SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2002; **32**:135–164 (DOI: 10.1002/spe.432)

A taxonomy and survey of grid resource management systems for distributed computing

Klaus Krauter[1,*,†], Rajkumar Buyya[2] and Muthucumaru Maheswaran[1]

[1] *Advanced Networking Research Laboratory, Department of Computer Science, University of Manitoba, Winnipeg, Canada*
[2] *School of Computer Science and Software Engineering, Monash University, Melbourne, Australia*

SUMMARY

The resource management system is the central component of distributed network computing systems. There have been many projects focused on network computing that have designed and implemented resource management systems with a variety of architectures and services. In this paper, an abstract model and a comprehensive taxonomy for describing resource management architectures is developed. The taxonomy is used to identify approaches followed in the implementation of existing resource management systems for very large-scale network computing systems known as Grids. The taxonomy and the survey results are used to identify architectural approaches and issues that have not been fully explored in the research. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: distributed computing; grids; resource management; scheduling; taxonomies

INTRODUCTION

A distributed *network computing* (NC) system is a virtual computer formed by a networked set of heterogeneous machines that agree to share their local resources with each other. A Grid [1] is a very large scale, generalized distributed NC system that can scale to Internet-size environments with machines distributed across multiple organizations and administrative domains. The emergence of a variety of new applications demand that Grids support efficient data and resource management mechanisms. Designing a Grid architecture that will meet these requirements is challenging due to several issues [1,2]. Some of these issues are: (a) supporting adaptability, extensibility, and scalability, (b) allowing systems with different administrative policies to inter-operate while preserving site

[*] Correspondence to: Klaus Krauter, Redfern Broadband Networks, Level 1, 1 Central Avenue, Australian Technology Park, Eveleigh NSW1430, Australia.
[†] E-mail: kkrauter@rbni.com

Received 6 June 2001
Revised 12 September 2001
Accepted 17 September 2001

autonomy, (c) co-allocating resources, (d) supporting quality of service, and (e) meeting computational cost constraints.

For a Grid to efficiently support a variety of applications, the *resource management system* (RMS) that is central to its operation must address the above issues in addition to issues such as fault-tolerance and stability [3]. The RMS manages the pool of resources that are available to the Grid, i.e. the scheduling of processors, network bandwidth, and disk storage. In a Grid, the pool can include resources from different providers thus requiring the RMS to hold the trust of all resource providers. Maintaining the required level of trust should not hinder the efficiency of the RMS by increasing the overhead for basic operations. The resource providers may not participate in the Grid unconditionally, i.e. there may be different policies that govern how the resources should be used by the Grid such that the resources could still meet the local resource demands. It is the responsibility of the RMS to ensure that it handles the various resources while adhering to the different usage policies. Because of the different administrative policies, resource heterogeneity, and expected proportions of the Grid it may be necessary to employ a federation of RMSs instead of a single RMS. The RMSs in the federation should interoperate using an agreed set of protocols to manage the resources.

Applications may either directly or indirectly request resources from the Grid. Such resource requests are considered as jobs by the Grid. Depending on the application, the job may specify *quality of service* (QoS) requirements or accept best-effort service levels. The RMS is required to perform resource management decisions while maximizing the QoS metrics delivered to the clients when jobs have QoS constraints [4]. In practice, a Grid RMS may be required to handle different jobs using different policies. For example, some jobs may require QoS support while others may require best-effort processing. In general, requiring the RMS to support multiple policies can compel the scheduling mechanisms to solve a multi-criteria optimization problem.

The abstract model presented in this paper succinctly captures the essential components and functions of a Grid RMS. The significance of the different components and functions of the model are also discussed. A taxonomy is presented that classifies the approaches that form the 'design space' for the various components in the abstract model. The taxonomy is illustrated by applying it to categorize the approaches taken by a selected set of existing Grid RMSs. This application of the taxonomy also reveals some of the areas that are yet to be researched fully.

RELATED WORK

A distributed computing scheduling taxonomy is presented in Reference [5]. This taxonomy includes static scheduling techniques that are not addressed in our taxonomy. It also does not consider scheduling, state estimation, and resource models separately when classifying dynamic scheduling. The taxonomy for dynamic scheduling presented in Reference [6] only considers two aspects of resource management, scheduling and state estimation. Our taxonomy provides classification of resource models and examines scheduling and state estimation at a finer level. Several advances in distributed resource management since the publication of Reference [6] have also been incorporated into our taxonomy.

A taxonomy for heterogeneous computing environments is presented in Reference [7]. The taxonomy covers the application model, target platform model, and mapping heuristic model. Application models are not covered in our taxonomy and there is no differentiation on the target

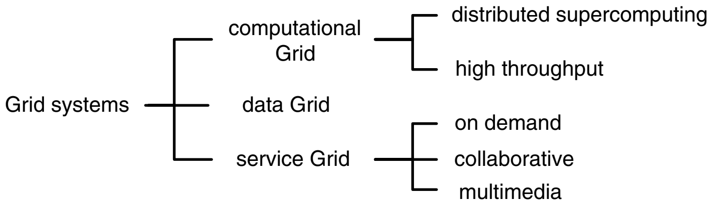

Figure 1. A Grid systems taxonomy.

platform model because the focus is on issues relevant to the designers of RMSs rather than issues relevant to application and toolkit designers.

Several taxonomies for characterizing a distributed system are presented in References [8] and [9]. The EM^3 taxonomy in Reference [8] classifies a heterogeneous computing system based on the number of execution modes and machine models. An extended version of the taxonomy developed in Reference [5] is also presented in Reference [8] to characterize the scheduling algorithms in heterogeneous computing systems. Our taxonomy is focused on RMS design issues and thus differs from the taxonomies presented in Reference [8]. The taxonomy presented in Reference [9] provides a broad characterization based on the external interfaces, internal system design, class of hardware and software resource support, and resource management issues. Our taxonomy of RMS is more detailed than the one presented in Reference [9].

GRID SYSTEM TAXONOMY

The design objectives and target applications for a Grid motivate the architecture of the RMS. This paper groups design objectives into three themes: (a) improving application performance, (b) data access, and (c) enhanced services. Using these themes, Grid systems are placed into the categories shown in Figure 1.

The *computational Grid* category denotes systems that have higher aggregate computational capacity available for single applications than the capacity of any constituent machine in the system. Depending on how this capacity is utilized, these systems can be further subdivided into *distributed supercomputing* and *high throughput* categories. A distributed supercomputing Grid executes the application in parallel on multiple machines to reduce the completion time of a job. Typically, applications that require distributed supercomputing are grand challenge problems such as weather modeling and nuclear simulation. A high throughput Grid increases the completion rate of a stream of jobs and are well suited for ‘parameter sweep’ type applications such as Monte Carlo simulations [10,11].

