and $RESFree_{ij}$ are the maximum units and remaining free units of the resource at node j for the deadline duration of request i, respectively. Thus, $RESFree_{ij}$ has been deducted units of resource committed for other confirmed reservations and request i for its deadline duration.

The factors α and β for the static and dynamic components of `Libra + $`, respectively, provides the flexibility for the resource provider to easily configure and modify the weightage of the static and dynamic components on the overall price P_{ij}. `Libra + $` is fair since requests are priced according to the amount of different resources utilized. It is also dynamic because the overall price of a request varies depending on the availability of resources for the required deadline. Finally, it is adaptive as the overall price is adjusted depending on the current supply and demand of resources to either encourage or discourage request submission.

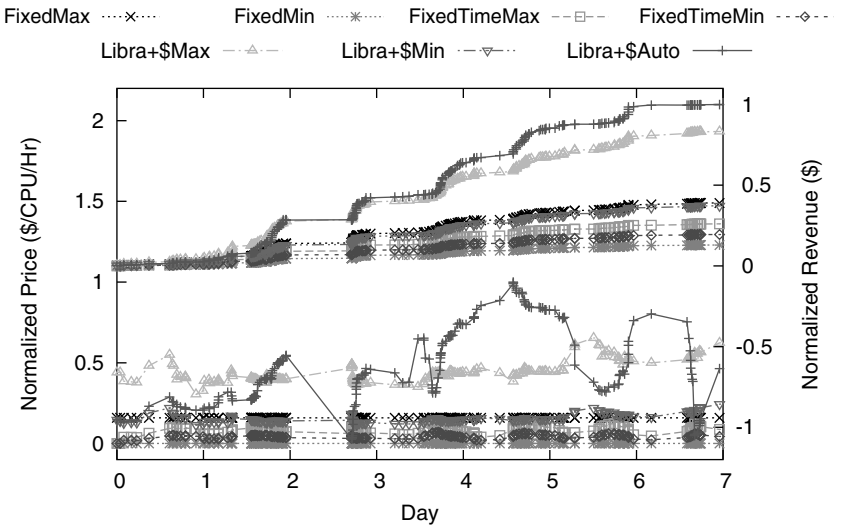


Figure 26.14 Price/revenue ratio of high-urgency requests.

However, these three mechanisms rely on static pricing parameters that are difficult to be accurately derived by the resource provider to produce the best performance where necessary. Hence, we propose `Libra+$Auto`, an autonomic `Libra+$` that automatically adjusts β per the availability of compute nodes. `Libra+$Auto` thus considers the pricing of resources across nodes, unlike `Libra+$`, which considers pricing of resources only at each node j via P_{ij}.

Figure 26.14 shows the performance results for the seven pricing mechanisms in an enterprise Grid for high-urgency requests from sequential applications over a 7-day time period that have been normalized to produce standardized values within the range of 0–1 for easier comparison. The performance metrics being measured are the price for a confirmed reservation [in $/(CPU·h)] and the accumulated revenue for confirmed reservations (in $). The revenue of a confirmed reservation is calculated using the assigned price (depending on the specific pricing mechanism) and reserved duration at each reserved node for all its reserved nodes. Then, the price of a confirmed reservation can be computed to reflect the average price across all its reserved nodes.

Of the four fixed pricing mechanisms listed in Table 26.4, `FixedMax` provides the highest revenue (maximum bound), followed by `FixedTimeMax`, `FixedTimeMin`, and `FixedMin` with the lowest revenue (minimum bound). Nevertheless, `FixedTime` mechanisms is easier to derive and more reliable than `Fixed` mechanisms since it supports a range of prices across various time periods of resource usage. However, all four mechanisms do not consider service requirements of users such as deadline and budget.

On the other hand, `Libra+$` charges a lower price for a request with longer deadline as an incentive to encourage users to submit requests with longer deadlines that are more likely to be accommodated than shorter deadlines. For a request with short deadline, `Libra+$Max` and `Libra+$Min` charge a higher price relative to

their β in Table 26.4. Libra + $Max provides higher revenue than Libra + $Min because of a higher value of β.

