Architectural Models for Resource Management in the Grid

Rajkumar Buyya[†], Steve Chapin*, and David DiNucci [§]

School of Computer Science and Software Engineering †
Monash University,
Melbourne, Australia
rajkumar@csse.monash.edu.au

Dept. of Electrical Engineering and Computer Science *
Syracuse University,
Syracuse, NY, USA.
chapin@ecs.syr.edu

Elepar [§]
14380 N W Hunters Dr.
Beaverton, Oregon, USA
dave@elepar.com

Abstract: The concept of coupling geographically distributed (high-end) resources for solving large-scale problems is becoming increasingly popular, forming what is popularly called grid computing. The management of resources in the grid environment becomes complex as they are (geographically) distributed, heterogeneous in nature, owned by different individuals/organizations each having their own resource management policies and different access-and-cost models. In this scenario, a number of alternatives exist while creating a framework for grid resource management. In this paper, we discuss the three alternative models—hierarchical, abstract owner, and market—for grid resource management architectures. The hierarchical model exhibits the approach followed in (many) contemporary grid systems. The abstract owner model follows an order and delivery approach in job submission and result gathering. The (computational) market model captures the essentials of both hierarchical and abstract owner models and proposes the use of computational economy in the development of grid resource management systems.

1. Introduction

The growing popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we do computing and use computers today. The interest in coupling geographically distributed (computational) resources is also growing for solving large-scale problems, leading to what is popularly known as grid computing. In this environment, a wide variety of computational resources (such as supercomputers, clusters, and SMPs including low-end systems such as PCs/workstations), visualisation devices, storage systems and databases, special class of scientific instruments (such as radio telescopes), computational kernels, and so on are logically coupled together and presented as a single integrated resource to the user (see Figure 1). The user essentially interacts with a resource broker that hides the complexities of grid computing. The broker discovers resources that the user can access through grid information server(s), negotiates with (grid-enabled) resources or their agents using

middleware services, maps tasks to resources (scheduling), stages the application and data for processing (deployment) and finally gathers results. It is also responsible for monitoring application execution progress along with managing changes in the grid infrastructure and resource failures. There are a number of projects worldwide [5], which are actively exploring the development of various grid computing system components, services, and applications. They include Globus [7], Legion [9], NetSolve [10], Ninf [15], AppLes [11], Nimrod/G [3], and JaWS [16]. In [2], all these grid systems have been discussed.

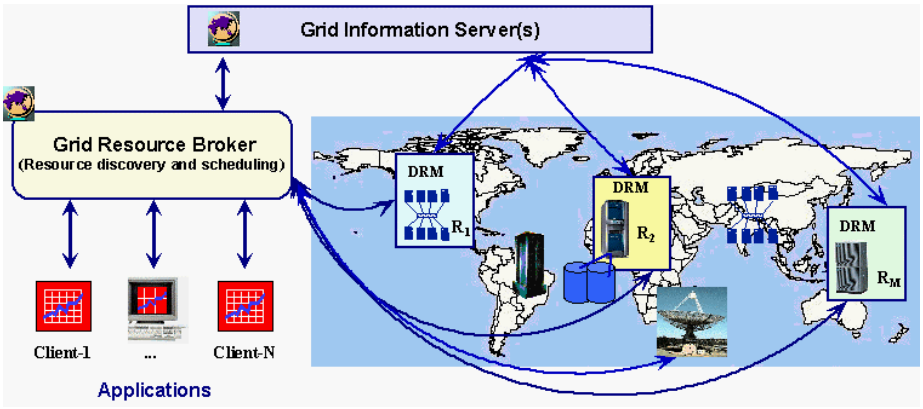

Figure 1: A Generic View of GRID System.

The current research and investment into computational grids is motivated by an assumption that coordinated access to diverse and geographically distributed resources is valuable. In this paradigm, it is not only important to determine mechanisms and policies that allows such coordinated access, but it also seems reasonable that owners of those resources, or of mechanisms to connect and utilize them should be able to recoup some of the resulting value from users or clients. Approaches to recouping such value in the existing Internet/web infrastructure, where e-commerce sites use advertising and/or mark-ups on products sold to show revenue, do not translate well (or are unsuitable) to a computational grid framework, primarily due to the fact that the immediate user of any specific resource in a computational grid is often not a human. Instead, in a grid, many different resources, potentially controlled by diverse organizations with diverse policies in widely-distributed locations, must all be used together, and the relationship between the value provided by each resource and the value of the product or service delivered to the eventual human consumer may be very complex. In addition, it is unrealistic to assume that human-created contracts can be developed between all potential resource users and resource owners in these situations, since the potential of computational grids can only be fully exploited if similar resources owned by different owners can be used almost interchangeably.

Still, the existing real world must be acknowledged. Grid resources are largely owned and used by individuals or institutions who often provide "free" access for solving problems of common interest/public good (e.g., SETI@Home [13]), prize/fame (e.g., distributed.net [14] response to challenge for breaking RSA security

algorithms), collaborative resources (GUSTO [6]), or by companies that are loathe to allow others to use them, primarily due to concerns about competition and security. The existing control over resources is subject to different policies and restrictions, as well as different software infrastructure used to schedule them. Any new approach to manage or share these resources will not be viable unless it allows a gradual layering of functionality or at least a gradual transition schedule from existing approaches to more novel ones. Even in the existing cases where money does not actually change hands, it is often important to provide a proper accounting of cross-organizational resource usage. In order to address these concerns, we propose different approaches for modeling grid resource management systems.

