12

SLA-BASED RESOURCE MANAGEMENT AND ALLOCATION

Jordi Guitart, Mario Macías, Omer Rana, Philipp Wieder, Ramin Yahyapour, and Wolfgang Ziegler

12.1 INTRODUCTION

The aim of the chapter is to describe how service-level agreements (SLAs) could be utilized to provide the basis for resource trading based on economic models. SLAs enable a service user to identify their requirements, and a provider to identify their capabilities. Subsequently, the terms in an SLA are necessary to ensure that mutually agreeable quality is being delivered by the provider according to the agreement. The use of service-level agreements (SLAs) in a resource management system to support Grid computing applications is described. To this end, we provide an architecture that supports the creation and management of SLAs. The architecture of the system, in terms of the components and their interactions, is first presented, followed by a description of the specific requirements for a market-oriented Grid economy. We use SLAs as a means to support reliable quality of service for Grid jobs. The creation of such an SLA requires planning and orchestration mechanisms. We will discuss these functionalities and also consider the economic aspects such as dynamic pricing and negotiation mechanisms. These mechanisms are necessary to enable SLA formation and use, and to ensure that an SLA is being adhered to during service provision.

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

12.2 BACKGROUND AND RELATED WORK

An SLA represents an agreement between a service user and a provider in the context of a particular service provision. An SLA may exist between two parties, for instance, a single user and a single provider, or between multiple parties, for example, a single user and multiple providers. SLAs contain certain quality-of-service (QoS) properties that must be maintained by a provider during service provision—generally defined as a set of service-level objectives (SLOs). These properties need to be measurable and must be monitored during the provision of the service that has been agreed in the SLA. The particular QoS attributes that are used must be preagreed to between the user and provider(s), before service provision begins, and also they define the obligations of the user/client when the provider meets the quality specified in the SLA. The SLA must also contain a set of penalty clauses when service providers fail to deliver the preagreed- to quality. Although significant work exists on how SLOs may be specified and monitored, not much work has focused on actually identifying how SLOs may be impacted by the choice of specific penalty clauses. The participation of a trusted mediator may be necessary in order to resolve conflicts between involved parties. Automating this conflict resolution process clearly provides substantial benefits. Different outcomes from such a process are possible. These include monetary penalties, impact on potential future agreements between the parties and the enforced rerunning of the agreed service. Market mechanisms provide an important basis for attributing the cost of meeting/violating an SLA. While it may seem reasonable to penalize SLA noncompliance, there are a number of concerns when issuing such penalties. For example, determining whether the service provider is the only party that should be penalized, or determining the type of penalty that must be applied to each party becomes a concern that needs to be considered.

12.3 SLA LIFECYLCE

As outlined in Chapter 10 of this book, an SLA goes through various stages within its lifecycle. Assuming that an SLA is initiated by a client application, these stages include the following:

- *Identifying the Provider.* This could either be "hardwired" (i.e., predetermined) or obtained through the use of a discovery (registry) service. Provider selection is an activity often outside the scope of the SLA lifecycle, but nevertheless an important stage to be executed.
- *Defining the SLA.* This stage involves identifying the particular terms that should be included in the SLA. These terms may relate to QoS issues that must subsequently be monitored, and also form the basis for penalty clauses. The definition also includes the period during which the SLA is valid.
- *Agreeing on the Terms of the SLA.* This stage involves identifying the constraints that must be met by a provider during service provisioning. A negotiation

process may be used to converge on such constraints. This stage would also involve identifying penalty clauses.

- *Provisioning and Execution.* This stage is responsible to use the previously agreed-on SLA and facilitate the practical provisioning. This stage involves the interaction with execution management services to setup resources and services accordingly.
- *Monitoring SLA Violations.* This stage involves monitoring the agreed-to terms and ensuring that they are not being violated. Who does the monitoring and how often is an aspect that needs to be considered at this stage.
- *Destroying SLAs.* Once a service provision has completed, the SLA must be destroyed.
- *Penalties for SLA Violation.* Once a service provision has completed, the monitoring data may be used to determine whether any penalties need to be imposed on the service provider.

12.3.1 Provider Identification Phase

The provider identification phase involves choosing possible partners to interact with. This may consist of a discovery phase, which involves searching a known registry (or a number of distributed registries) for providers that match some profile—generally using predefined metadata. The outcome of this stage is a list of providers (which may comprise only one provider) that offer the capability a client needs. Once a service provider (or multiple service providers) has been identified, the next stage involves defining the SLA between the client and the provider. The SLA may be between a single client and provider, or it may be between one client and multiple providers. In the subsequent analysis, we assume a two-party SLA (i.e., one involving a single client and a single provider).

12.3.2 Definition Phase

The definition of the SLA impacts the other stages in the SLA lifecycle—as the mechanisms used to identify particular service-level objectives (SLOs) will determine how violations will be identified in the future. Hence, an SLA may be defined using (name, value) pairs—where "name" refers to a particular SLO and "value" represents the requested quality/service level. An alternative is to use constraints that are more loosely defined—such as the use of (name, relationship, value) triples. In this context, provided the "relationship" between the "name" and "value" holds, the provider would have fulfilled the SLA requirements. Examples of relationships include "less than," "greater than," or a user-defined relationship function that needs to be executed by both the client and the provider. Other representation schemes have included the use of server-side functions—whereby an SLA is defined as a function $f(x_1, x_2, \ldots, x_n)$, where each (x_i) corresponds to a metric that is managed by the service provider. Using this approach, a client requests some capability from the service provider that is a function of what is available at the service provider. For instance, if the service provider has 512 GB of available memory at a particular point in time, the client requests 50% of this. In this context, $f(x)$ is evaluated according to the currently available capacity at the service provider [1]. An SLA must also be valid within some time period, a parameter that also needs to be agreed upon by the client and the provider.

12.3.3 Negotiation–Agreement Phase

Agreeing on SLA terms takes place once a description scheme has been identified. The next step is to identify the particular SLOs and their associated constraints. There needs to be some shared agreement on term semantics between the client and the provider. There is, however, no way to guarantee this, unless both the client and the provider use a common namespace (or term ontology), and therefore rely on the semantic definitions provided within this namespace. For example, the job submission description language (JSDL) [2] a proposed recommendation of the Open Grid Forum is often used to express SLOs related to computational resources.

Agreeing on SLO terms may be a multishot process between the two parties. This process can therefore be expressed through a “negotiation” protocol (a process requiring a provider to make an “offer” to the client, and the client then making a “counter offer”). The intention is to either reach convergence/agreement on SLOs—generally within some time bounds (or number of messages)—or indicate that the SLOs cannot be met. Also associated with an SLA must be the “penalty” terms that specify the compensation for the client if the SLA was not observed by the service provider. These penalty terms may also be negotiated between a client and a provider—or a fixed set of penalty terms may be used.

12.3.4 Provisioning and Execution Phase

The actual provisioning of services and resources on the basis of previously agreed-on SLAs is typically part of the underlying resource management systems. This part can be abstracted from the SLA management. The enablement can utilize existing systems like OGSA-Basic Execution Service (OGSA-BES) or Globus Resource Allocation Manager (GRAM) to facilitate the execution. This may require additional features to ensure that particular SLOs can be guaranteed.