The *data Grid* category is for systems that provide an infrastructure for synthesizing new information from data repositories such as digital libraries or data warehouses that are distributed in a wide area network. Computational Grids also need to provide data services but the major difference between a data Grid and a computational Grid is the specialized infrastructure provided to applications for storage management and data access. In a computational Grid the applications implement their own storage management schemes rather than use Grid provided services. Typical applications for these systems include special purpose data mining that correlates information from multiple different data sources. The data Grid initiatives, European DataGrid Project [12] and Globus [13], are working on developing large-scale data organization, catalog, management, and access technologies.

The *service Grid* category is for systems that provide services that are not provided by any single machine. This category is further subdivided as *on-demand*, *collaborative*, and *multimedia Grid* systems. A collaborative Grid connects users and applications into collaborative workgroups. These systems enable real time interaction between humans and applications via a virtual workspace. An on-demand Grid category dynamically aggregates different resources to provide new services. A data visualization workbench that allows a scientist to dynamically increase the fidelity of a simulation by allocating more machines to a simulation would be an example of an on-demand Grid system. A multimedia Grid provides an infrastructure for real-time multimedia applications. This requires supporting QoS across multiple different machines whereas a multimedia application on a single dedicated machine may be deployed without QoS [14].

Most ongoing research activities developing Grid systems fall into one of the above categories. Development of truly general-purpose Grid systems that can support multiple or all of these categories remains a hard problem.

RMS DEFINITIONS AND REQUIREMENTS

This section develops an abstract model for resource management systems to outline the different architectural choices made in several existing and upcoming RMSs. To keep the abstract model compact, only the core functions of an RMS are included. Essential definitions and key resource management issues are presented before describing the proposed model.

In Grids, a *resource* is a reusable entity that is employed to fulfill a job or resource request. It could be a machine, network, or some service that is synthesized using a combination of machines, networks, and software. The *resource provider* is defined as an agent that controls the resource. For example, a resource broker that acts as the resource provider for a resource could provide the consumers with a ‘value added’ abstract resource [15]. Similarly, a *resource consumer* is defined as an agent that controls the consumer. A *resource management system* is defined as a service that is provided by a distributed NC system that manages a pool of named resources that is available to the NC such that a system- or job-centric performance metric is optimized.

Due to issues such as extensibility, adaptability, site autonomy, QoS, and co-allocation, resource management in Grid systems is more challenging than in traditional *distributed computing environments* (DCEs) [3]. In addition to the issues such as scalability, responsiveness, fault-tolerance, and stability that are encountered by the RMSs of traditional DCEs, the Grid RMSs have to deal with the issues that arise due to distributed ownership of the resources. Further, an issue may be addressed differently by DCE and Grid RMSs. For example, traditional DCE RMSs may be work conserving whereas Grid RMSs may not be work conserving. In Grids, due to resource heterogeneity and security concerns it may be best to execute a job only on a subset of resources. Another example would be the way QoS issue is handled. Traditional DCEs typically span a single administrative domain and are likely to handle jobs that originate from clients that belong to the organization. Therefore, QoS could be implemented in such systems, to a certain extent using job priorities.

However, in a Grid with jobs from different owners, providing QoS is more challenging. It is essential for the RMS to consider the jobs' access privileges, type of subscription, and resource requirements while determining the level of QoS. Depending on the type of QoS assurances given, a contract may be formed between the RMS and job. In this contract, the assurances given by the RMS may be contingent upon the job operating within some predefined resource limits. In order to forge a meaningful contract the expected resource usage of the job should be determined. However, accurately predicting the resource requirements of network applications remains a hard problem. When a job overruns its resource usage predictions, the RMS should ensure that it does not infringe on the resource allocations of other jobs, i.e. the Grid RMSs should provide 'isolation' among the jobs for fair resource management.

In general, a Grid application can have several components with the different components mapped onto different resources. In such situations, the RMS should use a global abstraction of the application to accredit and allocate resources [16]. The abstraction should be able to handle various application requirements such as co-allocation, QoS, and deadlines.

RMS ABSTRACT MODEL

Experience with large scale network computing systems has shown that efficient application and system performances are not necessarily the same [17]. More specifically, it may not be possible for the same scheduler to optimize application and system performances. One solution is to have a multi-layer RMS [18]. For example, to use an application scheduler such as Application Level Scheduling (AppLeS) [19] in conjunction with a resource scheduler such as Globus [20] to form a two-layer RMS. Further, due to the expected scale of Grid systems, a Grid RMS is most likely to be an interconnection of various RMSs that are cooperating with one another within an accepted framework. Figure 2 shows a block diagram for a system with multiple interconnected RMSs and each RMS having multiple levels. User applications use the services of grid toolkits to implement their functionality. The grid toolkits use the RMS service interfaces to present suitable application layer abstractions. In the above figure, the same abstract model is used to represent the different instances of the RMSs and the different layers within an instance. However, the different instances may implement the abstract model distinctly.

Figure 3 shows the abstract model for the core functions supported by an RMS. The abstract model is intended to be generic enough to be specialized as an application-specific manager or resource-specific manager. The model contains several functional units and four interfaces: (1) *resource consumer interface* (RCI), (2) *resource provider interface* (RPI), (3) *resource manager support interface* (RMSI), and (4) *resource manager peer interface* (RMPI). The resource consumers that interface via RCI can be either actual applications or another RMS that implements a 'higher' layer. Similarly, the resource provider that interfaces via the RPI can be an actual resource or another RMS that implements a 'lower' layer. The support functions such as naming and security that are essential for the operation of the RMS interface via the RMSI. If needed, other support functions may also interface via RMSI. The RMPI provides the protocols to interconnect with other RMSs. Several protocols including resource discovery, resource dissemination, trading, resolution, and co-allocation may be supported by the RMPI.

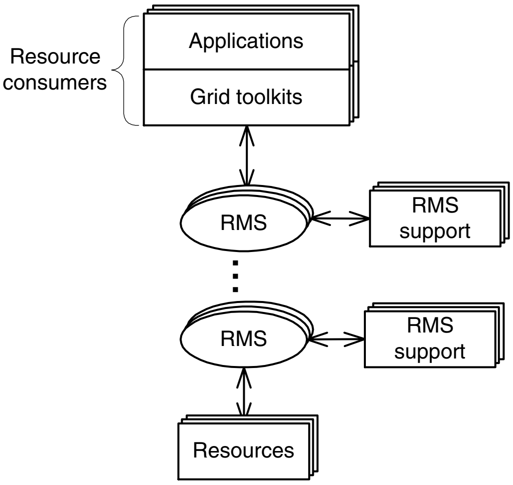

Figure 2. RMS system context.