Both Libra + $Auto and Libra + $Max are able to provide a significantly higher revenue than other pricing mechanisms through higher prices for shorter deadlines. Figure 26.14 shows that Libra + $Auto continues increasing prices to higher than that of Libra + $Max and other pricing mechanisms when demand is high such as during the latter half of days 1, 2, 3, and 5. But when demand is low, such as during the early half of days 2, 3, 5, and 6, Libra + $Auto continues to reduce prices to lower than that of Libra + $Max to accept requests that are not willing to pay more. Hence, Libra + $Auto is able to exploit budget limits to achieve the highest revenue by automatically adjusting to a higher β to increase prices when the availability of nodes is low and to a lower β to reduce prices when there are more unused nodes that will otherwise be wasted.

26.7 GRID-FEDERATION

As enterprise Grids grow to include a large number of resources (on the order of thousands), the centralized model for managing the resource set does not prove to be efficient as it requires the manager to coordinate a large number of components and handle a large number of messages on its own. This means that the central coordinator does not scale well, lacks fault tolerance, and warrants expensive server hardware infrastructure. Since participants in a Grid can join and leave in a dynamic fashion, it is also an impossible task to manage such a network centrally. Therefore, there is a need for an efficient decentralized solution that can gracefully adapt and scale to the changing conditions. This can be achieved by partitioning the resource set into smaller installations that are then federated to create a single, cooperative, distributed resource-sharing environment [18–20]. In a federated organization, an enterprise domain can deal efficiently with bursty resource requests through policy-based or opportunistic leasing of resources from the resource pool. This basically relieves an enterprise domain from the responsibilities of maintaining and administering different kinds of resources and expertise within a single domain. This section postulates how a Grid-Federation can be engineered, including its primary components and how existing Gridbus middleware can be used to realize such an environment.

26.7.1 Characteristics of a Grid-Federation

The unique challenges in efficiently managing a federated Grid computing environment include the following characteristics:

- *Distributed ownership*—every participant makes decisions independently.
- *Open and dynamic*—the participants can leave and join the system at will.
- *Self-interested*—each participant has distinct stakeholdings with different aims and objective functions.

- *Large-scale*—composed of distributed participants (e.g., services, applications, users, providers) who combine together to form a massive environment.
- *Resource contention*—depending on resource demand pattern and lack of cooperation among distributed users, a particular set of resources can be swamped with excessive workload, which significantly reduces the amount of useful utility that the system delivers.

We perceive that by designing appropriate scalable protocols for cooperation among users, allowing users to express preferences for resources, and letting providers decide their allocation policies, it is possible to overcome the problem of resource contention, distributed ownership, large scale, and dynamism in a large-scale federated Grid system. Therefore, our design of a Grid-Federation focuses on two important aspects: a distributed resource discovery system [21,25] and a market-based resource allocation system [26]. Grid-Federation allows cooperative sharing of topologically and administratively distributed Grid resources. To enable policy-based transparent resource sharing between resource domains, Grid-Federation instantiates a new RMS, called *Grid-Federation agent* (GFA). A GFA exports a resource site to the federation and is responsible for undertaking activities related to resource sharing, selection, and reporting. GFAs in the system interconnect using a *distributed hash table* (DHT) overlay [22–24], which makes the system scalable and decentralized. The Grid-Federation considers computational economy driven SLA negotiation protocol for enforcing cooperation and establishing accountability among the distributed participants (e.g., providers, users, schedulers) in the system.

We are realizing the Grid-Federation resource sharing model within the Aneka system by implementing a new software service, called *Aneka Coordinator*. The Aneka Coordinator basically implements the resource management functionalities and resource discovery protocol specifications defined by the GFA service. An *Aneka-Federation* integrates numerous small-scale Aneka desktop Grid services and resources that are distributed over multiple control and administrative domains as part of a single coordinated resource leasing abstraction. The software design of the Aneka-Federation system decouples the fundamental decentralized interaction of participants from the resource allocation policies and the details of managing a specific Aneka service.

26.7.2 Resource Discovery

The distributed resource discovery service in the Grid-Federation allows GFAs to efficiently search for available resources that match the user's expressed QoS parameters. The resource discovery service [25] organizes the information by maintaining a logical multidimensional publish/subscribe index over a DHT overlay [22–24] of GFAs (refer to Fig. 26.15). In general, a GFA service undertakes two basic types of queries [21]: (1) a *resource lookup query* (RLQ)—a query issued by a GFA service to locate resources matching the user's application QoS requirement and (2) a *resource update query* (RUQ), which is an update query sent to a resource discovery by a GFA (on behalf of the Grid site owner) about the underlying resource