12.3.5 Monitoring Phase

Once a service is executed according to an SLA, the compliance of the service provision with the SLA is monitored. A copy of the SLA must be maintained by both the client and the provider. It is necessary to distinguish between agreeing an SLA, and subsequently providing a service on the basis of the SLOs that have been agreed. A request to invoke a service based on the SLOs, for instance, may be undertaken at a time much later than when the SLOs were agreed. During provision it is necessary to determine whether the terms agreed in the SLA have been adhered to. In this context, the monitoring infrastructure is used to identify the difference between the agreed-on SLO and the value that was actually delivered during service provisioning—which is *trusted* by both the client and the provider. It is necessary to also define what constitutes a violation.

12.3.6 Termination Phase

An SLA may be destroyed/deleted in two situations: (1) when the service being defined in the SLA has completed and (2) when the time period over which the SLA has been agreed on has expired. In both cases, it is necessary for the SLA to be removed from both the client and the provider. Determining penalties for any SLA violation takes place after provisioning. Where an SLA was actually used to provision a service, it is necessary to determine whether any violations had occurred during provisioning. As indicated above, penalty clauses are also part of the SLA, and need to be negotiated between the client and the provider.

These stages demonstrate one cycle through the creation, use, and deletion of an SLA. Any violations detected in the SLA may be used in future cycles to choose between service providers. Clearly, the main parts of such a lifecycle need direct interaction with resource management functions. For instance, agreement and negotiation requires knowledge about the resource allocation plans in many application scenarios. The agreement might also require advance reservation of resources, which requires such capability to exist in resource management systems (RMSs) to establish an agreement. Similarly, the provisioning and termination needs enactment on the level of the resources. Considering the execution of a computational application as subject of the SLA, the SLA management needs some linkage to the underlying job management systems. In the following sections, we provide more details about the practical aspects of SLA-based resource management, and offer an architectural vision and methodology.

12.4 AN SLA-BASED RESOURCE MANAGEMENT–ALLOCATION ARCHITECTURE

12.4.1 Architectural Overview

As described above, a consumer and a provider use a predefined SLA to agree on certain QoS metrics. To achieve this, and then execute, monitor, and finalize any activity according to the SLA, a number of additional services are necessary. To ensure generality, we consider that several parties may be involved in the process of providing an SLA. This may include, besides consumers and providers, also brokers and intermediary SLA providers. In the following we consider a simplified model with only consumer and provider. Figure 12.1 shows a general SLA-based architecture with the necessary services. We use a classification in the following main areas:

- Consumer (or application)
- SLA negotiation and supervision
- Middleware services
- Provider
- Supporting services, which may be SLA-specific or more general in scope

We will discuss the individual entities in more detail in the subsequent paragraphs.

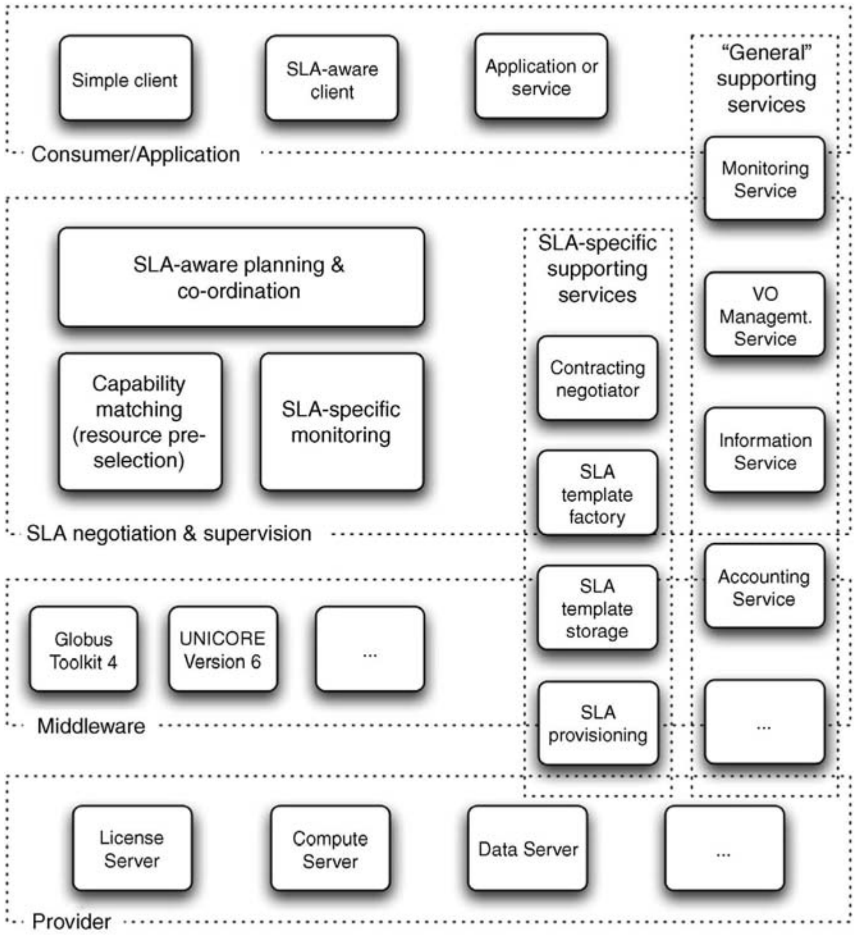

Figure 12.1 SLA-based resource management–allocation architecture.

12.4.2 Stakeholders and Services

12.4.2.1 Consumer View. Figure 12.1 shows different ways in which the client can interact with an SLA-based Grid: (1) SLA-aware services can access the SLA service layer directly through service calls, (2) customer services or applications can implement "SLA-specific" APIs to exploit the functionality of the SLA service layer, and (3) "simple" clients can be used without any SLA functionality in cases where it is not required. In general it cannot be expected that consumers directly deal with SLAs. Instead, they specify their requirements according to their business-level objectives. Those requirements are then included into an SLA for further processing, where characteristics and level of detail are dependent on the domain or the application respectively. Mapping such business-level objectives into SLAs remain a difficult and time-consuming process.

12.4.2.2 Provider View. The core business of a provider in a Grid is the provisioning of resources or services. In the context we consider here, this is achieved according to one or more SLOs which, a priori, have been agreed upon and captured in an SLA. In contrast to the consumer, who normally wants to deal mainly with application-specific requirements and not with SLAs, the provider has to "translate" the low-level technical details of what can be provided into SLA templates. That is, the user is interested in the high-level fulfillment of certain quality features, but may not necessarily know how these will be achieved. However, to achieve this goal, specific technical information may need to be included in the SLA. It is useful to note that both the providers and consumer operate according to their business-level objectives, which may differ substantially [3].

12.4.2.3 SLA Service Layer: Negotiation and Supervision. This layer covers the functionalities of finding suitable service providers (capability matching) and negotiating with them to establish usable SLA. On the basis of the requirements provided by the SLA consumers, the SLA service layer can automatically select SLA offers and plan the service allocation. These services include some limited scheduling functionality to coordinate the SLA. Different resource types can be considered in a similar fashion. Within this layer, the use of resources is considered in only a generic fashion, where different resource types can be modeled with different SLA templates. The selection is based on the information given in the SLA request to optimize a given preference. More sophisticated strategies are necessary to plan workflow-like SLA requests in which several different resources/services may be required. Such requirements may require combining several SLAs. This layer also includes SLA monitoring facilities in which the current SLAs are checked for fulfillment. This includes the timely identification of current and potential future SLA violations. According to data obtained from a monitoring service, the SLA service layer may optionally deploy automatic recovery mechanisms.