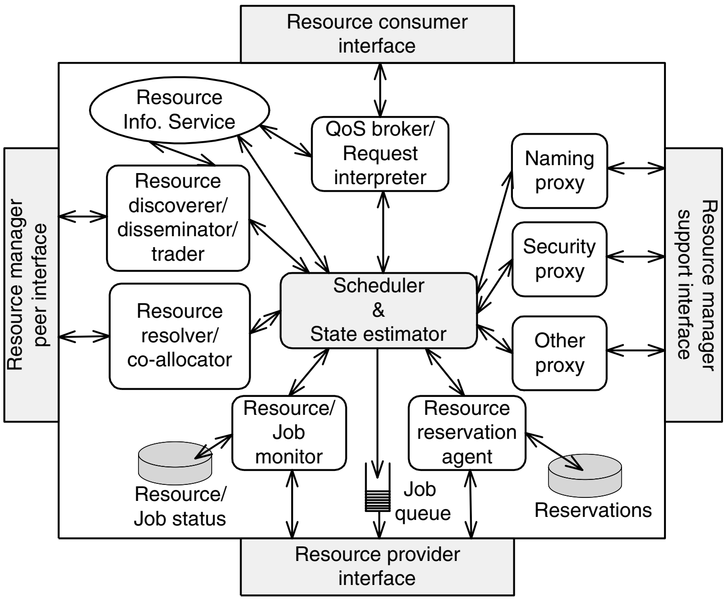
Resource consumer interface
Resource Info. Service
QoS broker/ Request interpreter
Naming proxy
Resource discoverer/ disseminator/ trader
Security proxy
Resource manager peer interface
Resource manager support interface
Resource resolver/ co-allocator
Scheduler & State estimator
Other proxy
Resource/ Job monitor
Resource reservation agent
Resource/ Job status
Job queue
Reservations
Resource provider interface

Figure 3. RMS system abstract structure.

The resource dissemination and discovery protocols provide a way for an RMS to determine the state of the resources that are managed by it and other RMSs that interoperate with it. The resource dissemination protocol provides information about the resources or a pointer to a information server. The resource discovery protocol provides a mechanism by which resource information can be found on demand. In some RMS architectures, no distinction is made between these two protocols. For example, an RMS could use a replicated network directory that contains resource information. The resource dissemination protocol could be implemented as a directory replication protocol. The resource discovery function would then consist of searching the nearest network directory. Alternatively, a Grid could maintain a central network directory where dissemination consists of advertising the resource status and discovery consists of querying the central directory.

Instead of resource dissemination and discovery protocols, some RMSs use resource trading protocols [11]. Typically, trading protocols encapsulate the resource capabilities and statuses using price. The resource trading takes place based on price differentials. The price itself may be fixed by auctioning mechanisms or price functions.

Other peer protocols include resource resolution and co-allocation protocols. Once an RMS becomes aware of a ‘remote’ resource that is more suitable to service a job than any of the local resources, it should contact the remote RMS to schedule the job at the remote resource. The resource resolution protocol among the RMSs supports this process. The operation of this protocol depends on the organization of the RMSs, i.e. whether the RMSs are hierarchically or flatly organized. For some jobs, it may be necessary to simultaneously acquire multiple resources. The resource co-allocation protocol supports this process. In some implementations, the resolution and co-allocation protocols may be combined together.

The RMSI is an extensible interface and supports functions that are essential for the operation of the RMS. In Figure 3, naming and security are placed as support functions with a proxy inside the RMS for efficient operation. The naming function enforces the namespace rules for resources and maintains a database for resource information. The structure, content, and maintenance of the resource database are important differentiating factors. An external instead of an internal naming function facilitates interoperability of the RMSs because the resources will have uniform naming.

The resource/QoS broker function is responsible for examining the resource requests that are specified in some *resource specification language* and translating the requests into the internal resource management protocols used within the RMS.

RMS TAXONOMY OVERVIEW

The taxonomy classifies RMSs by characterizing different attributes. The intent of the different parts of the taxonomy is to differentiate RMS implementations with a view to impact on overall Grid system scalability and reliability. Thus a RMS is classified according to machine organization within the Grid, resource model, dissemination protocols, namespace organization, data store organization, resource discovery, QoS support, scheduler organization, scheduling policy, state estimation, and rescheduling approach.

In the following sections, *resource requester* identifies the machine that is requesting a resource, *resource provider* identifies the machine that is providing the resource, and *resource controller*

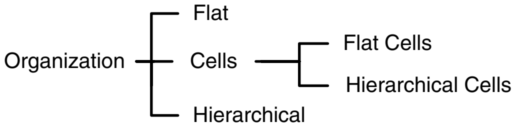

Figure 4. Organization taxonomy.

identifies the machine that is responsible for allocating the resource. Current Grid systems have machines that function in one or more of these roles.

MACHINE ORGANIZATION

The organization of the machines in the Grid affects the communication patterns of the RMS and thus determines the scalability of the resultant architecture. Figure 4 shows the taxonomy for the machine organization. The organization describes how the machines involved in resource management make scheduling decisions, the communication structure between these machines, and the different roles the machines play in the scheduling decision.

Previous taxonomies used centralized and decentralized categories. In a centralized organization a single controller or designated set of controllers performs the scheduling for all machines. This approach is not used in Grid systems since centralized organizations suffer from scalability issues. In a decentralized organization the roles are distributed among machines in the Grid. Decentralized organizations have been previously divided into sender and receiver initiated categories. This is too simple since there is a need to distinguish how the resource requesters, resource providers, and the resource controllers organize their dialogue.

For example, a resource requester may utilize an agent-based approach to search out resource providers and allocate the resources on behalf of the originator. This is a sender-initiated approach. A resource requester may consult locally or globally available resources and then makes requests to the resource providers. This is also a sender-initiated approach with a significantly different architecture than the agent-based approach. In this paper a different approach is used. The organization structure and resource management implementation are characterized separately. This section describes the organization and the next section describes the resource management implementation.

In a flat organization all machines can directly communicate with each other without going through an intermediary. In a hierarchical organization machines in the same level can directly communicate with the machines directly above them or below them, or peer to them in the hierarchy. The fan out below a machine in the hierarchy is not relevant to the classification. Most current Grid systems use this organization since it has proven scalability.