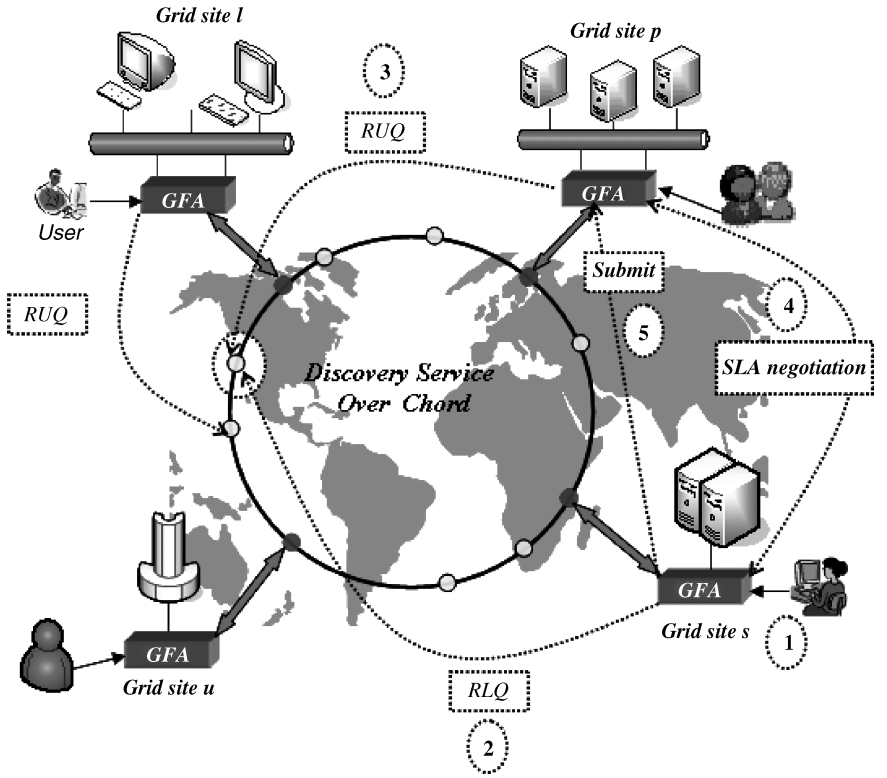


Figure 26.15 Grid-Federation – GFAs and Grid sites over Chord overlay. Dark dots indicate GFA services that are currently part of the Chord-based Grid network. Light dots represent the RUQ and RLQ objects posted by GFAs in the system.

conditions. Since a Grid resource is identified by more than one attribute, a RLQ or RUQ is always multidimensional.

Further, both these queries can specify different kinds of constraints on the attribute values depending on whether the value is a point or range query. A *point search query* specifies a fixed value for each resource attribute [e.g., `cpu_type = intel, processor_count = 50, price = 7` (Grid dollars/h)]. On the other hand, a range search query specifies a range of values for attributes (`e.g. cput_-type = intel` or `sparc, 50 < processor_count < 100, 5 < price < 10`). Currently, the resource discovery allows users to search for resources based on both point- and range-specifying RLQs. The providers can update the status (e.g, resource utilization, price, queue size, completion rate) with the service through point RUQs.

Because resources are dynamic, and can exhibit changing temporal characteristics, the providers can periodically update their status with the resource discovery service through RUQs. The mapping of RLQ and RUQ to the DHT-based overlay is accomplished through a multidimensional publish/subscribe index. The index

builds a multidimensional Cartesian space based on the Grid resource attributes. The logical index assigns regions of space [30] to GFAs in the resource discovery system. If a GFA is assigned a region in the multidimensional space, then it is responsible for handling all the activities related to RLQs and RUQs associated with that region.

Further, we extend the functionality of the resource discovery service to support an abstraction of peer-to-peer coordination/cooperation space [28], wherein the users, providers, and marketmakers cooperate their activities. The peer-to-peer coordination space acts as a kind of blackboard system that can be concurrently and associatively accessed by all participants in the federation.

In the context of the Aneka-Federation software system, the responsibility for decentralized resource discovery and coordination is undertaken by the Aneka peer service. The dynamic resource and scheduling information routing in Aneka-Federation is facilitated by the FreePastry[1] structured peer-to-peer routing substrate. FreePastry offers a generic, scalable, and efficient peer-to-peer routing substrate for development of decentralized Grid services. The FreePastry routing substrate embeds a logical publish/subscribe index for distributing the load of query processing and data management among Aneka peers in the system.