12.4.2.4 Middleware Layer. Although support for SLAs is provided in some existing middleware, such as GRIA [4], our architecture assumes "classical," non-SLA-enabled middleware, which is used mainly to submit, monitor, and control workflows or jobs. Such middleware is deployed at a majority of sites, with gLite [5], Globus [6], and UNICORE [7] as three of the more popular representatives, and it also provides the basis for a number of SLA-related developments such as the VIOLA metascheduling service [8] or ASKALON [9]. From the architecture in Figure 12.1, the middleware is the entity that executes the workflow or job once the consumer and provider have agreed on an SLA. The middleware provides command-line tools, Web service interfaces or APIs that enable services from the SLA service layer or the supporting services to access middleware functions. Additional services are needed for a real-life deployment of an SLA. This includes security-related services but also accounting functions. Such account and billing services are necessary for market-oriented Grids in which resource consumption is subject to cost.

12.4.2.4.1 SLA-Specific Supporting Services. This category includes services that provide support for SLA template repositories, such as factories for creation of new templates and services for storing templates. In addition, there is a need for initiating the provisioning of resources on the basis of constraints defined in the SLA.

12.4.2.4.2 General Supporting Services. This category of services comprises those that are often deployed in Grid infrastructures independent of the use of SLAs. This includes information and monitoring services about resources and jobs. In addition, there is need for security management and authorization support for virtual organizations, and support for accounting and billing.

12.4.3 Web Services Agreement

The Web Services Agreement (WS-Agreement) specification [10] from the Open Grid Forum (OGF) describes a protocol for establishing an agreement on the usage of services between a service provider and a consumer. Although other similar specifications exist (e.g., the Web Services Agreement Language [11]), one can observe that the use of WS-Agreement prevails in the Grid community [12].The specification defines a language and a protocol to represent the services of providers, create agreements based on offers, and monitor agreement compliance at runtime. An agreement defines a relationship between two parties that is dynamically established and dynamically managed. The objective of this relationship is to deliver a service by one of the parties. In the agreement each party agrees on the respective roles, rights, and obligations. A provider in an agreement offers a service according to conditions described in the agreement. A consumer enters into an agreement with the intent of obtaining guarantees on the availability of one or more services from the provider. Agreements can also be negotiated by entities acting on behalf of the provider and/or the consumer. An agreement creation process usually consists of three steps: (1) the initiator retrieves a template from the responder, who advertises the types of offers the responder is willing to accept; (2) the initiator then makes an offer, which is either accepted or rejected by the responder; and then (3) WS-Agreement–Negotiation, which sits on top of WS-Agreement, furthermore describes the re/negotiation of agreements. An agreement consists of the agreement name, its context, and the agreement terms. The context contains information about the involved parties and metadata such as the duration of the agreement. Agreement terms define the content of an agreement—service description terms (SDTs) define the functionality that is delivered under an agreement. A SDT includes a domain-specific description of the offered or required functionality (the service itself). *Guarantee terms* define assurance on service quality of the service described by the SDTs. They define SLOs, which describe the quality of service aspects of the service that have to be fulfilled by the provider. WS-Agreement allows the usage of any domain specific or standard condition expression language to define SLOs. The specification of domain-specific term languages is explicitly left open and subject to domain-specific standardization.

12.5 INTERACTION WITH GRID RESOURCE MANAGEMENT AND SCHEDULING

The use of SLAs is typically part of the Grid resource management and scheduling. In other words, if a Grid scheduling instance needs to select suitable resources for a Grid job, SLAs can be used for establishing well-defined quality of service. This happens on the higher level between a service provider and a service consumer and on the lower level between the Grid scheduler and a component of the respective local resource management system. The roles of service provider and service consumer may range from individuals to institutions, software agents, or other systems acting on behalf of physical entities. The “upper level” SLA (primarily focusing on business policy) often are based on general agreements, such as framework contracts that govern the relationship between the parties. These general agreements may include legal aspects and may set boundaries for SLAs, which have to be respected in the agreement templates. Today, the technology most frequently used is the aforementioned WS-Agreement [12].

12.5.1 Deducing SLA-Relevant Information from Provider Layer

Information from the provider layer is needed

- Before starting the dynamic creation of an SLA using a WS-Agreement factory, the provider has to be selected that is expected to have the capacity for the job.
- After the creation of the SLA both parties need information to verify whether the SLA is fulfilled or violated.

Moreover, with a large number of providers, the manual selection of resources is no longer feasible and will be delegated to a resource selection service (RSS) [13,14]. The RSS creates a shortlist of potentially appropriate resources and respective providers that the Grid-level scheduler then uses for SLA negotiation; naturally, this shortlist will be based on the information about the requirements of the application and information from the provider.

12.5.2 Retrieving Templates from a Template Storage

In our architecture, we consider SLA templates as a starting point for establishing an SLA. Providing templates for SLAs in template storage is a convenient approach for a service provider minimizing the overhead and complexity of negotiating individual SLAs on the fly. The approach using templates from template storage is similar to shopping in supermarkets where the goods tailored in advance to suit the expected demand of the customers—either the packaging unit does or does not meet the demand. Retrieving preconfigured templates from storage offers both the provider and the customer a simple way to create SLAs for frequently recurring patterns of resource usage. The customer selects a template appropriate for the task to be executed and sends the template to the provider as an agreement offer. The provider either rejects the offer or agrees and sends an endpoint reference (EPR) back to the customer. Depending on the template, the provider may additionally allow limited modifications of the template before the customer sends it to the provider as an offer. In this case the template also contains constraints on the types of modification that can be made, and allows the provider to easily check the validity of the customer's offer.

12.5.3 Accessing a Factory to Create a New Template

Creating templates on the fly using a factory increases the flexibility of both parties. For the consumer the template created dynamically may better reflect the actual resource requirements, the QoS, and the acceptable policy for a task. The service provider, on the other hand, has the possibility to react dynamically if important properties and conditions of the resource pool change. For example, more resources may have to be allocated to particular types of customers; furthermore, resources with new characteristics may enter the pool or may drop out of the pool because of failure.

The factory is maintained by the service provider and may be accessed via an EPR. In case consumers are allowed to access the factory to create templates, the EPR has to be published beforehand. The major drawback of using dynamically created templates is the additional effort for validating the SLOs defined in the templates "guarantee terms and penalties."

12.6 ADVANCE RESERVATION OF RESOURCES

12.6.1 Advance Reservation for Better Planning of Resource Usage

Submitting a job to a Grid resource usually results in inserting the job into a queue of the local resource management system for execution. This kind of service level is called *best effort*, indicating that there are no guarantees associated with the requested service. However, this mode of operation is not sufficient for many application scenarios. The benefit of advance reservation for the customer is the guarantee associated with the SLA that the required resources would be available for the execution of a job while the provider may better plan resource utilization.

12.6.2 Increasing Reliability

However, results of an application are often required at a certain time requiring resources to be made available at a fixed time, for example

- Because results of an application need to be available by a deadline
- To limit waiting time between the different tasks of a workflow
- To support interactive and collaborative applications

Advance reservation is just another QoS requirement and must be supported by the local resource management system(s). Providing guarantees with respect to availability of resources through advance reservation has proved to be an approach offering both the customer and the provider a number of advantages.