2. Architecture Models

As the grid logically couples multiple resources owned by different individuals or organisations, the choice of the right model for resource management architecture plays a major role in its eventual (commercial) success. There are a number of approaches that one can follow in developing grid resource management systems. In the next three sections, we discuss the following three different models for grid resource management architecture:

- Hierarchical Model
- Abstract Owner Model
- Computational Market/Economy Model

In the first, we characterize existing resource management and scheduling mechanisms by suggesting a more general view of those mechanisms. Next, we suggest a rather idealistic and extensive proposal for resource sharing and economy, which for the most part, ignores existing infrastructure in order to focus on long-term goals. Finally, we describe a more incremental architecture that is already underway to integrate some aspects of a computational economy into the existing grid infrastructure. Table 1 shows a few representative systems whose architecture complies with one of these models.

MODEL	REMARKS	SYSTEMS
Hierarchical	It captures architecture model followed in most contemporary systems.	Globus, Legion, Ninf, NetSolve.
Abstract Owner	It follows an order and delivery model for resource sharing, which for the most part, ignores existing infrastructure in order to focus on long-term goals.	Expected to emerge.
Economy/Market	It follows economic model in resource discovery and scheduling that can co-exist or work with contemporary systems and captures the essence of both hierarchical and abstract owner models.	Nimrod/G, JaWS, Myriposa, JavaMarket.

Table 1: Three Models for a Grid Resource Management Architecture.

The grid architecture models need to encourage resource owners to contribute their resources, offer a fair basis for sharing resources among users, and regulate resource demand and supply. They influence the way scheduling systems are built as they are responsible for mapping user requests to the right set of resources. The grid scheduling systems need to follow multilevel scheduling architecture as each resource has its own scheduling system and users schedule their applications on the grid using super-schedulers called resource brokers (see Figure 1).

3. Hierarchical Resource Management

The hierarchical model for grid resource management architecture (shown in Figure 2) is an outcome of the Grid Forum [20] second meeting proposed in [21]. The major components of this architecture are divided into passive and active components. The passive components are:

- *Resources* are things that can be used for a period of time, and may or may not be renewable. They have owners, who may charge others for using resources and they can be shared, or exclusive. Resources might be explicitly named, or be described parametrically. Examples of resources include disk space, network bandwidth, specialized device time, and CPU time.
- *Tasks* are consumers of resources, and include both traditional computational tasks and non-computational tasks such as file staging and communication.
- *Jobs* are hierarchical entities, and may have recursive structure; i.e., jobs can be composed of subjobs or tasks, and subjobs may themselves contain subjobs. The *leaves* of this structure are tasks. The simplest form of a job is one containing a single task.
- *Schedules* are mappings of tasks to resources over time. Note that we map tasks to resources, not jobs, because jobs are containers for tasks, and tasks are the actual resource consumers.

The active components are:

- *Schedulers* compute one or more schedules for input lists of jobs, subject to constraints that can be specified at runtime. The unit of scheduling is the job, meaning that schedulers attempt to map all the tasks in a job at once, and jobs, not tasks, are submitted to schedulers.
- *Information Services* act as databases for describing items of interest to the resource management systems, such as resources, jobs, schedulers, agents, etc. We do not require any particular access method or implementation; it could be LDAP, a commercial database, or something else entirely.
- *Domain Control Agents* can commit resources for use; as the name implies, the set of resources controlled by an agent is a control domain. This is what some people mean when they say local resource manager. We expect domain control agents to support reservations. Domain Control Agents are distinct from Schedulers, but control domains may contain internal Schedulers. A Domain Control Agent can provide state information, either

through publishing in an Information Service or via direct querying. Examples of domain control agents include the Maui Scheduler, Globus GRAM, and Legion Host Object.

- *Deployment Agents* implement schedules by negotiating with domain control agents to obtain resources and start tasks running.
- *Users* submit jobs to the Resource Management System for execution.
- *Admission Control Agents* determine whether the system can accommodate additional jobs, and reject or postpone jobs when the system is saturated.
- *Monitors* track the progress of jobs. Monitors obtain job status from the tasks comprising the job and from the Domain Control Agents where those tasks are running. Based on this status, the Monitor may perform outcalls to Job Control Agents and Schedulers to effect remapping of the job.
- *Job Control Agents* are responsible for shepherding a job through the system, and can act both as a proxy for the user and as a persistent control point for a job. It is the responsibility of the job control agent to coordinate between different components within the resource management system, e.g. to coordinate between monitors and schedulers.

We have striven to be as general as is feasible in our definitions. Many of these distinctions are logical distinctions. For example, we have divided the responsibilities of schedulers, deployment agents, and monitors, although it is entirely reasonable and expected that some scheduling systems may combine two or all three of these in a single program. Schedulers outside control domains cannot commit resources; these are known as metaschedulers or super schedulers. In our early discussions, we intentionally referred to control domains as "the box" because it connotes an important separation of "inside the box" vs. "outside the box." Actions outside the box are requests; actions inside the box may be commands. It may well be that the system is fractal in nature, and that entire grid scheduling systems may exist inside the box. Therefore, we can treat the control domain as a black box from the outside.