26.7.3 Resource Market

Grid-Federation considers computational economy as the basis for enforcing distributed cooperation among the participants, who may have conflicting needs. Computational economy promotes efficiency by allocating a resource to its best use, giving incentives to resource providers for contributing their resources to the federation, and promoting further long-term investments in new hardware and software infrastructure by resource providers as a result of the economic gains that they receive from the system.

Grid-Federation applies a decentralized commodity market model for efficiently managing the resources and driving the QoS-based scheduling of applications. In the commodity market model, every resource has a price, which is based on the demand, supply, and value. A resource provider charges a unit of virtual or real currency, called *access cost*, to the federation users for letting them use his/her resources. All federation users express how much they are willing to pay, called a *budget*, and required response time, called a *deadline*, on a per-job basis. The providers and users maintain their virtual or real credits with accounting systems such as GridBank. The Grid-Federation scheduling method considers the following optimizations with respect to the economic efficiency of the system: (1) resource provider's objective function (e.g., incentive) and (2) user's perceived QoS constraints (e.g., budget and deadline).

Realizing a true cooperative resource-sharing mechanism between dynamic and distributed participants warrants robust protocols for coordination and negotiations. In decentralized and distributed federated Grid environments, these coordination

[1]See http://freepastry.rice.edu/FreePastry/.

and negotiation protocols can be realized through dynamic resource information exchanges between Grid brokers and site-specific resource managers (such as PBS, Alchemi, and SGE). Grid-Federation utilizes one such SLA-based coordination and negotiation protocol [27], which includes the exchange of QoS enquiry and QoS guarantee messages between GFAs. These QoS constraints include the job response time and budget spent. Inherently, the SLA is the guarantee given by a resource provider to the remote site job scheduler (such as GFA and resource broker) for completing the job within the specified deadline and agreed-on budget.

A SLA-based job scheduling approach has several significant advantages: (1) promotes cooperation among participants; (2) it inhibits schedulers from swamping a particular set of resources; (3) once a SLA is finalized, users are certain that agreed QoS shall be delivered by the system; (4) job queuing and processing delay are significantly reduced, thus leading to enhanced QoS; and (5) it gives every site in the system enhanced autonomy and control over local resource allocation decisions.

Our SLA model considers a collection of resource domains in the Grid-Federation as a contract-net. As jobs arrive, GFAs undertake one-to-one contract negotiation with the other GFAs that match the resource configuration requirements of the submitted job. Each GFA becomes either a manager or a contractor. The GFA to which a user submits a job for processing is referred to as the *manager GFA* (*scheduler GFA*). The manager GFA is responsible for successfully scheduling the job in the federated contract-net. The GFA, which accepts the job from the manager GFA and overlooks its execution, is referred to as the *contractor GFA* (*allocator GFA*). Individual GFAs are assigned these roles in advance. The role may change dynamically over time as per the resource management requirement, namely, scheduling or allocation. A GFA alternates between these two roles or adheres to both over the processes of scheduling and resource allocation.

The general Grid-Federation scheduling and resource allocation technique operates as follows. In Figure 26.15, a user who has membership to Grid site *s* submits her application to its local GFA (see step 1 in Fig. 26.15). Following this, the GFA at site *s* adheres to the role of manager GFA and submits a RLQ object to the Chord-based resource discovery service (refer to step 2 in Fig. 26.15). Consequently, the GFA at site *p* reports or updates its resource availability status by sending a RUQ object to the discovery service (shown as step 3 in Fig. 26.15). As the posted RUQ object matches the resource configuration currently searched by GFA at site *s*, the discovery service notifies the GFA accordingly.

Following this, the GFA at site *s* undertakes one-to-one SLA negotiation (refer to step 4 in Fig. 26.15) with the GFA at site *p* (contractor GFA) about possible allocation of its job. If site *p* has too much load and cannot complete the job within the requested SLA constraints (deadline), then a SLA fail message is sent back to the GFA at site *s*. In this case, the GFA at site *s* waits for future match notifications. Alternatively, if GFA at site *p* agrees to accept the requested SLA, then the manager GFA goes ahead and deploys its job at site *p* (shown as step 5 in Fig. 26.15). The one-to-one SLA-based negotiation protocol guarantees that (1) no resource in the federation would be swamped with excessive load and (2) users obtain an acceptable or requested level of QoS delivered for their jobs.