Until now the advance reservation of the required resources often was negotiated with the provider by mail or telephone. As the number of providers and resources in the Grid increases, doing this negotiation manually becomes tedious or unfeasible. The growing use of SLAs for expressing and negotiating SLOs gives new opportunities, and indeed SLAs play an important role for reserving resources in advance. Negotiating the availability of appropriate resources during a given timeslot is usually the task of a Grid scheduler—as this scheduler may connect to a number of local and remote resource management systems for the negotiation.

12.6.3 Enabling Resource Orchestration and Coallocation

Using advance reservation, more complex tasks may be performed efficiently:

- Coallocation of distributed applications that need multiple resources at the same time, for example, for multiple physics simulations such as fluid dynamics or structural mechanics [15].
- Orchestration of workflows with dependencies between the tasks distributed across multiple resources; for example, workflow tasks depending on results of the preceding tasks may be executed with smaller gaps between the individual tasks, resulting in smaller makespans [16], as depicted in Figure 12.2.
- Reserving additional types of resources like storage, network, and software licenses along with computational resources may further increase the efficiency and reliability of complex task execution in both coallocation and orchestration scenarios [17].

In addition to specifying the normal job requirements, a user now additionally specifies the date and time when the resources are needed and the duration of the usage.

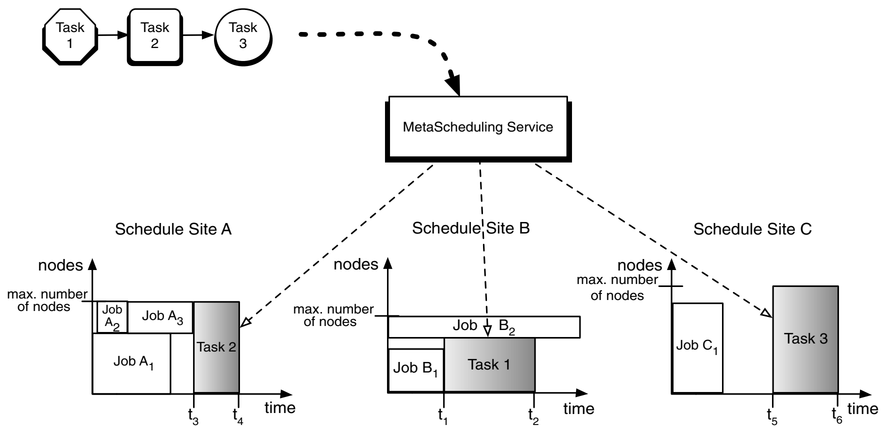

Figure 12.2 A Grid scheduler reserving slots for components of a workflow across three sites.

On receiving a request, the Grid scheduler begins negotiation for the availability of resources offering the required properties with the local resource management systems. Depending on whether the application is a single job requiring only one resource or a workflow-like application where multiple resources are needed, the negotiation process may need multiple iterations until all required resources are reserved [18].

12.6.4 Renegotiation of SLAs

Currently, SLAs once negotiated are static and may not be altered; canceling an SLA and negotiating a new one is the only way to modify an existing SLA. However, in many situations one of the parties may wish to modify an agreement, for example

- The provider has a higher interest in using assigned resources for another job.
- The user detects that the job will run longer than expected.
- The user needs the results earlier and needs more resources than initially planned.

Renegotiation of existing SLAs is studied in a number of projects [19–21] and in the GRAAP (Grid Resource Allocation Agreement Protocol) working group of the Open Grid Forum [22]. Besides the open questions of the renegotiation itself, such as how to create a completely new SLA or an extension to the existing one, or how to constrain the SLA to speed up the negotiation process by reducing the complexity, there are a number of issues with the local resource management systems; however currently only very few of these support dynamic changes to the resource allocation for an application or a user. If there are not enough resources available, other jobs must be migrated or terminated.

12.7 MARKET MODELS FOR RESOURCE ALLOCATION SUPPORT

The practical application of some of the concepts outlined above requires negotiation and planning, which may be supported through the use of economic models and algorithms. In the discussion to follow, negotiation mechanisms are based primarily on price with reference to particular QoS attributes (or SLOs). In reality, a multicriteria approach is often likely to be adopted. Such approaches (whether single or multicriteria) would fit into the "SLA negotiation and supervision" services (and utilize the general and supporting services) shown in Figure 12.1. One scenario to consider is a Grid market in which resource providers and consumers compete in an electronic market, which is composed of several independent participants who act selfishly and follow their own concrete objectives. Taking this into account, it is useful to provide sophisticated mechanisms that allow market participants customize their behavior in base to economic policies that help them to achieve their goals.

For a resource provider, the most common objective is the *midterm* revenue maximization. This section describes some of the most popular economic policies used to achieve this goal: dynamic pricing, SLA negotiation strategies, penalties and rewards, and selective SLA fulfillment.

12.7.1 Dynamic Pricing

Economic markets are composed of a set of tradable goods (*supply*) offered by the market, and a set of buyers (*demand*) who ask for a subset of the supplied goods. In a Grid market the *supply* consists of computer resources, such as storage systems, or clusters, and the *demand* are the users and applications that are willing to pay for accessing these resources. The goal of dynamic pricing is to find an efficient allocation of the supply among the demand.

The *law of demand* states that the higher the price of a good, the lesser the demand for this good; the *law of supply* states that the higher the price of a good, the higher the quantity that will be supplied, because selling higher amounts of goods at higher prices will increase the revenue. Thus, both supply and demand can be controlled by changing the prices of goods; at lower prices, supply is low and demand is high, and at higher prices, supply is high and demand is low. This leads to the market being in disequilibrium.

In a market in disequilibrium, if the prices are too low, the demand cannot be satisfied because the supply is too scarce, and if the prices are too high, the supply cannot be sold completely because demand is too low. The efficient allocation that dynamic pricing pursues is achieved when it reaches the *equilibrium* point, and both demand and supply are satisfied. Two approaches to finding the optimal pricing within such a market will be explained: tâtonnement and auctioning [23].

12.7.1.1 Tâtonnement. *Tâtonnement* (literally meaning "groping about" in French) is an iterative method used to progressively adapt the prices to the current market status. Ferguson et al. [24] propose the following algorithm for this process:

1. Choose an initial price vector $\rightarrow\vec{p} = \{p_1, p_2, \ldots, p_n\}$, where $p_1, p_2, \ldots, p_n$ are the prices for the resources numbered between 1 and n, respectively, and a minimum price vector $\vec{m} = \{m_1, m_2, \ldots, m_n\}$, which is the price below which the provider will not sell the resource.
2. For each resource, find the *excess demand function*$Z_i(\vec{p})$, which is the demand for a resource i at a given price p_i minus the supply of it.
3. If for each resource i, $Z_i(\vec{p}) = 0$ or $Z_i(\vec{p}) \leq 0$ and $p_i = m_i$, equilibrium has been reached, then the iteration stops.
4. Otherwise, for all the resources update the price vector $\vec{p}$ following the next formula:

$$p_i = \max\left[p_i + p_i \frac{Z_i(\vec{p}\,)}{S_i}, m_i\right]$$

   where S_i is the supply for resource *i*.
5. Go to step 2.