We have intentionally not defined any relationship between the number of users, jobs, and the major entities in the system (admission agents, schedulers, deployment agents, and monitors). Possibilities range from per-user or per-job agents to a single monolithic agent per system; each approach has strengths and weaknesses, and nothing in our definitions precludes or favors a particular use of the system. We expect to see local system defaults (e.g. a default scheduler or deployment agent) with users substituting their personal agents when they desire to do so.

One can notice that the word *queue* has not been mentioned in this model; queuing systems imply homogeneity of resources and a degree of control that simply will not be present in true grid systems. Queuing systems will most certainly exist within control domains.

Interaction of Components

The interactions between components of the resource management system are shown in Figure 2. An arrow in the figure means that communication is taking place

between components. We will next describe, at a high level, what we envision these interactions to be. This is the beginning of a protocol definition. Once the high-level operations are agreed upon, we can concern ourselves with wire-level protocols.

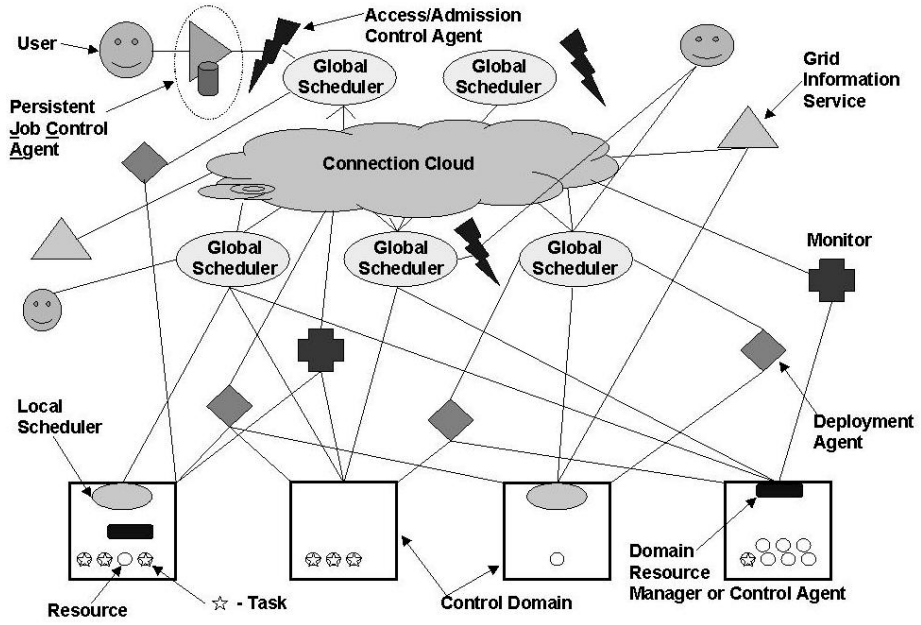

Figure 2: Hierarchical Model for Grid Resource Management.

We will begin with an example. A user submits a job to a job control agent, which calls an admission agent. The admission agent examines the resource demands of the job (perhaps consulting with a grid information system) and determines that it is safe to add the job to the current pool of work for the system. The admission agent passes the job to a scheduler, which performs resource discovery using the grid information system and then consults with domain control agents to determine the current state and availability of resources.

The scheduler then computes a set of mappings and passes these mappings to a deployment agent. The deployment agent negotiates with the domain control agents for the resources indicated in the schedule, and obtains reservations for the resources. These reservations are passed to the job control agent. At the proper time, the job control agent works with a different deployment agent, and the deployment agent coordinates with the appropriate domain control agents to start the tasks running. A monitor tracks progress of the job, and may later decide to reschedule if performance is lower than expected.

This is but one way in which these components might coordinate. Some systems will omit certain functionality (e.g. the job control agent), while others will combine multiple roles in a single agent. For example, a single process might naturally perform the roles of job control agent and monitor.

4. Abstract Owner (AO) Model

Where is the grid, and who owns it? These puzzles are not unique to the grid. When one makes a long distance phone call, who "owns" the resource being used? Who owns the generators that create the electricity to run an appliance? Who owns the Internet? Users of these resources don't care, and don't want to care. What they do want is the ability to make an agreement with some entity regarding the conditions under which the resources can be used, the mechanisms for using the resources, the cost of the resources, and the means of payment. The entity with which the user deals (the phone company, power company, or ISP) is almost certainly not the owner of the resources, but the user can think of them that way abstractly. They are actually brokers, who may in turn deal with the owners, or perhaps with more brokers. At each stage, the broker is an abstraction for all of the owners and so it is with the grid.

The grid user wants an abstraction of an entity that "owns" the grid, and to make an arrangement with that "owner" regarding the use of their resources, possibly involving a trade of something of value for the usage (which could be nothing more tangible than goodwill or the future use of their own resources). It is proposed here that each grid resource, ranging in complexity from individual processors and instruments to the grid itself, be represented by one or more "abstract owners" (abbreviated as *AOs*) that are strongly related to schedulers. For complex resources, an AO will certainly be a broker for the actual owners or other brokers, though the resource user doesn't need to be aware of this. (A resource user will hereafter be assumed to be a program, and referred to as a *client*. Human clients are assumed to use automated agents to represent him/her in negotiations with an AO.) The arrangement between the client and an AO for acquiring and using the resource can be made through a pre-existing contract (e.g. flat rate or sliding scale depending on time until resource available) or based on a dialogue between client and AO regarding the price and availability of the resource.

The remainder of this AO proposal describes what an AO looks like (externally and internally), what a resource looks like, how a client negotiates with an AO to acquire a resource, how a client interacts with a resource, and how AOs can be assembled into other constructs which may more closely resemble traditional schedulers. This work is still in the high-level design stages, in hopes that it will draw out refinements, corrections, and extensions that might help it to become viable.