26.7.4 Performance Evaluation

We present an evaluation of the Grid-Federation system through a set of simulated experiments designed to test the performance of resource discovery and resource market services with regards to efficiency, scalability, and usability. We realize the simulation infrastructure by combining two discrete-event simulators: GridSim [31] and PlanetSim [32]. GridSim offers a concrete base framework for simulation of different kinds of heterogeneous resources, services, and application types. On the other hand, PlanetSim is an event-based overlay network simulator that supports routing of messages using well-known DHT methods, including Chord and Symphony. Next, we describe the simulation environment setup, including peer-to-peer network configuration, resource configuration, and workload.

The experiments run a Chord overlay with a 32-bit configuration, specifically, the number of bits utilized to generate GFA and key (RLQ and RUQ object) IDs. The Grid-Federation network includes 100 Grid resource domains. The Grid network processes 500 messages per second and can queue up to 10,000 messages at any given instance of time. GFAs inject RLQ and RUQ objects based on the exponential interarrival time distribution. The value for RLQ interarrival delay is distributed over [60,600] in steps of 120 s. GFAs update their host Grid site status after a fixed interval of time. In this study, we configure the RUQ interarrival delay to be 120 and 140 s.

Both RLQ and RUQ objects represent a Grid resource in a five-dimensional attribute space. These attribute dimensions include the number of processors, their speed, their architecture, operating system type, and resource access cost (price). The distributions for these resource dimensions are obtained from the Top 500 supercomputer list.[2] We assume that the resource access cost does not change during the course of simulation. Resource owners decide the access cost on the basis of a linear function whose slope is determined by the access cost and processing speed of the fastest resource in the federation. In other words, every resource owner charges a cost relative to the one offered by the most efficient resource in the system. The fastest Grid owner in the federation charges 6.3 Grid dollars/per hour for providing space for shared access to his/her resources. We generate the workload distributions across GFAs according to the model given by Lublin and Feitelson [29]. The processor count for a resource is fed to the workload model based on the resource configuration obtained from the Top 500 list.

26.7.5 Results and Discussion

To measure the Grid-Federation system performance, we use metrics such as resource discovery delay, response time on per-job basis, and total incentive earned by providers as a result of executing local and remote jobs of the federation users. The response time for a job summarizes the latencies for (1) a RLQ object to be mapped to the appropriate peer in the network per the distributed indexing logic, (2) waiting time until a RLQ object is hit by a RUQ object, (3) the SLA negotiation delay between the manager and contractor GFA, and (4) the actual execution time on the remote site machine.

[2]See http://www.top500.org/.

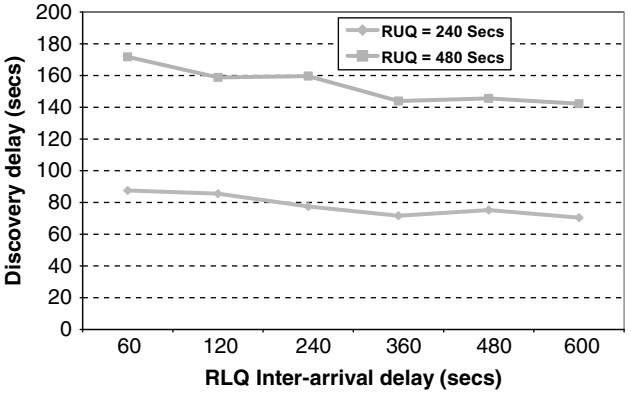


Figure 26.16 Average RLQ interarrival delay (secs) versus discovery delay (in seconds).

Figure 26.16 depicts the results of average resource discovery delay in seconds with increasing mean RLQ interarrival delay for different resource status update intervals (RUQ delay). The results show that at a higher RUQ update interval, with a large number of competing requests (high RLQ rate), the users have longer waiting time with regard to discovering resources that can satisfy their QoS metrics. The main reason behind this system behavior is that the RLQ objects for jobs have to wait for a longer time before they are hit by RUQ objects, because of the large number of competing requests in the system. Specifically, the distributed RLQ–RUQ match procedure also accounts for the fact that the subsequent allocation of jobs to resources should not lead to contention problems. Hence, with a large number of competing requests and infrequent resource update events, jobs are expected to suffer longer delay.