The formula in step 4 can be replaced by another that the provider considers better for its requirements. Using this approach (step 4), the market evolves to equilibrium quickly in the first few iterations, with an increase in accuracy in the next ones. If the number of resources is high, the prices will be always low and the market will not be profitable. In this case, the provider should consider applying a *quantity tâtonnement* process [25], where the supply is adapted to the demand by altering the prices.

12.7.1.2 Auctions. In *auctions*, the price can be initially established by the seller, but the final price is determined by the customer who wins the auction based on specific rules (determined by the type of auction being considered):

- *English auction*—seller gives a start price, and buyers who are interested in acquiring the resource increment the price in their bids. The highest bidder obtains the resource.
- *Dutch auction*—seller gives the highest price, and it is gradually lowered by the seller until one of the buyers claims the resource.
- *Hybrid auction*—asking price of a resource is increased if a bid is submitted, and decreased if no bid is submitted.
- *Sealed-bid auction*—each customer submits a sealed bid, without knowing the bids from other customers and vice versa. When all the bids are received, the seller gives access to the resource to the highest bidder.

12.7.2 SLA Negotiation Strategies

Negotiation is the process toward creating suitable agreements between different parties in a Grid. The whole task of negotiation is challenging, as the resources are heterogeneous and the service provisioning is not a standardized good (as, e.g., in stockmarkets) but depends on the individual requirements and preferences of the user for a particular task. During the negotiation process, the conflicts between the different objectives and policies of the negotiating parties must be reconciled.

Strategic negotiation models are introduced in the paper by Li and Yahyapour [26], where bilateral negotiation models consist of (1) the negotiation protocol, (2) utility functions or preference relationships for the negotiating parties, and (3) the negotiation strategy that is applied during the negotiation process. In the negotiation model described, the negotiation parties do not know the opponents' private reservation information and their preferences/utility functions. The utility function of the customer and the preference relationship of the seller are given for their respective automated negotiation agents. Typically, objectives of customers are related to specifications such as response time, CPU (number, speed, architecture), and amount of memory; such specifications may also include business requirements such as "best price" or "maximize efficiency." The main objective of resource/service providers usually is to maximize the economic profit. All the different objectives are independent and should be dealt with simultaneously, taking into account the multiple criteria [29].

Li and Yahyapour [26] propose to divide the different criteria in subutility functions and express the user preferences by assigning different weights to them. For example

$$U_{\text{price}} = \frac{P_c^{\max} - P_c^t}{P_c^{\max} - P_c^{\min}}, \qquad U_{\text{time}} = \frac{T_c^{\max} - T_c^t}{T_c^{\max} - T_c^{\min}}$$

where U_{price} is the subutility function for the price of a Grid job and U_{time} is the subutility function for the job's waiting time; $P_c^{\max}$ is the maximum acceptable price for the user, and $P_c^{\min}$ is the minimum offered price from the user. The user weighs his/her preferences with $\mathrm{W}_{\text{price}}$ and W_{time} to obtain the following aggregate utility function:

$$U_{\text{job}} = W_{\text{price}} U_{\text{price}} + W_{\text{time}} U_{\text{time}}, \qquad W_{\text{price}} + W_{\text{time}} = 1$$

In the negotiation process, an agent may change its preference by changing the weight associated to that issue.

12.7.3 Penalties and Rewards

The *reward* can be defined as the amount of money that a client must pay to a Grid provider when it finishes a task or service correctly. If the provider does not fulfill one of the SLA terms, the *penalty amount* must be paid to the client. Both reward and penalty must be agreed at negotiation time and must be defined specified in the SLA.

Penalty can be specified in SLA as a *penalty function* [27], which not only penalizes the provider for service failures but also compensates the users for them. An example of a reward and the penalty function is as follows:

$$\mathtt{reward} = \mathtt{price} - (\mathtt{delay} \times \mathtt{penalty\_rate}),$$

$$\texttt{where } \mathtt{delay} = \begin{cases} 0, \ \mathtt{if\ finish\_time} \le \mathtt{deadline} \\ (\mathtt{finish\_time} - \mathtt{submit\_time}) - \mathtt{deadline} \ \texttt{otherwise} \end{cases}$$

Figure 12.3 illustrates the reward/penalty function; the provider earns the agreed- to price if it completes the job before the established deadline. The value decreases linearly with the delay until the reward is negative, which represents a penalty. The penalty could be bounded to avoid excessive penalties.

Reward/penalty functions can be improved with additional parameters; for instance, they may contain not only the finish time but also additional parameters such as disk speed or number of CPUs. Also, the SLA can specify in more precise terms what constitutes an SLA violation—to violate a single SLA term, a particular number of terms, violate the SLA terms during a time interval, or over repeated intervals.

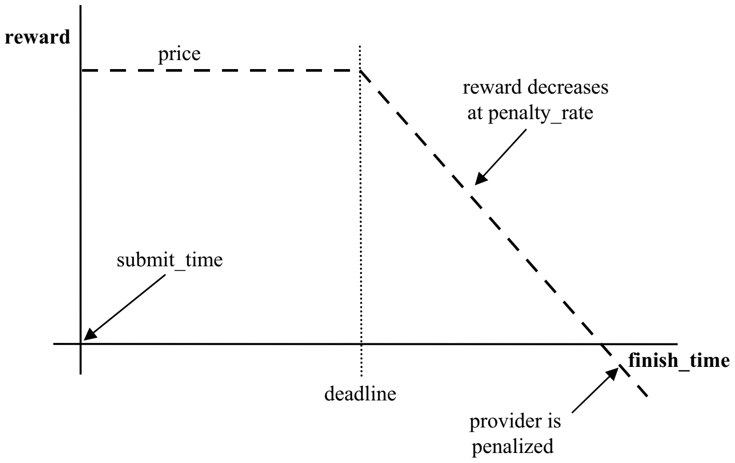

Figure 12.3 Example of reward/penalty defining function.

The reward/penalty function can be used by the resource provider to perform more accurate scheduling for its incoming jobs, by considering reward, penalty rates, deadlines, and managing the risk of paying any penalties. Macias et al. [28] performed some experiments that demonstrated the economic benefit of taking into account penalties and rewards in system overloading scenarios: if not all the SLAs can be agreed to, it is possible to combine selective SLA violations and task reallocations to minimize the economic loss. Figure 12.4 shows that using selective SLA fulfillment will allow providers have up to 60% of higher economic revenue when the resource pool is heavily overloaded: in a farm of 75 resources, if part of the resource pool

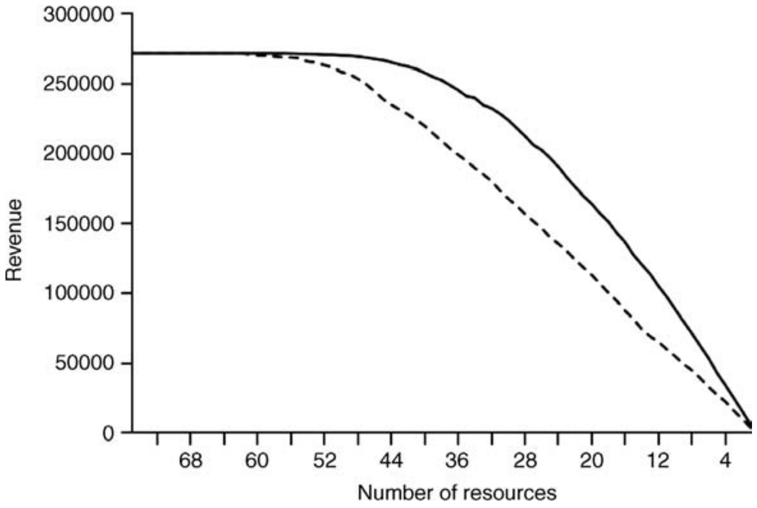

Figure 12.4 Comparison of revenue between policies usage (continuous line) and no policy (dashed line).

crashes, a portion of the previously accepted SLAs will be violated. Figure 12.4 also shows how the maximum revenue is maintained with fewer resources if economic policies are applied.