General Structure of AO

At its most abstract, an AO outwardly resembles a fast-food restaurant (see Figure 3a). To acquire access to a resource from an AO that "owns" it, the prospective client (which may be another AO) negotiates with that AO through its Order Window. These negotiations may include asking how soon the resource may become available, how much it might cost, etc. If the prospective client is not happy with the results of the negotiations, it may just terminate negotiations, or might actually place an order. After being ordered, the resources are delivered from the AO to the client through the Pickup Window. The precise protocol to be used for acquiring the resources is flexible and may also be negotiated at order time--e.g. the client may be expected to pick up the resource at a given time, or the AO may alert the client (via an interrupt or

signal) when the resource is ready. Even if an order is placed (but the resource has not yet been delivered), the client may cancel the order through the order window.

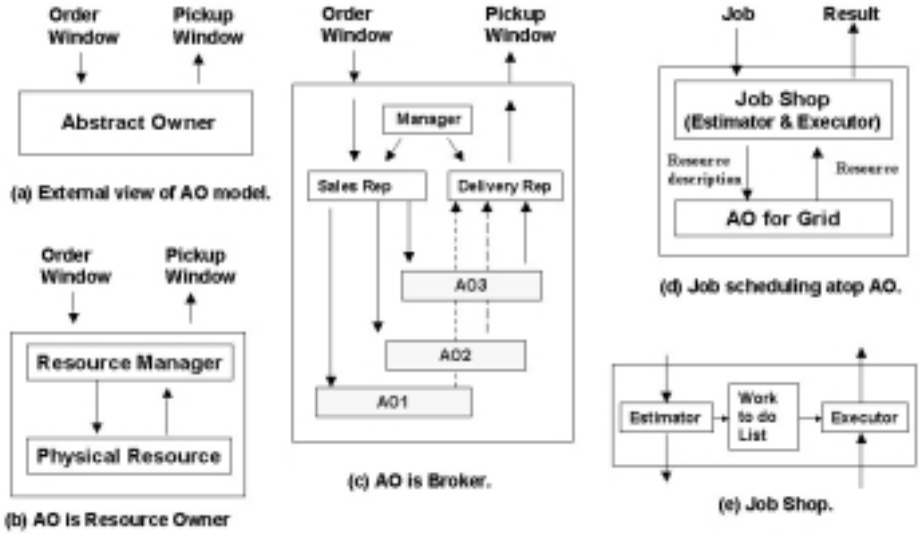

Figure 3: Abstract Owner Model for Grid Resource Management Architecture.

Little more is said here about the actual form of these “windows” except that they need to be accessible remotely, and must support a standard procedure-like interface in which values are passed to and returned from the window. Since interaction with an AO is likely to be rather infrequent and requires a relatively small amount of information flow, maximum efficiency is not necessarily required: CORBA or any number of other remote procedure invocation techniques can be used.

For the purposes of this discussion, a *resource* is roughly defined as any combination of hardware and software that helps the client to solve a problem, and a *task* is that part of a problem that is specified by the client after the resource has been delivered ("picked up") from the AO. Note that, unlike some other definitions of "task", these tasks may be very simple (e.g. a data set to be analyzed or a message to be sent), more general (e.g. a process to be executed), or very complex (e.g. a complete multi-process program and/or set of programs or processes to be executed in some order). While AOs do not specifically deal with entities called "jobs", techniques for applying the AO approach to traditional job scheduling will be addressed in the last subsection.

Resources can (and will) be regarded as objects, in the sense that they have an identity, a set of methods for initiating and controlling tasks, and attributes that serve to customize the resource. In general, the desired attributes will be determined during negotiation through the Order Window, when the client requests the resource, and will only be queried (not altered) after the resource is delivered. The methods may take many different forms, depending upon circumstances such as the type of resource, availability of hardware protections, and whether the method is to be invoked locally or remotely. For example, access to a local memory resource may have virtually no method protocol interfering with standard memory access

operations, while initiating a process on a distant processor may require more substantial method invocation protocol. A resource is relinquished by invoking its "relinquish" method (or by timing out).

The external structure of an AO was formulated to allow any level of nesting. Internally, an AO will differ in structure depending on whether it is a broker or an owner (or a combination). A pure owner of a single physical resource might be very simple (see Figure 3b), where the "manager" includes the intelligence required to negotiate, keep the schedule, and deliver the resource. For a higher-level broker, it might be more complex (see Figure 3c). Here, AO1, AO2, and AO3 represent other Abstract Owners, each with an Order Window used by the Sales Representative, and a Pickup Window used by the Delivery representative. Though these subordinate AOs are shown within a single parent AO, there is no reason that this relation must be hierarchical; a single AO may provide resources to a number of different parent AOs, which may assemble these into more complex resources in different ways or for different clients sets or may support different protocols or strategies or policies.

Grid Resources

Three primary classes are proposed here to represent resources: Instruments, Channels, and Complexes. An Instrument is a resource which logically exists at some location for some specific period of time, and which creates, consumes, or transforms data or information. The term "location" may be as specific or general as the situation merits. A Channel is a resource that exists to facilitate the explicit transfer of data or information between two or more instruments, either at different locations, or in the same location at different times (acting as sort of a temporary file in that case), or instruments which share space-time coordinates but have different protection domains. A Channel connects to an Instrument through a Port (on the instrument). A Complex is nothing more than a collection of (connected) Channel and Instrument resources.