In Figure 26.17, we show the total incentive (in Grid dollars) earned by all providers in the federation. The providers earned almost similar incentive with varying rates of RLQ and RUQ objects, which is expected as we consider a static

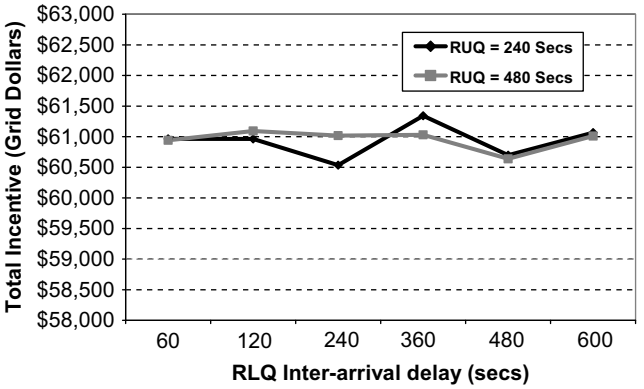


Figure 26.17 Average RLQ interarrival delay (in seconds) versus total incentive (in Grid dollars).

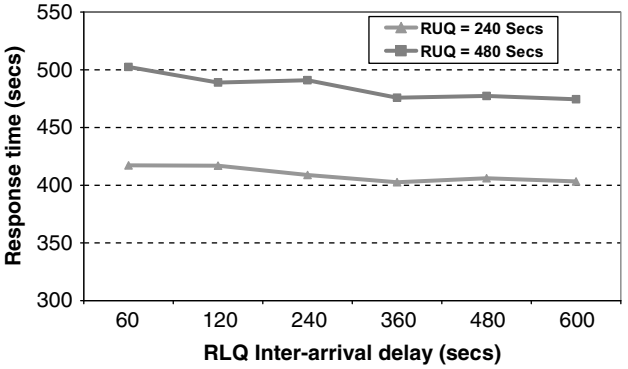


Figure 26.18 Average RLQ interarrival delay (in seconds) versus response time (in seconds).

resource access cost for the entire simulation period. However, the providers can dynamically vary their resource access cost with respect to the supply and demand in the federation. We intend to investigate this aspect of the system as part of our future work.

Figure 26.18 shows the average response time utility derived for federation users according to the resources they request and receive. The result shows that growth in the response time function for a user's job is similar to that for the resource discovery delay functions with varying RLQ and RUQ rates. For fixed RUQ rate, the result shows that at high RLQ interarrival delay, the jobs in the system face comparatively low resource discovery delay.

The main argument for this behavior is that under these settings, the RLQ objects encounter less network traffic and competing requests, which lead to an overall decrease in the discovery delay across the system.

26.8 CONCLUSION AND FUTURE DIRECTIONS

We have presented an overview of the Gridbus toolkit for service-oriented Grid and utility computing based on computational economy. The Gridbus project is actively pursuing the design and development of next-generation computing systems and fundamental Grid technologies and algorithms driven by Grid economy for data and utility Grid applications.

From a resource provider's perspective, appropriate market-based Grid resource management strategies that encompass both customer-driven service management and computational risk management are required in order to maximize the provider's profitmaking ability. Supporting customer-driven service management on the basis of customer profiles and requested service requirements is a critical issue since customers generate the revenue for providers in a Grid service market and have different needs. Many service quality factors can influence customer satisfaction, such as providing personalized attention to customers and encouraging

trust and confidence in customers. Therefore, a detailed understanding of all possible customer characteristics is essential to address customer-driven service management issues. In addition, defining computational risk management tactics for the execution of applications with regard to service requirements and customer needs is essential. Various elements of Grid resource management can be perceived as risks, and hence risk management techniques can be adopted. However, the entire risk management process consists of many steps and must be studied thoroughly so as to fully apply its effectiveness in managing risks. The risk management process consists of the following steps: (1) establish the context; (2) identify the risks involved; (3) assess each of the identified risks; (4) identify techniques to manage each risk; and (5) finally, create, implement, and review the risk management plan. In the future, we expect to implement such a process into Aneka's resource management system so that it becomes more capable as a resource provisioning system.

Within a market-oriented Grid, consumers have to locate providers that can satisfy the application requirements within their budget constraints. They may prefer to employ resource brokers that are optimized toward satisfying a particular set of requirements (e.g., a time-constrained workflow execution) or a particular set of constraints (e.g., the most cost-effective workflow executions). In such cases, brokers have to predict capacity requirements in advance and form agreements with resource providers accordingly. The nature and form of Grid markets are still evolving, and researchers are experimenting with new mechanisms and protocols. Brokers may have to participate in different markets with different interaction protocols. Brokers may also eventually have their own utility functions depending on which they will accept user requests. Therefore, it can be said that future Grid brokers will require capabilities for negotiation and decisionmaking that are far beyond what today's brokers can support. We expect to provide such capabilities in the Gridbus broker, thereby enhancing it to function as an equal participant in future Grid markets. To this end, we will also apply results from research carried out in the intelligent agent community for these areas.