In a cell structure, the machines within the cell communicate between themselves using flat organization. Designated machines within the cell function acts as boundary elements that are responsible for all communication outside the cell. The internal structure of a cell is not visible from another cell, only the boundary machines are. Cells can be further organized in flat or hierarchical

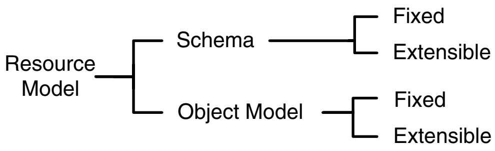

Figure 5. Resource model taxonomy.

structures. A Grid that has a flat cell structure has only one level of cells whereas a hierarchical cell structure can have cells that contain other cells. The major difference between a cell structure and hierarchical structure is that a cell structure has a designated boundary with a hidden internal structure whereas in a hierarchical structure the structure of the hierarchy is visible to all elements in the Grid.

RESOURCES

The taxonomies in this section describe the different aspects of the RMS that provide the interface to the resources managed by the Grid RMS. Arguably the interfaces to resource components are even more important than the scheduling components because they are ultimately what the applications or toolkits use to implement their functionality. If the resource interfaces do not provide the correct abstractions and efficient implementations it is unlikely that a Grid will be used effectively.

Resource model

The resource model determines how applications and the RMS describe Grid resources. Figure 5 shows the resource model taxonomy. The taxonomy is focused on how the resources and operations on the resources are described.

In a schema based approach, the data that comprises a resource is described via a description language along with some integrity constraints. In some systems, a query language is integrated with the schema language. The schema languages are further characterized by the ability to extend the schemas. In an extensible schema, new schema types for resource descriptions can be added. Predefined attribute-value based resource models are in the fixed schema category. The Condor ClassAd approach using semi-structured data approach is in the extensible schema category.

In an object model, the operations on the resources are defined as part of the resource model. As with schemas the object model can be predetermined and fixed as part of the definition of the RMS. In the object model extensible approach the resource model provides a mechanism to extend the definition of the object model managed by the RMS. The Legion approach provides an extensible object model oriented around resource management. It may be difficult to implement a high performance extensible object model. The current fixed object models are basic and provide few primitive operations on resources.

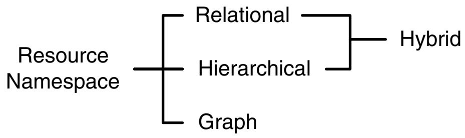

Figure 6. Namespace organization taxonomy.

Resource namespace organization

A Grid RMS operates on a pool of globally named resources. The organization of the namespace that could be created and maintained by an entity external to RMS greatly influences the design of the resource management protocols and affects the discovery methods. Figure 6 shows the taxonomy for namespace organization.

A relational namespace divides the resources into relations and uses concepts from relational databases to indicate relationships between tuples in different relations. A hierarchical namespace divides the resources in the Grid into hierarchies. The hierarchies are typically organized around the physical or logical network structures. The relational/hierarchical namespace hybrid consists of relations where the contents of the relations are broken into a hierarchy in order to distribute them across the machines in the Grid. Most network directory based namespaces utilize a hybrid structure. Globus MDS is an example of a network directory that utilizes a hybrid structure.

A graph-based namespace uses nodes and pointers where the nodes may or may not be complex entities. Namespaces that are implemented using an object-oriented paradigm typically use graph namespaces with object as nodes and inter-object references being the pointers. 2K provides a graph based namespace based on the CORBA object model. A given resource in a hierarchical namespace may occur more than once in different part of the hierarchy with an embedded reference to another part of the hierarchy. This is not considered a graph namespace since the fundamental navigation method is to descend the hierarchy rather than to chase references as in object-oriented approaches.

QoS support

In Grid systems, QoS is not limited to network bandwidth but extends to the processing and storage capabilities of the nodes. Thus the focus is on the degree a Grid can provide end-to-end QoS rather than providing only QoS on the network. When a Grid job has QoS requirements, it may be necessary to negotiate a *service level agreement* (SLA) to enforce the desired level of service. Resource reservation is one of the ways of providing guaranteed QoS in a Grid with dedicated resources. Globus provides some resource reservation capabilities. It is very inefficient to guarantee network bandwidth and not guarantee processing cycles for the application components communicating over this link. Resource reservation is also considered to be a fundamental QoS attribute. A Grid that provides the ability to specify QoS at job submission time but cannot reserve resources in advance is considered to provide only a partial solution to QoS. Most current Grid implementations can only provide partial QoS since most processor operating systems do not provide strict performance guarantees.

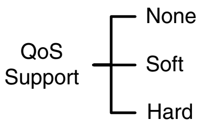

Figure 7. QoS support taxonomy.

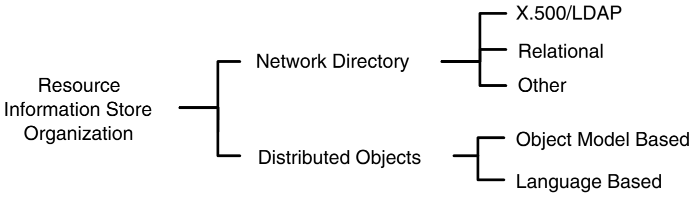

Figure 8. Resource information store taxonomy.

There are two parts to QoS, admission control and policing. Admission control determines if the requested level of service can be given and policing ensures that the job does not violate its agreed upon level of service. Figure 7 shows the taxonomy for QoS support. A RMS that provides explicit QoS attributes for resource requests but cannot enforce SLAs via policing provides soft QoS support. Hard QoS support is provided when all nodes in the Grid can police the SLAs guaranteed by the RMS. Darwin and GOPI provide hard QoS support since they are oriented towards supporting network applications.

Resource information store organization

The resource information store organization determines the cost of implementing the resource management protocols because resource dissemination and discovery may be provided by the data store implementation. Figure 8 shows the taxonomy for resource information data store organizations. Distributed object data stores utilize persistent object services that are provided by language independent object models such as CORBA or a language based model such as that provided by persistent Java object implementations. 2K and Javelin provide object-oriented information stores based around their respective underlying object models.

Network directory data stores are based on a database engine and utilize a standard interface language to operate on the schema data. Many existing network directories use X.500/LDAP interfaces to access an underlying relational database or a specialized distributed database implementation. Network directories may also provide access to underlying databases using query interfaces other than X.500/LDAP. Network directories implemented using relational databases may provide access using SQL or a restricted subset of it. The original implementation of Globus MDS was based on a standard LDAP implementation but this was converted to a specialized implementation to increase performance.