Some important sub-classes of the Instrument class are the Compute instrument, the Archival instrument, and the Personal instrument. The Compute instrument corresponds to a processor or set of processors along with associated memory, temp files, software, etc. Archival Instruments (of which a permanent file is one sub-class) correspond to persistent storage of information. Personal instruments are those that are assumed to interface directly to a human being, ranging from a simple terminal to a more complex CAVE or speech recognition/synthesis device, and its specification may include the identity of the person involved. Of course, the Instrument class is also meant to accommodate other machines and instruments such as telescopes, electron microscopes, automatic milling machines, or any other sink or source for grid data.

As stated, an instrument exists in a location, and its methods may need to be called either locally (from the instrument itself) or remotely. For example, if a (reference to a) Compute instrument is acquired from an AO, the potentially distant process may want to invoke a "load_software" method to initiate a program on the resource. This new program may then want to invoke methods to access the temporary files or ports associated with the resource. Since the latter accesses will be local and must be

efficient, it is desirable to provide separate method invocation protocols for remote and local method invocation. Moreover, remote method invocations (RMIs) may themselves require the use of intermediate communication resources between the client and the resource, perhaps with associated quality of service (QoS) constraints.

To facilitate remote method invocations, any port(s) of an instrument can be specially designated as an RMI port. Such ports will have the appropriate RMI protocol handlers assigned to them. This designation is an attribute of the port--i.e., specified at resource negotiation time, through the "order window", just as authorization and notification style are. Methods can be invoked through such a port either by connecting a channel to the port and issuing the RMI request through the channel or in a connectionless mode by specifying the object and port. The former approach is best when issuing repeated RMI calls or when QoS is desired for RMI calls, the latter is best for one-time-only calls such as initializing an instrument which has just been acquired from an AO.

Negotiating with an AO

When negotiating through the order window, the client first effectively creates a "sample" resource object of the appropriate structure and assigns each attribute either (1) a constant value, (2) a "don't care" value, or (3) a variable name (which will actually take the form of, and be interpreted as, an index into a *variable value table*). If the same variable name is used in multiple places, it has the effect of constraining those attributes to have the same value. An example of this is to use a single variable to specify the "beginning time" attribute on several Instrument objects to cause them to be co-scheduled. Another is to specify variables for Instruments' object IDs, then to use those same variables when specifying the endpoints of the channels between them. The client may also specify simple constraints on the variables in a separate constraint list.

Usually, the values in the variable value table are filled and returned by the AO when the resource is acquired, but the client can designate some subset of those variables as *negotiating variables*. For these, the AO will propose values during negotiation, which the client can then examine to decide whether or not to accept the resource. (If accepted, these values essentially become constants.) In general, it is quicker for the client to specify additional constraints instead of using negotiation variables, allowing the decision on suitability to be made wholly within the AO, but negotiating variables can help when more complex constraints are required or when a client must decide between similar resources offered by different AOs.

In all, submissions to the Order Window from the client include the sample object attributes, the variable constraint list, a Negotiation Style, a Pickup Approach, an Authorization, a Bid, and a Negotiation ID. The Negotiation Style specifies whether the AO is to schedule the resource immediately (known as “Immediate”), or is to return a specified number of sets of proposed values for the negotiation variables (known as “Pending”), or is to finish scheduling based on an earlier-returned set of negotiation variable values (known as “Confirmation”), or is to cancel an earlier Pending negotiation (known as “Cancel”). The Pickup Approach specifies the protocol to be used between the AO and client at the Pickup Window—i.e. whether

the AO will alert the client with a signal, interrupt, or message when the resource becomes available, or the client will poll the Pickup Window for the resource, or the client can expect to find the resource ready at the window at a specified time. The Authorization is a capability or key which allows the AO to determine the authority of the client to access resources (and to bill the client accordingly when the resources are delivered). The Bid is a maximum price that the client is willing to pay for the resource, and may denote a pre-understood algorithm (or "contract") specifying how much the resource will cost under certain conditions. The Negotiation ID serves as a "cookie", and is passed back and forth between the client and AO to provide an identity and continuity for a multi-interaction negotiation, and continuity between the negotiation of a resource and the ultimate delivery of the resource through the Pickup Window. (A zero Negotiation ID designates the beginning of a new negotiation.)

If a Pending negotiation style is specified, the AO returns a value table containing sets of proposed values for the negotation variables, and an "Ask" price for each set. The intent of the Ask price is to inform the client of a sufficient Bid price to be used when requesting the resource, but the AO may conceivably accept even lower Bid prices depending upon the specific situation. For all negotiations, the AO returns a return code informing the client of the success of the operation, a Negotiation ID, (equal to that submitted, if it was nonzero), and an expiration date for the Negotiation ID. A single negotiation can continue until the Negotiation ID expires or a Negotiation Style other than "pending" is specified.

On a successful Immediate or Confirm request, the client can then submit the Negotiation ID to the Pickup Window, (at a time consistent with the Negotiation Style), to retrieve the resource. The Pickup Window returns the resource object, the variable value table, and a return code. Although the returned resource is logically an object, it is assumed that any attribute values that the client is concerned with are being returned in the Variable Value table, so the resource object just takes the form of a handle to access the resource object's methods.