Markets strive for efficiency; therefore, it is imperative to have a communication bus that is able to disseminate information rapidly without causing message overload. It would be an interesting research topic to design and realize a completely decentralized auction mechanism, that has the potential to deliver a scalable market platform for dynamic interaction and negotiation among Grid participants. Such a mechanism would use existing research performed on decentralization in peer-to-peer networks. The auctioneers (resource owners) can advertise their items, auction types, and pricing information, while the buyers (resource brokers) can subscribe for the auctioned items. A resource provider can choose to hold the auctions locally or may distribute the work to a Grid marketmaker, which is also part of the peer-to-peer market system. We expect to extend our current work on peer-to-peer Grid-Federation to satisfy these requirements.

Composing applications for market-based Grids is radically different; therefore, we aim to investigate and develop algorithms, software framework, and middleware

infrastructure to assist developers in exploiting the potential of such Grids. In particular, we intend to develop Grid middleware services that have the abilities to (1) coordinate resource usage across the system on the basis of market protocols (self-configuring); (2) interconnect participants (marketmakers, auctioneers, users) using on a decentralized overlay, such as a peer-to-peer network (self-organizing); (3) scale gracefully to a large number of participants; (4) make applications adapt to dynamic market, resource, and network conditions (self-managing applications); (5) take into account the application scheduling and resource allocation policy (pricing, supply, and demand) heterogeneity (self-optimizing); and (6) gracefully and dynamically adapt to the failure of resources and network conditions (self-healing). In this manner, applications and systems are expected to be autonomic, that is, run with minimal intervention from humans.

The Gridbus project is continuously enhancing and building on the various Grid technologies presented in this chapter. The project is also actively investigating and developing new Grid technologies such as the Grid Exchange, which enable the creation of a Stock Exchange–like Grid computing environment. For detailed and up-to-date information on Gridbus technologies and new initiatives, please visit the project Website: http://www.gridbus.org.

ACKNOWLEDGMENTS

This project was partially funded by Australian Research Council (ARC) and the Department of Innovation, Industry, Science and Research (DIISR) under Discovery Project and International Science Linkage grants, respectively. We would like to thank all members of the Gridbus project for their contributions. This chapter is partially derived from earlier publications [3–7,25,26].

REFERENCES

1. A. Rubinstein, Perfect equilibrium in a bargaining model, *Econometrica* **50**(1): 97–109 (1982).
2. R. Buyya, D. Abramson, and J. Giddy, A case for economy Grid architecture for service-oriented Grid computing, *Proc. 10th Heterogeneous Computing Workshop (HCW 2001): 15th International Parallel and Distributed Processing Symp. (IPDPS 2001)*, San Francisco, CA, April 23–27, 2001.
3. R. Buyya, D. Abramson, and S. Venugopal, The Grid economy, *Proceedings of the IEEE* **93**(3): 698–714 (2005).
4. B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, Data management and transfer in high-performance computational grid environments, *Parallel Computing* **28**(5): 749–771 (2002).
5. S. Venugopal, R. Buyya, and L. Winton, A Grid service broker for scheduling e-science applications on global data Grids, *Concurrency and Computation: Practice and Experience* **18**(6): 685–699 (2006).