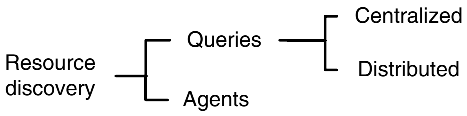

Figure 9. Resource discovery taxonomy.

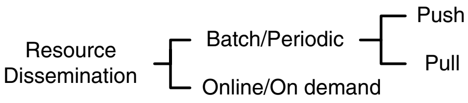

Figure 10. Resource dissemination taxonomy.

The important difference between distributed object and network directory approaches is that in network directories the schema and operations are separated with the operations defined externally to the data store schema. In an object-oriented approach the schema defines the data and the operations together. Other approaches such as Sun Microsystem's Jini provide a network information store oriented around the underlying language facilities. Jini utilizes the features of Java such as RMI and code mobility. Jini could be used to implement a more generalized Grid resource information store.

Resource discovery and dissemination

Resource discovery and dissemination may be viewed as providing complementary functions. Discovery is initiated by a network application to find suitable resources within the Grid. Dissemination is initiated by a resource trying to find a suitable application that can utilize it. Figure 9 shows the taxonomy for resource discovery and Figure 10 shows the taxonomy for resource dissemination.

The implementation of the resource description database in current systems seems to determine the approach to resource discovery. Network directory based systems mechanisms such as Globus MDS use parameterized queries that are sent across the network to the nearest directory. Query based systems are further characterized depending on whether the query is executed against a distributed database or a centralized database. The updating of the resource description database is characterized by the resource dissemination approaches. Agent based approaches send active code fragments across machines in the Grid that are interpreted locally on each machine. The Bond system is based on agents that use KQML as an agent communication language. Agents can also passively monitor and either periodically distribute resource information or in response to another agent. Thus agents can mimic a query based resource discovery scheme. The major difference between a query based approach and an agent based approach is that agent based systems allow the agent to control the query process and make resource discovery decisions based on its own internal logic rather than rely on an a fixed function query engine.

Resource dissemination is categorized by the approach taken for updating the resource information. The mechanisms used to implement dissemination determine the amount of data that is sent between machines in the Grid. For example, aggregate resource status may be sent using a different protocol than detailed resource description information in order to reduce the data transferred and the latency time.

In a batch/periodic approach, resource information is batched up on each Grid machine and then periodically disseminated through the Grid. Information can be sent from the originating machine to other machines in which case it is pushing the information or another machine in the Grid can request the information from the originating machine in which case it pulls the information from the machine. Condor and the European DataGrid are examples that utilize the batch approach. In an online or on-demand approach information is disseminated from the originating machine immediately. In this case the information is pushed to other machines in the Grid. The set of machines that the information is sent to depends on the organization of the Grid and the resource database implementation. The 2K system uses a demand based resource dissemination model.

Summary and research issues

This section presented taxonomies for describing resource models, resource namespace organization, QoS support, resource information store organization, resource discovery and resource dissemination. The solutions to these problems determine the functionality of the resources and the efficiency by which they can be managed by other components of the RMS. For example, the QoS support issue determines what kind of QoS support an application could receive from a resource. The organization of the resource information store could determine the efficiency of the scheduling or allocation algorithms. Following is a sampling of research issues in the area of the problems considered in this subsection that need further examination.

This paper assumes that the RMS operates on a ‘globally’ named pool of resources. Several current generation RMSs have naming as an internal function to the RMS. Further research is necessary to closely examine the trade-offs of placing the naming function. One motivation for making naming a global function is it facilitates interoperability of different RMSs which may be essential for the Grid to scale to Internet proportions.

Resource discovery and dissemination are two key functions for a wide-area RMS. The status information of the resources that is central to these functions change dynamically. Hence the above schemes should consider the fact that the information will become stale after a certain time period [21]. Further, a Grid RMS is likely to handle a highly heterogeneous set of resources with some more significant than others [22]. For dissemination schemes to reduce the message overhead and become scalable, issues such as how to quantify significance and measure it efficiently should be addressed.

QoS support is another significant issue in Grids. There is variety of resources that are capable of supporting QoS to varying degrees so it is essential to develop flexible QoS abstraction mechanisms.

SCHEDULING

The previous section examined several taxonomies that described the functionality of the resources both for the management system as well as the end-user applications. This section examines

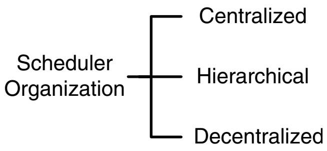

Figure 11. Scheduler organization taxonomy.

taxonomies that describe the scheduling and resource allocation operation of the RMS and discusses how these management schemes relate to some of the schemes examined in previous sections. In particular, taxonomies for the organization of the scheduling components, degree of extensibility, rescheduling approaches, and state estimation are presented.

Scheduler organization

The scheduling component of the RMS can be organized in three different ways as shown in Figure 11. In the centralized organization, there is only one scheduling controller that is responsible for the system-wide decision making. Such an organization has several advantages including easy management, simple deployment, and the ability to co-allocate resources. In a Grid RMS the disadvantages of this organization such as the lack of scalability, lack of fault-tolerance, and the difficulty in accommodating multiple policies outweigh the advantages. Condor utilizes a centralized scheme based around the ClassAd matchmaker.

The other two organizations, hierarchical [23] and decentralized have more suitable properties for a Grid RMS scheduler organization. In a hierarchical organization, the scheduling controllers are organized in a hierarchy. One obvious way of organizing the controllers would be to let the higher level controllers manage larger sets of resources and lower level controllers manage smaller sets of resources. Compared with the centralized scheduling this mode of hierarchical scheduling addresses the scalability and fault-tolerance issues. It also retains some of the advantages of the centralized scheme such as co-allocation. Many of the Grid systems such as 2K, Darwin, and Legion utilize a hierarchical scheduler.

One of the key issues with the hierarchical scheme is that it still does not provide site autonomy and multi-policy scheduling. This might be a severe drawback for Grids because the various resources that participate in the Grid would want to preserve control over their usage to varying degrees. Many Grid resources would not dedicate themselves only to the Grid applications. Therefore hierarchical scheduling schemes should deal with dynamic resource usage policies.