12.8 CASE STUDY

Various applications could benefit from the use of SLA-based electronic market mechanisms within a Grid computing infrastructure. Let us consider one application scenario that requires a highly specialized service, such as, a specific data-mining service, with several suppliers offering this service. An application example is the *c*atallactic *co*llaborative *vi*rtual *te*ams (Cat-COVITE) [30,31] prototype application consisting of three main elements: (1) one or more user services; (2) a "master Grid service" (MGS), responsible for interacting with a Grid market manager (GMM) to find an EPR of a service instance; and (3) one or more service instances that are being hosted on a particular resource. The Cat-COVITE application involves searching through distributed product catalogs—modeled as a Web services–enabled database, and using a distributed search strategy. The particular approach adopted in the Cat-COVITE application is also employable in other industrial applications that make use of distributed databases.

The interaction between the application prototype and the GMM is based on the WS-Agreement. When a client issues a request, the application determines which services are required to fulfill it. These services represent either software executables (e.g. a mathematical algorithm) or computational resources. The application service translates these requirements to a WS-Agreement template, which is submitted to the GMM. The middleware searches among the available service providers, which have registered their particular service specifications, such as contractual conditions, policies, and QoS levels. When a suitable service provider is found, the application requirements are negotiated by agents implemented in the middleware who act on behalf of the service providers as sellers, and the applications as buyers. Once an agreement is reached between trading agents, a service instance is created for the application, and a reference is returned to the application (which can subsequently be used to invoke it).

In the COVITE prototype, suppliers, and purchasers collaborate to procure supplies for a particular construction project by using the COVITE application. These projects are usually unique, very complex, and involve many participants from a number of organizations. These participants need to work concurrently, thus requiring real-time collaboration between geographically remote participants. Each consortium is in effect a virtual organization (VO). The application requires the search to take place across a large number of supplier databases to retrieve products matching a criteria set by the purchasers or contractors. The application enables a search to be performed using a cluster of machines in a Grid network.

The COVITE prototype application is divided into two functional services: (1) security service and (2) multiple database search services (DSSs). Each DSS enables searching across a large number of supplier databases (SDs) using a MGS

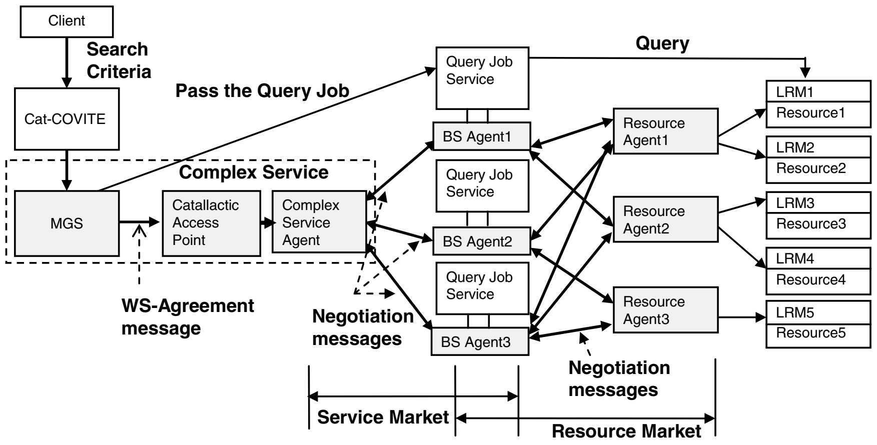

Figure 12.5 Cat-COVITE prototype and catallactic agents.

instance via a cluster. In this instance, the query is defined according to a data model that is specific to a given application domain. Arbitrary text queries (e.g., as in the Google search engine) are not allowed. The COVITE application allows VOs to plan, schedule, coordinate, and share components between designs, and from different suppliers. The ability of a free-market economy to adjudicate and satisfy the needs of VOs, in terms of services and resources, represents an important feature of the catallaxy mechanism. There are also possibilities for each of these VOs to try maximizing their own utility on the market.

12.8.1 Cat-COVITE Prototype and the Catallactic Grid Markets

Figure 12.5 shows the Cat-COVITE components and related catallactic agents as buyers and seller in a service and a resource market. The complex service, which is an abstract component consisting of the MGS, the catallactic access point (CAP)—the access point between the application and the market, and the complex service agent; the query job service—a type of basic service; and the resource where the query service is executed—as a type of computational resource. We do not consider any service composition mechanisms and concentrate on the allocation process for each basic service.

‘The complex service agent is the buyer entity in the service market, and the query job service, via the basic service agent, is the seller entity on the service market. complex service agent starts parallel negotiation with a number of agents representing query job services (basic services) and choses one of them on the basis of negotiation on price, while the complex service, via MGS, translates a request to a basic service, of query job service type. The complex service includes the following activities:

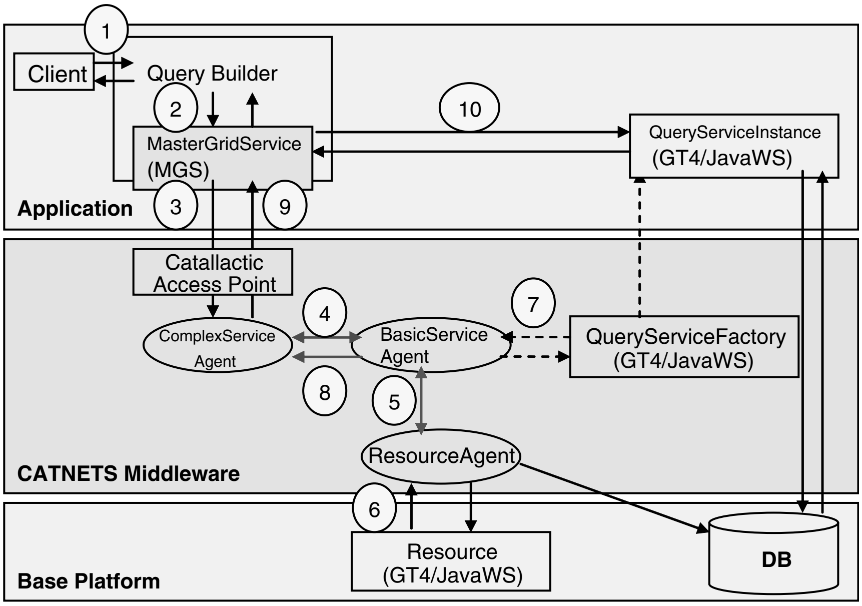

Figure 12.6 Integration of Cat-COVITE and catallactic middleware.