6. J. Yu, S. Venugopal, and R. Buyya, A market-oriented Grid directory service for publication and discovery of Grid service providers and their services, *The Journal of Supercomputing* **36**(1): 17–31 (2006).
7. A. Barmouta and R. Buyya, GridBank: A Grid accounting services architecture (GASA) for distributed systems sharing and integration, *Proc. 3rd Workshop on Internet Computing and E-Commerce (ICEC 2003), 17th International Parallel and Distributed Processing Symp. (IPDPS 2003)*, Nice, France, April 22–26, 2003.
8. D. A. Reed, Grids, the TeraGrid, and beyond, *Computer* **36**(1): 62–68 (2003).
9. A. Chien, B. Calder, S. Elbert, and K. Bhatia, Entropia: Architecture and performance of an enterprise desktop Grid system, *Journal of Parallel and Distributed Computing* **63**(5): 597–610 (2003).
10. X. Chu, K. Nadiminti, C. Jin, S. Venugopal, and R. Buyya, Aneka: Next-generation enterprise Grid platform for e-science and e-business applications, *Proc. 3th IEEE International Conf. e-Science and Grid Computing (e-Science 2007)*, Bangalore, India, Dec. 10–13, 2007.
11. I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organizations, *International Journal of High-Performance Computing Applications* **15**(3): 200–222 (2001).
12. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, A directory service for configuring high-performance distributed computations, *Proc. 6th IEEE Symp. High Performance Distributed Computing (HPDC 1997)*, Portland, OR, Aug. 5–8, 1997.
13. L. Camarinha-Matos and H. Afsarmanesh, eds., *Infrastructures for Virtual Enterprises: Networking Industrial Enterprises*, Kluwer Academic Press, 1999.
14. R. Buyya and S. Vazhkudai, Compute power market: Towards a market-oriented grid, *Proc. 1st IEEE/ACM International Symp. Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, May 15–18, 2001.
15. Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/, May 23, 2008.
16. D. E. Irwin, L. E. Grit, and J. S. Chase, Balancing risk and reward in a market-based task service, *Proc. 13th IEEE International Symp. High Performance Distributed Computing (HPDC 2004)*, Honolulu, HI, June 4–6, 2004.
17. C. S. Yeo and R. Buyya, Pricing for utility-driven resource management and allocation in clusters, *International Journal of High-Performance Computing Applications* **21**(4): 405–418 (2007).
18. R. Ranjan,*Coordinated Resource Provisioning in Federated Grids*, PhD thesis, Univ. Melbourne, Australia, July 2007.
19. N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, OurGrid: An approach to easily assemble Grids with equitable resource sharing, *Proc. 9th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2003), LNCS 2862/2003*, Seattle, WA, June 24, 2003.
20. D. Irwin, J. Chase, L. Grit, A. Yumerefendi, and D. Becker, Sharing networked resources with brokered leases, *Proc. 2006 Usenix Annual Technical Conf. (Usenix 2006)*, Boston, MA, May 30–June 3, 2006.
21. R. Ranjan, A. Harwood, and R. Buyya, Peer-to-peer resource discovery in global Grids: A tutorial, *IEEE Communication Surveys and Tutorials* **10**(2): 6–33 (2008).

22. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications, *Proc. 2001 ACM SIGCOMM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2001)*, San Diego, CA, Aug. 27–31, 2001.
23. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, A scalable content-addressable network, *Proc. 2001 ACM SIGCOMM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication*, San Diego, CA, Aug. 27–31, 2001.
24. A. Rowstron and P. Druschel, Pastry: Scalable, decentralized object location, and routing for large scale peer-to-peer systems, *Proc. 3rd IFIP/ACM International Conf. Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, Nov. 12–16, 2001.
25. R. Ranjan, L. Chan, A. Harwood, S. Karunasekera, and R. Buyya, Decentralized resource discovery service for large scale federated Grids, *Proc. 3rd IEEE International Conf. e-Science and Grid Computing (e-Science 2007)*, Bangalore, India, Dec. 10–13, 2007.
26. R. Ranjan, A. Harwood, and R. Buyya, A case for cooperative and incentive-based federation of distributed clusters, *Future Generation Computing Systems* **24**(4): 280–295 (2008).
27. R. Ranjan, A. Harwood, and R. Buyya, SLA-based coordinated superscheduling scheme for computational Grids, *Proc. 8th IEEE International Conf. Cluster Computing (Cluster 2006)*, Barcelona, Spain, Sept. 25–28, 2006.
28. R. Ranjan, A. Harwood, and R. Buyya,*Coordinated Load Management in Peer-to-Peer Coupled Federated Grid Systems*, Technical Report GRIDS-TR-2008-2, Grid Computing and Distributed Systems Laboratory, Univ. Melbourne, Australia, 2008.
29. U. Lublin and D. G. Feitelson, The workload on parallel supercomputers: Modeling the characteristics of rigid jobs, *Journal of Parallel and Distributed Computing* **63**(11): 1105–1122 (2003).
30. E. Tanin, A. Harwood, and H. Samet, Using a distributed quadtree index in peer-to-peer networks, *The VLDB Journal* **16**(2): 165–178 (2007).
31. R. Buyya and M. Murshed, GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing, *Concurrency and Computation: Practice and Experience* **14**(13–15): 1175–1220 (Nov.–Dec. 2002).
32. P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo, PlanetSim: A new overlay network simulation framework, *Proc. 4th International Workshop on Software Engineering and Middleware (SEM 2004), Lecture Notes in Computer Science* **3437**: 20–21 (Sept. 2004).