Job Shops

AOs apparently perform only part of the standard job scheduling process—i.e. acquiring a resource—leaving the remainder to the client—i.e. assigning tasks to the resources and monitoring their completion and/or cleanup, often in sequential and dependent steps. But this is only partially true. Recall that a Compute Instrument, exclusive of the task that is eventually assigned to it by the client, may consist of both hardware and software components. While the software components often serve to create an environment in which the eventual task will execute (such as libraries or interpreters), they may also be compilers and/or complete user programs. That is, the Compute Instrument itself can be defined as a processor executing a specific program. The task assigned to such an instrument may be a data-set or source code to be read by that program (or compiler), or even nothing at all if the resource is completely self-contained. Since the AO is responsible for preparing the instrument for delivery through the Pickup Window and recovering it after it has been relinquished, it is indeed responsible for initiating this software and cleaning up after it.

The traditional sequential nature of job steps has resulted from the prevalence of

uniprocessors and traditional sequential thinking, but it is already common for parallel "make" utilities, for example, to exploit potential parallelism in job-like scripts. Similarly, in an AO resource, compute instruments running the individual "job steps" can be connected to communicate through channels, allowing them to be scheduled locally or in a distributed fashion, and scheduled sequentially or in parallel by the AO, subject to the dependences dictated by the channels and the QoS constraints assigned to those channels by the client. In this way, a job can be represented as a Complex Instrument in the AO infrastructure, where it will be scheduled.

Even with these capabilities, there is always the possibility that a more traditional job scheduler is required. In such a case, consider a new construct called a *job shop*, which uses AOs only to acquire resources, as shown in Figure 3d. See Figure 3e for an example of the internals of a standard job shop. The job shop primarily comprises "estimator" and "executor", much like an auto repair shop. The estimator deals with the customer to help determine how soon the job might be done and how much it might cost, requests the resources needed from the grid AO (through its order window), and records what needs to be done (in a job queue) when the resources are ready. The executor takes ready resources from the AO delivery window, dequeues the associated work from the job queue, builds any necessary environment for those tasks (e.g. telling message passing routines which channels to use), initiates tasks, collects answers, and notifies and returns the answer to the client.

Nesting job shops (or traditional job schedulers in general) is not as natural as nesting AOs, primarily because a job shop provides little feedback to the client until it has acquired resources *and* assigned tasks to them. This means that tasks are often assigned to some resources even before others have been allocated, and may be shipped around to where the resources are, long before they are needed there.

AO Summary

There are many remaining gaps in the above description, both in detail and in functionality. For example, little has been said about how any client, whether an end-user or another AO, will find AOs that own the desired kind of resources. Certainly, one approach is to imagine a tree of AOs (as in Figure 3c), with the client always interacting with the root AO, but it is unrealistic to consider this tree as being hardwired when residing in an environment as dynamic as a computational grid. More likely, existing Internet protocols can be adapted for this purpose, and an AO might have a third "business dealings" window to facilitate them. Before an approach like AO has any likelihood of acceptance in a large community, it must address many such challenges. Even a potentially useful and well-defined (successfully prototyped) AO protocol will not be viable unless it can coexist with other contemporary approaches. It is therefore important to understand how AOs and constructs in these other systems can build upon one another and mimic one another.

5. Economy/Market Model

The resources in the grid environment are geographically distributed and each of them is owned by a different organisation. Each of them has its own resource management

mechanisms and policies and may charge different prices for different users necessitating the need for the support of computational economy in resource management. In [17], we have presented a number of arguments for the need of an economy (market) driven resource management system for the grid. It offers resource owners better "incentive" for contributing their resources and help recover cost they incur while serving grid users or finance services that they offer to users and also make some profit. This return-on-investment mechanism also helps in enhancing/expanding computational services and upgrading resources. It is important to note that an economy[1] is one of the best institutions for regulating demand and supply. Naturally, in a computational market environment, resource users want to minimise their expenses (the price they pay) and owners want to maximise their return-on-investment. This necessitates a grid resource management system that provides appropriate tools and services to allow both resource users and owners to express their requirements. For instance, users should be allowed to specify their "QoS requirements" such as minimise the computational cost (amount) that they are willing to pay and yet meet the deadline by which they need results. Resource owners should be allowed to specify their charges—that can vary from time to time and users to users—and terms of use. Systems such as Mariposa [17], Nimrod/G [3], and JaWS [16], architect their user service model based on the economy of computations and it is likely that more and more systems are going to emerge based on this concept.

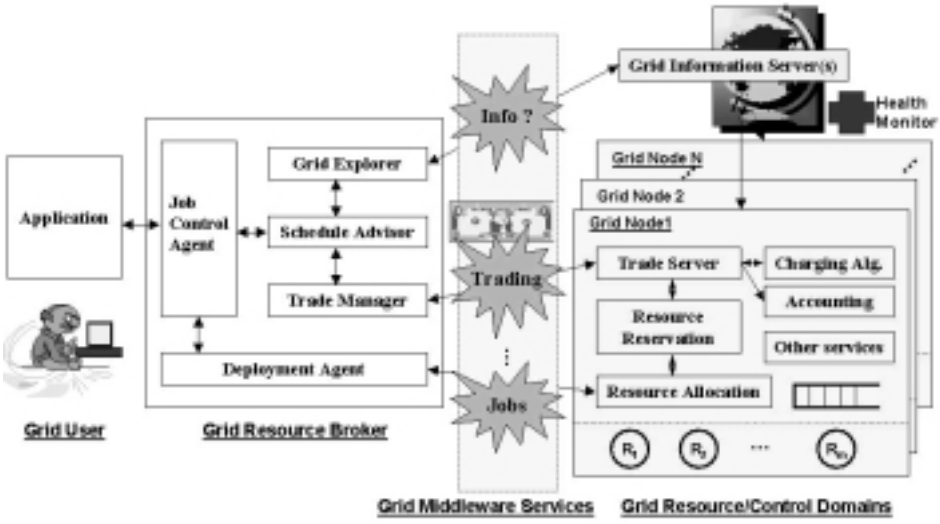

Figure 4: Economy Model for Grid Resource Management Architecture.