- Translates a request to a basic service—of query job service type
- Starts parallel negotiation with a number of agents representing query job services (basic services)
- Sends a query to a list of query job services (basic services)

The query job service is the buyer entity in the resource market, and the local resource managers (LRMs) are the seller entities in the resource market. The main function of a basic service agent within the resource market is coallocation of resources (resource bundles) by parallel negotiation with different resource providers (LRM entities). The query job service, which is a type of basic service, involves query execution on a particular database and consists of

- Query job execution environment (offers the deployment of "slaves," which are able to execute the query)
- Translation of query to resource requirements

Within the Cat-COVITE application, the query job service needs to support response time as the main QoS metric. With this goal the query job service buys resources in the resource market. Resource seller entities are able to provide a set of resources via the LRM. The resource agents act on behalf of these LRMs, which hide the physical resources behind them.

Figure 12.6 shows a detailed view of the architecture, identifying the placement of logical components along the three layers: the application layer, the catallactic middleware layer, and the base platform layer. At the application layer, the application must provide an interface to the middleware, which must issue the request for services to the middleware, and use the references to service instances provided by the middleware to execute such services. At the middleware layer, a set of agents provide the capabilities to negotiate for services and the resources needed to execute them. The complex service agent acting on behalf of the application initiates the negotiation. Basic service and resource agents manage the negotiation for services and resources, respectively. Also, a service factory is provided to instantiate the service on the execution environment selected during the negotiation process. Finally, at the base platform layer, a resource is created to manage the allocation of resources to the service. This resource represents the "state" of the service from the perspective of the middleware. (*Note:* This does not mean that the service is stateful from the perspective of the application.)

The flow of information among the logical components can be summarized as follows: A client issues a request to the application (1), which builds a query and requests the execution of query to the MGS (2). The MGS contacts a CAP asking for a WS-agreement template for such a service. The MGS fills in the template and sends back an agreement offer (3). Note that the generator of the WS-agreement ensures that there is an agreement on terms between the agreement initiator and the agreement offer provider. The creation of the agreement template in this way ensures that there is some consensus on the use of particular terms between the application and the CAP. The complex service agent initiates catallactic mechanisms to find the appropriate basic services and resources. The complex service agent uses discovery mechanisms implemented in the middleware to locate basic service agents providing a query service. When a number of basic service agents are discovered, it starts negotiations with one of them (4). In turn, such basic service agent must discover and negotiate with a resource agent for query execution resources in the resource market (5). Negotiations are implemented by the economic framework layer, where different protocols can be used depending on the agent's strategy. When an agreement with a basic service agent is reached, the resource agent instantiate a Resource to keep track of the allocated resources and provides to the basic service agent a handle for this resource (6). Subsequently, a basic service agents use the query service factory to instantiate the query service on the selected GT4 container (7). A basic service agent returns to the complex service agent the reference of the newly instantiated query service and the related resource(s) (8). The reference to the query service is returned to the MGS (9), which uses it to invoke the service, passing the query to be executed (10).

The example scenario in terms of the Cat-COVITE application in the prototype interacting with the middleware is as follows. An MGS needs to run a search request. The MGS sends an agreement offer (AO), based on the agreement template (AT) (see Listing 12.1) downloaded from the CAP, to find a query job service. The complex service agent, acting on behalf of the complex service (in this case represented by the MGS) negotiates with the basic service agent for query services to fulfil the job. The AT specifies the service description elements that are allowed by the factory that advertises it.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementTemplate AgreementID="QueryTemplate-v001"
xmlns:wsag="http://schemas.ggf.org/graap/2005/09/ws-
agreement">
<wsag:Name>QueryComplexService</wsag:Name>
<wsag:Context>
   <wsag:AgreementInitiator> <! - can be a URI or a security
identity of the initiator -> NameOfTheInitiator
</wsag:AgreementInitiator>
  <wsag:ExpirationTime>DateTime</wsag:ExpirationTime>
  <wsag:TemplateID>QueryTemplate-001</wsag:TemplateID>
  <wsag:TemplateName>QueryComplexService </wsag:Template
   Name>
</wsag:Context>
<wsag:Terms>
   <BasicServiceType>QueryBasicService </BasicServiceType>
   <NumberOfBasicServiceNodes> <!- between 1 to 10 ->
   </NumberOfBasicServiceNodes>
   <BasicServiceConstraints>
      <ResponseTimePerRequest>10
      <!- maximum milliseconds ->
      </ResponseTimePerRequest>
   </BasicServiceConstraints>
   <Price> </Price>
</wsag:Terms>
</wsag:AgreementTemplate>
```

Listing 12.1 Agreement template.

The agreement offer is initiated by the agreement initiator, in this case the MGS and presented in Listing 12.2. If the agreement provider does not accept the offer, the agreement initiator has to send another agreement offer. The negotiation between the agreement provider and initiator, in this scenario, is based on price.

12.9 CONCLUSIONS

An overview of current efforts making use of service-level agreements for resource management and allocation has been provided. The emergence of service-oriented Architectures provide new approaches for increasing the pool of available resources beyond local ones for executing distributed applications. Adding resources provided by collaborating institutions or from a commercial service provider to satisfy resource demand exceeding the locally available capacity is close to becoming a common operation rather than a proof of concept deployed in a limited context within a project environment. However, distributed environments crossing administrative boundaries have no centralized coordinators, and need new mechanisms to ensure a level of reliability similar to that in the local environment. The use of SLAs to specify and

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementTemplate AgreementID="QueryTemplate-v001"
xmlns:wsag="http://schemas.ggf.org/graap/2005/09/ws-
agreement">
<wsag:Name>QueryComplexService</wsag:Name>
<wsag:Context>
  <wsag:AgreementInitiator> <! - can be a URI or a security
identity of the initiator -> NameOfTheInitiator </wsag:
AgreementInitiator>
  <wsag:ExpirationTime>DateTime</wsag:ExpirationTime>
  <wsag:TemplateID>QueryTemplate-001</wsag:TemplateID>
  <wsag:TemplateName>QueryComplexService </wsag:Template
  Name>
</wsag:Context>
<wsag:Terms>
  <BasicServiceType>QueryBasicService </BasicServiceType>
  <NumberOfBasicServiceNodes>1 <!- between 1 to 10 ->
  </NumberOfBasicServiceNodes>
  <BasicServiceConstraints>
    <ResponseTimePerRequest>10
    <!- maximum milliseconds ->
    </ResponseTimePerRequest>
  </BasicServiceConstraints>
  <Price>100</Price>
</wsag:Terms>
</wsag:AgreementTemplate>
```

Listing 12.2 Agreement offer.

provision access to resources on the basis of QoS properties, even where the service level cannot be enforced because of to the autonomy of the different domains, has become an important new research objective recently.

With the proposed OGF recommendation WS-Agreement the technology is now available to define and implement SLAs in an interoperable way. Over the last year a number of resource management systems and Grid schedulers have been adopting SLAs as a means for reliable allocation. Applying SLAs for resource management and allocation allows one to consider computational and data resources as tradable goods. This broadens the traditional approach of resource management towards implementation of economic models. As shown in section 12.7, this facilitates the establishment of sophisticated methods for resource selection, dynamic pricing, and allocation. Finally, we presented a case study highlighting the use of SLAs in an auction-based economic model for allocating jobs to resources in an environment for the architecture/engineering/construction industry (COVITE). The next step involves making the management of SLAs more dynamic. To that end, the possibility of negotiating agreements needs to be extended. Providing mechanisms to securely renegotiate existing agreements will provide additional flexibility in supporting highly dynamic service provisioning environments in the future.