The decentralized organization is another alternative. It naturally addresses several important issues such as fault-tolerance, scalability, site-autonomy, and multi-policy scheduling. However, decentralized organizations introduces several problems of their own some of them include management, usage tracking, and co-allocation. This scheme is expected to scale to large network sizes but it is necessary for the scheduling controllers to coordinate with each other via some form of resource discovery or resource trading protocols. The overhead of operation of these protocols will be the determining factor for the scalability of the overall system. Lack of such protocols may reduce the overhead but the

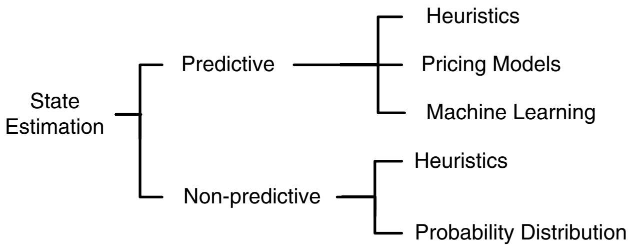

Figure 12. State estimation taxonomy.

efficiency of scheduling may also decrease. Systems such as Bond, MOL, and Ninf utilize decentralized scheduling approaches.

State estimation

Previous taxonomies of state estimation concentrated on the degree of information available for estimated state and the communication organization by which this information was distributed. In Grid systems, state estimation is always done on partial or stale information due to information propagation delay in large distributed systems. The focus of this taxonomy is on the mechanism used to estimate state that affects the implementation of the current and historical data stores in our abstract model. Figure 12 shows the state estimation taxonomy.

Non-predictive state estimation uses only the current job and resource status information since there is no need to take into account historical information. Non-predictive approaches use either heuristics based on job and resource characteristics or a probability distribution model based on an offline statistical analysis of expected job characteristics. A predictive approach takes current and historical information such as previous runs of an application into account in order to estimate state. Predictive models use either heuristic, pricing model or machine learning approaches. In a heuristic approach, predefined rules are used to guide state estimation based on some expected behavior for Grid applications. In a pricing model approach, resources are bought and sold using market dynamics that take into account resource availability and resource demand [2]. In machine learning, online or offline learning schemes are used to estimate the state using potentially unknown distributions. Note that heuristics or statistics-based techniques can be used for both predictive and non-predictive state estimation approaches. The Punch system uses machine learning based predictive approach and the Ninf system uses pricing model approach for state estimation.

Rescheduling

The rescheduling characteristic of a RMS determines when the current schedule is re-examined and the job executions reordered. The jobs can be reordered to maximize resource utilization, job throughput, or other metrics depending on the scheduling policy. The rescheduling approach determines the suitability of a RMS for different types of Grid systems. Figure 13 shows the rescheduling taxonomy.

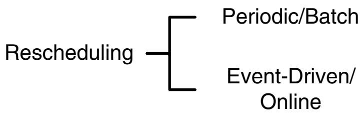

Figure 13. Rescheduling taxonomy.

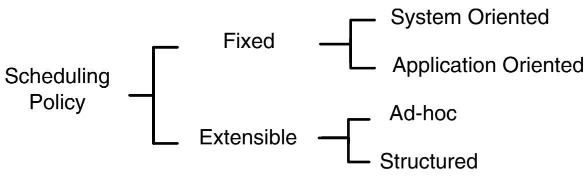

Figure 14. Scheduling policy taxonomy.

Periodic or batch rescheduling approaches group resource requests and system events which are then processed at intervals. This interval may be periodic or may be triggered by certain system events. The key point is that rescheduling is done to batches instead of individual requests or events. Event driven online rescheduling performs rescheduling as soon the RMS receives the resource request or system event.

Batch rescheduling allows potentially more effective utilization of the Grid resources since more requests can be considered at one time. Predictive state estimation schemes may also work better with periodic or batch rescheduling. Hard QoS would be difficult to provide using a batch rescheduling approach since the violation of service level would not cause immediate rescheduling of the offending job. Online schemes can be more reactive and show less latency for jobs. It may be quite difficult to implement a general-purpose online RMS scheme that can address the different types of Grid systems. It is difficult to determine the rescheduling approach of most Grid systems since it is usually not explicitly stated. Systems such as Darwin provide online scheduling whereas some application oriented Grids such as PUNCH seem to utilize a more batch oriented rescheduling approach.

Scheduling policy

The RMS uses the scheduling policy to determine the relative ordering of requests and jobs when rescheduling. Figure 14 shows the scheduling policy taxonomy. This taxonomy focuses the degree that the scheduling policy can be altered by entities outside the RMS. Large Grid systems with many different administrative domains will most likely have different resource utilization policies. Thus it is unlikely that an unalterable scheduling policy will suffice for the different needs.

In a fixed approach the policy implemented by the resource manager is predetermined. Fixed policies are further subdivided into maximizing system throughput schemes or maximizing application oriented schemes. Application oriented schemes try to optimize some specific metric such as application completion time. Some fixed policy schemes allow fine-tuning by providing specific control parameters to fine-tune the gross level scheduling objective such as maximizing overall resource utilization. Many application oriented schedulers such as Netsolve and AppLeS have more fixed scheduling policies.

Extensible scheduling policy schemes allow external entities the ability to change the scheduling policy. In the *ad hoc* extensible scheme the resource manager implements a fixed scheduling policy but provides an interface whereby an external agent can change the resulting schedule. This is typically done only for specific resources that demand special treatment. In a structured extensible scheme the resource manager provides a model of the scheduling process with associated semantics. External agents are allowed to override the default RMS supplied behaviors with their own thus changing the default scheduling policy. The Legion system provides a well structured extensible scheduling scheme. Most of the other Grid system provides some level of *ad hoc* scheduling extensibility.

Summary and research issues

This subsection discusses taxonomies that describe the approaches for scheduling and state estimation in an RMS. In a wide-area RMS such as a Grid RMS, a resource discovery/dissemination protocol is expected to work in conjunction with the state estimation. This concept is very different from previous generation DCEs such as Utopia [24]. In such systems the state estimation algorithms maintain the 'state' of the system which is used by the scheduling algorithm to allocate jobs to resources. In a wide-area RMS, the state estimation schemes may maintain distributed and independent status databases that may interoperate using the resource discovery/dissemination protocols.

Although scheduling is an area that has been investigated for a long time in distributed computing and other areas, the Grid computing systems present unique characteristics that demand a reexamination. Some of the issues in the area of scheduling include: development of Grid level resource container abstractions similar to the resource containers developed for uni-processor machines [25] and multi-policy scheduling schemes.

GRID RESOURCE MANAGEMENT SYSTEMS SURVEY

The example system surveyed in this section is not exhaustive, but comprehensive enough to cover many of the classes developed in our taxonomy. A summary of architectural design choices made by the representative set of Grid systems is shown in Table I.