The economy/market model for grid resource management captures the essentials of both hierarchical and AO model presented above. Many of the contemporary grid systems fit to the hierarchical model and AO appears to be futuristic, but points out the need for economy in computation implicitly. The issues discussed in the hierarchical model apply to the market model, but it emphasizes the use of economic

[1] We use terms "economy" and "market" interchangeably.

based resource management and scheduling. One of the possible architectures for grid resource management based on computational market model is shown in Figure 4. Resource trading model can vary depending on the method/protocol used (by trade manager) in determining the resource access cost.

The following are the key components of economy-driven resource management system:

- User Applications (sequential, parametric, parallel, or collaborative applications)
- Grid Resource Broker (a.k.a., Super/Global/Meta Scheduler)
- Grid Middleware
- Domain Resource Manager (Local Scheduler or Queuing system)

Grid Resource Broker (GRB)

The resource broker acts as a mediator between the user and grid resources using middleware services. It is responsible for resource discovery, resource selection, binding of software (application), data, and hardware resources, initiating computations, adapting to the changes in grid resources and presenting the grid to the user as a single, unified resource. The components of resource broker are the following:

- **Job Control Agent (JCA):** This component is a persistent central component responsible for shepherding a job through the system. It takes care of schedule generation, the actual creation of jobs, maintenance of job status, interacting with clients/users, schedule advisor, and dispatcher.
- **Schedule Advisor (Scheduler):** This component is responsible for resource discovery (using grid explorer), resource selection, and job assignment (schedule generation). Its key function is to select those resources that meet user requirements such as meet the deadline and minimize the cost of computation while assigning jobs to resources.
- **Grid Explorer:** This is responsible for resource discovery by interacting with grid-information server and identifying the list of authorized machines, and keeping track of resource status information.
- **Trade Manager (TM):** This works under the direction of resource selection algorithm (schedule advisor) to identify resource access costs. It interacts with trade servers (using middleware services/protocols such as those presented in [4]) and negotiates for access to resources at low cost. It can find out access cost through grid information server if owners post it.
- **Deployment Agent:** It is responsible for activating task execution on the selected resource as per the scheduler's instruction. It periodically updates the status of task execution to JCA.

Grid Middleware

The grid middleware offers services that help in coupling a grid user and (remote) resources through a resource broker or grid enabled application. It offers core services [12] such as remote process management, co-allocation of resources, storage access, information (directory), security, authentication, and Quality of Service (QoS) such as

resource reservation for guaranteed availability and trading for minimising computational cost. Some of these services have already been discussed in the hierarchical model, here we point out components that are specifically responsible for helping out in offering computational economy services:

- **Trade Server (TS)**: It is a resource owner agent that negotiates with resource users and sells access to resources. It aims to maximize the resource utility and profit for its owner (earn as much money as possible). It consults pricing algorithms/models defined by the users during negotiation and directs the accounting system to record resource usage.
- **Pricing Algorithms/Methods**: These define the prices that resource owners would like to charge users. The resource owners may follow various policies to maximise profit and resource utilisation and the price they charge may vary from time to time and one user to another user and may also be driven by demand and supply like in the real market environment.
- **Accounting System:** It is responsible for recording resource usage and bills the user as per the usage agreement between resource broker (TM, user agent) and trade server (resource owner agent) [19].

Domain Resource Manager

Local resource managers are responsible for managing and scheduling computations across local resources such as workstations and clusters. They are even responsible for offering access to storage devices, databases, and special scientific instruments such as a radio telescope. Example local resource managers include, cluster operating systems such as MOSIX [18] and queuing systems such as Condor [12].

Comments

The services offered by trade server could also be accessed from or offered by grid information servers (like yellow pages/advertised services or posted prices). In this case a trade manager or broker can directly access information services to identify resource access cost and then contact resource agents for confirmation of access. The trade manager can use these advertised/posted prices (through information server) or ask/invite for competitive quotes (tenders) or bids (from trade server/resource owner agents) and choose resources that meet user requirements.

From the above discussion it is clear that there exist numerous methods for determining/knowing access cost. Therefore resource trading shown in Figure 4 is one of the possible alternatives for computational market model and it can vary depending on, particularly, trading protocols like in real world economy. Some of the real-world trading methods that can also be applied for computational economies include:

- Advertised/posted prices (classified advertisements) through information server
- Commodity exchanges
- Negotiated prices
- Call for (closed) tenders
- Call for (open) bids

Each of these methods can be applied in different situations for computational

economies and they create a competitive computational market depending on the demand and supply and the quality of service. The mechanism for informing resource owners about the availability of service opportunities can vary depending on its implementation. One of the simplest mechanisms is users (buyers) or/and resource owners (sellers or their agents renting/leasing computational services) make available or post/publicise their requirements in a known location (for instance, "exchange centre, share market, or grid information service directory"). Any one or all can initiate computational service trading. Through these mechanisms one can perform the following types of actions like in real world market economies:

- Users can post their intentions/offers to buy access to resources/services (e.g., "20 cluster nodes for 2 hours for $50);
- Resource owners/grid nodes/providers/agents can post offers to sell (e.g., systems like NetSolve can announce "we solve 1000 simultaneous linear equations for $5");
- Users/resource owners can query about current opportunities including prices/bids and historical information.