REFERENCES

1. V. Yarmolenko and R. Sakellariou, An evaluation of heuristics for SLA based parallel job scheduling, *Proc. 3rd High Performance Grid Computing Workshop* (in conjunction with IPDPS 2006), Rhodes, Greece, 2006.
2. *Job Submission Description Language* (*JSDL*) *Specification v1.0*, Grid Forum Document, GFD.56, Open Grid Forum, 2005.
3. P. Masche, B. Mitchell, and P. Mckee, The increasing role of SLAs in B2B, *Proc. 2nd International Conf. Web Information Systems and Technologies,* Setubal, Portugal, April 2006.
4. M. Surridge, S. Taylor, D. de Roure, and E. Zaluska, Experiences with GRIA: Industrial applications on a Web services Grid, *Proc. 1st IEEE International Conf. e-Science and Grid Computing,* Melbourne, Australia, 2005.
5. *gLite—Lightweight Middleware for Grid Computing*, http://glite.web.cern.ch/glite/, last accessed June 2008).
6. I. Foster, Globus toolkit version 4: Software for service-oriented systems, *Proc. IFIP International Conf. Network and Parallel Computing,* LNCS, Vol. 3779, Springer, 2006.
7. M. Riedel et al., Web services interfaces and open standards integration into the European UNICORE 6 Grid middleware, *Proc. 2007 Middleware for Web Services* (*MWS 2007*) *Workshop at 11th International IEEE EDOC Conf. "The Enterprise Computing Conference,"* Annapolis, MD, 2007.
8. H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler, Reliable orchestration of resources using WS-Agreement, *Proc. 2006 International Conf. High Performance Computing and Communications* (*HPCC06*)*,* Munich, Germany, Sept. 2006.
9. T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. L. Truong, A. Villazón, and M. Wieczorek, ASKALON: A Grid application development and computing environment, *Proc. 6th IEEE IEEE International Workshop on Grid Computing* (*Grid 2005*)*,* Seattle, WA, 2005.
10. *Web Services Agreement Specification* (*WS-Agreement*), Grid Forum Document, GFD.107, Open Grid Forum, 2007.
11. A. Keller and H. Ludwig, The WSLA framework: Specifying and monitoring service level agreements for Web services, *Journal of Network and Systems Management* (special issue on e-business management) **11**(1):57–81 (March 2003).
12. J. Seidel, O. Wäldrich, P. Wieder, R. Yahyapour, and W. Ziegler, Using SLA for resource management and scheduling—a survey, *Proc. Workshop on Using Service Level Agreements in Grids* (in conjunction with Grid 2007), Austin, TX, Sept. 2007.
13. OGSA Resource Selection Services WG (OGSA-RSS-WG), http://www.ogf.org/gf/group_info/view.php?group=ogsa-rss-wg (last accessed June 2008).
14. R. Gruber, V. Keller, P. Manneback, M. Thiémard, O. Wäldrich, P. Wieder, and W. Ziegler, Integration of Grid cost model into ISS/VIOLA meta-scheduler environment, *Proc. UNICORE Summit 2006* (in conjunction with Euro-Par 2006), Dresden, Germany, 2006.
15. T. Eickermann, W. Frings, O. Wäldrich, P. Wieder, and W. Ziegler, Co-allocation of MPI jobs with the VIOLA Grid metascheduling framework, *Proc. German eScience Conf. GES2007,* Baden-Baden, Germany, 2007.

16. P. Wieder, O. Wäldrich, R. Yahyapour, and W. Ziegler, Improving workflow execution through SLA-based advance reservation, In *Achievements in European Research on Grid System*, CoreGRID Integration Workshop 2006 (selected papers), October 19–20, Krakow, Poland. S. Gorlatch, M. Bubak, T. Priol (eds.), Springer Science + Business Media, New York, 2008, pp. 207–222.
17. C. Barz, T. Eickermann, M. Pilz, O. Wäldrich, L. Westphal, and W. Ziegler, Co-allocating compute and network resources—bandwidth on demand in the VIOLA testbed, *Proc. CoreGRID Symp.,* Rennes, France, Aug. 2007.
18. P. Wieder, O. Wäldrich, and W. Ziegler, A meta-scheduling service for co-allocating arbitrary types of resources, *Proc. 6th International Conf. Parallel Processing and Applied Mathematics* (*PPAM 2005*)*,* Poznan, Poland, 2005.
19. NextGRID: Architecture for next generation Grids, http://www.nextgrid.org/ (last accessed June 2008).
20. CoreGRID—European Research Network on Foundations, software infrastructures and applications for large-scale distributed, GRID and peer-to-peer technologies, http://www.coregrid.net/ (last accessed July 2009).
21. BREIN—Business objective driven Reliable and Intelligent Grids for Real BussiNess, http://www.eu-brein.com/ (last accessed June 2008).
22. Grid Resource Allocation Agreement WG (GRAAP-WG), http://www.ogf.org/gf/group_info/view.php?group=graap-wg (last accessed July 2009).
23. D. F. Ferguson, *The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms*, PhD thesis, Columbia Univ., New York, 1989.
24. D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, Economic models for allocating resources in computer systems, in *Market-Based Control: A Paradigm for Distributed Resource Allocation*, S. Clearwater, ed., World Scientific, Hong Kong, 1996.
25. A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory*, Oxford Univ. Press, UK, 1995.
26. J. Li and R. Yahyapour, Learning-based negotiation strategies for Grid Scheduling, *Proc. 6th IEEE International Symp. Cluster Computing and the Grid,* Singapore, May 16–19, 2006.
27. C. S. Yeo and R. Buyya, Service level agreement based allocation of cluster resources: Handling penalty to enhance utility, *Proc. 7th International Conf. Cluster Computing,* Boston, MA, Sept. 26–30, 2005.
28. M. Macias, O. Rana, G. Smith, J. Guitart, and J. Torres, Maximising revenue in Grid markets using an economically enhanced resource manager, *Concurrency and Computation: Practice and Experience*, DOI 10.1002/cpe.1370.
29. B. Schnizler, D. Neumann, D. Veit, and C. Weinhardt, Trading Grid services—a multi-attribute combinatorial approach, *European Journal of Operational Research* **187**(3): 943–961 (2006).
30. L. Joita, J. S. Pahwa, P. Burnap, A. Gray, O. Rana, and J. Miles, Supporting collaborative virtual organisations in the construction industry via the Grid, *Proc. UK e-Science All Hands Meeting 2004,* Nottingham, UK, Aug. 31–Sept. 3, 2004.
31. L. Joita, O. Rana, P. Chacin, I. Chao, F. Freitag, L. Navarro, and O. Ardaiz, Application deployment using catallactic Grid middleware, *Proc. 3rd International Workshop on Middleware for Grid Computing* (*MGC 2005*)*,* co-located with ACM/USENIX/IFIP Middleware 2005, Grenoble, France, Nov. 28–Dec. 2, 2005.