The table groups the choices into three groups, Grid, Resources, and Scheduling. The Grid group classifies the system according to Grid type and the machine organization taxonomies. Some systems support more than one Grid type. The Resource group specifies the values for the resource taxonomies in the 'Resources' section. The Scheduling group specifies the values for the scheduling taxonomies presented in the 'Scheduling' section. Many systems are not fully classified since some of the information required to completely place them into taxonomies could not be determined from the references, they operate in conjunction with other Grid RMS systems, or they do not provide all the RMS features considered in this taxonomy.

Table I. Grid RMSs and their architecture choices.

System	Grid type	Resources	Scheduling
2K	On-demand Hierarchical	Extensible object model, graph namespace, soft network QoS, object model store, agent discovery, online dissemination	Hierarchical network resource scheduler, decentralized scheduler for other resources
AppLeS	High-throughput	Resource model provided by Globus, Legion, or Netsolve	Hierarchical scheduler, predictive heuristics, online rescheduling, fixed application oriented policy
Bond	On-demand Flat	Extensible object model, graph namespace, hard QoS, language based object store, agent discovery, periodic push dissemination	Decentralized scheduler, predictive pricing models, online rescheduling, fixed application oriented policy
Condor	Computational Flat	Extensible schema model, hybrid namespace, no QoS, network directory store, centralized queries discovery, periodic push dissemination	Centralized scheduler
Darwin	Multimedia Hierarchical	Fixed schema model, graph namespace, hard QoS	Hierarchical scheduler, non-predictive, online rescheduling, fixed system oriented policy
European DataGrid	Data Hierarchical	Extensible schema model, hierarchical namespace, no QOS, network directory store, distributed queries discovery, periodic push dissemination	Hierarchical scheduler, extensible scheduling policy
Globus	Various Hierarchical cell	Extensible schema model, hierarchical namespace, soft QoS, network directory store, distributed queries discovery, periodic push dissemination	Decentralized scheduler infrastructure, scheduling provided by external schedulers (AppLeS, Nimrod/G)
Javelin	Computational Hierarchical	Fixed object model, graph namespace, soft QoS, other network directory store, distributed queries discovery, periodic push dissemination	Decentralized scheduler, fixed application oriented policy
GOPI	Multimedia Flat	Extensible object model, graph namespace, hard QoS	Decentralized scheduler, *ad hoc* extensible policy
Legion	Computational Hierarchical	Extensible object model, graph namespace, soft QoS, object model store, distributed queries discovery, periodic pull dissemination	Hierarchical scheduler, extensible structured scheduling policy

Table I. Continued.

System	Grid type	Resources	Scheduling
MOL	Computational Hierarchical cell	Extensible schema model, hierarchical namespace, object model store, distributed queries discovery, periodic push dissemination	Decentralized scheduler, extensible *ad hoc* scheduling policies
NetSolve	Computational Hierarchical	Extensible schema model, hierarchical namespace, soft QoS, distributed queries discovery, periodic push dissemination	Decentralized scheduler, fixed application oriented policy
Nimrod/G	High-throughput hierarchical cell	Extensible schema model, hierarchical namespace, relational network directory data store, soft QoS, distributed queries discovery, periodic dissemination	Hierarchical decentralized scheduler, predictive pricing models, fixed application oriented policy
Ninf	Computational Hierarchical	Fixed schema model, relational namespace, no QoS, centralized queries discovery, periodic push dissemination	Decentralized scheduler
PUNCH	Computational Hierarchical	Extensible schema model, hybrid namespace, soft QoS, distributed queries discovery, periodic push dissemination	Hierarchical decentralized scheduler, predictive machine learning, fixed application oriented policy

2K: A distributed operating system

The 2K system is a distributed operating system [26,27] that provides a flexible and adaptable architecture for providing distributed services across a wide variety of platforms ranging from *personal digital assistants* (PDAs) to large scale computers. 2K is intended for development and deployment of distributed service applications rather than high performance grand challenge applications.

The core of 2K is a dynamic reflective CORBA *object request broker* (ORB) called dynamicTAO that is an extension of the TAO ORB [28]. The dynamicTAO ORB provides the ability to dynamically create environments for applications and move them across the 2K Grid machines using mobile reconfiguration agents. Code and service distribution is also managed using the 2K facilities.

The classification comes mainly from the use of CORBA as the underlying substrate for the system. The 2K system can be considered to be a demand service Grid that uses a flat RMS organization. In Reference [29], an extension of the current flat model to hierarchical model is described using CORBA traders and name servers. The 2K system uses an extensible object model with a graph based resource namespace and provides soft network QoS. The resource information store is object-model based using the CORBA object model. Resource discovery is performed through agents. Locating services and resource is also performed using the CORBA trading services. Resource dissemination is performed on demand by injecting mobile agents into the system.

The 2K system uses a one level hierarchical controller for scheduling network bandwidth. Other resources are scheduled locally using the *dynamic soft real time* (DSRT) scheduling at each resource provider. Thus 2K uses a decentralized controller for all other resources. There does not appear to be any state estimation function or rescheduling approach in the 2K system other than those provided by the underlying native operating system. The scheduling policy in the 2K system seems to be fixed.

AppLeS: A network enabled scheduler

The AppLeS [19] project primarily focuses on developing scheduling agents for individual applications on production computational Grids. AppLeS agents use application and system information to select a viable set of resources. AppLeS uses the services of other RMSs such as Globus, Legion, and NetSolve to execute application tasks. Applications have embedded AppLeS agents that performing resource scheduling on the Grid. AppLeS has been used in several applications areas including magnetohydrodynamics [30], gene sequence comparison, satellite radar images visualization, and tomography [31].

The AppLeS framework contains templates that can be applied to applications that are structurally similar and have the same computational model. The templates allow the reuse of the application specific schedulers in the AppLeS agents. Templates have been developed for parametric and master-slave type of applications.

The focus of AppLeS is on scheduling and thus it follows the resource management model supported by the underlying Grid middleware systems. An AppLeS scheduler is central to the application that performs mapping of jobs to resources, but the local resource schedulers perform the actual execution of application units. AppLeS schedulers do not offer QoS support. AppLeS can be classified with a predictive heuristic state estimation model, online rescheduling and fixed application oriented scheduling policy.