The different grid systems may follow different approaches in making this happen and it will be beneficial if they are all interoperable. The interoperability standards can be evolved through grid user/developer community forums or standardization organisations such as GF [20] and eGRID [22].

6. Discussion and Conclusions

In this paper we have discussed three different models for grid resource management architecture inspired by three different philosophies. The hierarchical model captures the approach followed in many contemporary grid systems. The abstract owner shows the potential of an order and delivery approach in job submission and result gathering. The (computational) market model captures the essentials of both hierarchical and abstract owner models and uses the concept of computational economy. We have attempted to present these models in abstract high-level form as much as possible and have skipped low-level details for developers to decide (as they mostly change from one system to another). Many of the existing, upcoming and future grid systems can easily be mapped to one or more of the models discussed here (see Table 1). It is also obvious that real grid systems (as they evolve) are most likely to combine many of these ideas into a hybridized model (that captures essentials of all models) in their architecture. For instance, our Grid Economy [4] is developed as a combination of Globus and GRACE services based on a (hybridized) market model.

The importance of market models for grid computing is also reported in the journal of *Scientific American* [23]: "So far not even the most ambitious metacomputing prototypes have tackled accounting: determining a fair price for idle processor cycles. It all depends on the risk, on the speed of the machine, on the cost of communication, on the importance of the problem--on a million variables, none of them well understood. If only for that reason, metacomputing will probably arrive with a whimper, not a bang". We hope that (our proposed) computational market model for grid systems architecture along with others will help the arrival of computational grids with a big bang (not a whimper)!

References

1. Ian Foster and Carl Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
2. Mark Baker, Rajkumar Buyya, Domenico Laforenza, The Grid: International Efforts in Global Computing, *Intl. Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Italy, 2000.
3. Rajkumar Buyya, David Abramson and Jon Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *4th Intl. Conf. on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, China.
4. Rajkumar Buyya, David Abramson and Jon Giddy, Economy Driven Resource Management Architecture for Computational Power Grids, *Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, USA.
5. Rajkumar Buyya, Grid Computing Info Centre: http://www.gridcomputing.com
6. Globus Testbeds - http://www.globus.org/testbeds/
7. Ian Foster and Carl Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11(2): 115-128, 1997.
8. Jack Dongarra, An Overview of Computational Grids and Survey of a Few Research Projects, *Symposium on Global Information Processing Technology*, Japan, 1999.
9. Steve Chapin, John Karpovich, Andrew Grimshaw, The Legion Resource Management System, *5th Workshop on Job Scheduling Strategies for Parallel Processing*, April 1999.
10. Henri Casanova and Jack Dongarra, *NetSolve:* A Network Server for Solving Computational Science Problems, *Intl. Journal of Supercomputing Applications and High Performance Computing*, Vol. 11, No. 3, 1997.
11. Fran Berman and Rich Wolski, The AppLeS Project: A Status Report, *8th NEC Research Symposium*, Berlin, Germany, May 1997. http://apples.ucsd.edu
12. Jim Basney and Miron Livny, Deploying a High Throughput Computing Cluster, *High Performance Cluster Computing*, Prentice Hall, 1999. http://www.cs.wisc.edu/condor/
13. SETI@Home – http://setiathome.ssl.berkeley.edu/
14. Distributed.Net – http://www.distributed.net/
15. Hidemoto Nakada, Mitsuhisa Sato, Satoshi Sekiguchi, Design and Implementations of Ninf: towards a Global Computing Infrastructure, *FGCS Journal*, October 1999.
16. Spyros Lalis and Alexandros Karipidis, JaWS: An Open Market-Based Framework for Distributed Computing over the Internet, *IEEE/ACM International Workshop on Grid Computing (GRID 2000)*, Dec. 2000. http://roadrunner.ics.forth.gr:8080/
17. Michael Stonebraker, Robert Devine, Marcel Kornacker, Witold Litwin, Avi Pfeffer, Adam Sah, Carl Staelin, An Economic Paradigm for Query Processing and Data Migration in Mariposa, *3rd International Conference on Parallel and Distributed Information Systems*, Sept. 1994. http://mariposa.cs.berkeley.edu:8000/mariposa/
18. Amnon Barak and Oren Laadan, The MOSIX Multicomputer Operating System for High Performance Cluster Computing, *FGCS Journal*, March 1998. www.mosix.cs.huji.ac.il
19. Bill Thigpen and Tom Hacker, Distributed Accounting on the Grid, *The Grid Forum Working Drafts*, 2000.
20. Grid Forum – http://www.gridforum.org
21. Steve Chapin, Mark Clement, and Quinn Snell, A Grid Resource Management Architecture, Strawman 1, *Grid Forum Scheduling Working Group*, November 1999.
22. European Grid Forum (eGRID) – http://www.egrid.org
23. W. Wayt Gibbs, Cyber View—World Wide Widgets, *Scientific American*, San Francisco, USA - http://www.sciam.com/0597issue/0597cyber.html