Bond: Java distributed agents

Bond is a Java based object-oriented middleware system for network computing [32]. Bond is based on agents [33] that communicate using the *knowledge querying and manipulation language* (KQML) for inter-object communication. KQML is used for inter agent communications rather than the lower level Java mechanisms in order to provide a uniform base of operation semantics between agents. Bond defines a uniform agent structure and agent extension mechanism. Agents are structured into finite state machines and strategy objects that define behavior in different states. External events cause state transitions that in turn trigger the strategy objects. Agents are dynamically assembled from components using a ‘blueprint’. Agents can be checkpointed and migrated by Bond. Agents can discover interface information via an interface discovery service that is accessed via a KQML message. Agent extensions are done using subprotocols. Subprotocols are small closed subsets of KQML commands and are used to dynamically extend the objects provided by the Bond library. Bond has a two level scheduler based on a stock market or computational economy approach. The Concerto extension of Bond [34] supports hard QoS and provides a general-purpose real time OS platform for multi-resource reservation, scheduling and signaling. Tempo is the bandwidth manager component and schedules network bandwidth. It is implemented as an extension of Sun Solaris.

Bond provides the infrastructure for an on-demand service Grid with a flat organization because there is no concept of autonomous domains with a border. Other organizations can be implemented on top of Bond but require extending the resource management function substrate provided by Bond. The resource model is the extensible object model, with hard QoS support, and a graph namespace. The resource information store is language-based distributed objects. Resources implement their interfaces in Java but exchange information between themselves using KQML. Resource discovery is agent based. Resource dissemination is accomplished through periodic push using probes. Schedulers are decentralized and use predictive pricing models for state estimation. Rescheduling is online. The scheduling policy seems to be fixed and application oriented.

Condor: Cycle stealing technology for high throughput computing

Condor [35,36] is a high-throughput computing environment that can manage a large collection of diversely owned machines and networks. Although it is well known for harnessing idle computers, it can be configured to share resources. The Condor environment follows a layered architecture and supports sequential and parallel applications. The Condor system allocates the resources in the Condor pool as per the usage conditions defined by resource owners. Through its remote system call capabilities, Condor preserves the job's originating machine environment on the execution machine, even if the originating and execution machines do not share a common file system and/or user ID scheme. Condor jobs with a single process are automatically checkpointed and migrated between workstations as needed to ensure eventual completion.

Condor can have multiple Condor pools and each pool follows a flat RMS organization. The Condor collector, which provides the resource information store, listens for advertisements of resource availability. A Condor resource agent runs on each machine periodically advertising its services to the collector. Customer agents advertise their requests for resources to the collector. The Condor matchmaker queries the collector for resource discovery that it uses to determine compatible resource requests and offers. Compatible agents contact each other directly and if they are satisfied the customer agents initiate computation on the resources.

Resource requests and offers are described in the Condor *classified advertisement* (ClassAd) language [37]. ClassAds use a semi-structured data model for resource description. The ClassAd language includes a query language as part of the data model, allowing advertising agents to specify their compatibility by including constraints in their resource offers and requests. Condor can be considered a computational Grid with a flat organization. It uses an extensible schema with a hybrid namespace. It has no QoS support and the information store is a network directory that does not use X.500/LDAP technology. Resource discovery is centralized queries with periodic push dissemination. The scheduler is centralized.

Darwin: Resource management for network services

Darwin is a customizable RMS [38] for creating value added network services. It is oriented towards resource management in network based equipment, but does provide mechanisms for scheduling computation in non-network nodes. Darwin provides a virtual network or mesh to distributed applications. An application provides an application input graph that describes the resource requirement. The input graph describes a set of end nodes and the network connections between them. The graph is annotated with QoS specifications that are used by Darwin in allocating resources. Darwin can provide hard network QoS since Darwin components run in routers and can control bandwidth at the network flow level using the built-in router functions.

The core component of Darwin is Xena, a request broker, which performs global allocation of resources. Control delegates perform runtime adaptations of the initial resource assignment. Data delegates operate on the network flows and provide services such as encryption, decryption, and data compression. Local resource managers provide low-level resource allocation and coordinate their activity using the Beagle signaling protocol [39]. Darwin uses a *hierarchical fair service curve scheduling* (H-FSC) algorithm for higher level resource allocation. The H-FSC algorithm was designed to efficiently support virtual networks for distributed applications.

The RMS is organized in Darwin as a one-level hierarchy because all requests are sent to a Xena request broker that interacts with its peer request brokers. The resource model is a fixed schema with hard QoS support and the resource namespace is a graph. Darwin does not provide a separate resource information store, resource discovery protocol, or resource dissemination protocol. Scheduling is hierarchical with non-predictive state estimation. Rescheduling is event driven and implemented by the control delegates. The scheduling policy is fixed and system oriented.

European DataGrid: Global physics data storage and analysis

The European DataGrid Project [12] focuses on the development of middleware services in order to enable distributed analysis of physics data. The core middleware system is the Globus toolkit with hooks for data Grids. Data on the order of several petabytes will be distributed in a hierarchical fashion to multiple sites worldwide. Global namespaces are required to handle the creation of and access to distributed and replicated data items. Special workload distribution facilities will balance the analysis jobs from several hundred physicists to different places in the Grid in order to have maximum throughput for a large user community. Application monitoring as well as collecting of user access patterns will provide information for access and data distribution optimization.

The DataGrid project has a multi-tier hierarchical RMS organization. For example, tier-0 is CERN, which stores almost all relevant data, several tier-1 regional centers (in Italy, France, U.K., U.S.A., Japan) will support smaller amounts of data, and so on. It has an extensible schema-based resource model with a hierarchical namespace organization. It does not offer any QoS and the resource information store is expected to be based on an LDAP network directory. Resource dissemination is batched and periodically pushed to other parts of the Grid. Resource discovery in the Data Grid is decentralized and query based. The scheduler uses a hierarchical organization with an extensible scheduling policy.

Globus: A toolkit for Grid computing

The Globus system enables modular deployment of Grid systems by providing the required basic services and capabilities in the *Globus Metacomputing Toolkit* (GMT). The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, data management, resource reservation, and communications. Globus is constructed as a layered architecture in which higher level services can be developed using the lower-level core services [40]. Its emphasis is on the hierarchical integration of Grid components and their services.

Globus offers Grid information services via an LDAP-based network directory called *Metacomputing Directory Services* (MDS) [41]. MDS currently consists of two components: *Grid Index Information Service* (GIIS) and *Grid Resource Information Service* (GRIS). GRIS provides resources discovery services on a Globus based Grid. The directory information is provided by a Globus component running on a resource or other external information providers. The resource information providers use a push protocol to update GRIS periodically. GIIS provides a global view of the Grid resources and pulls information from multiple GRIS to combine into a single coherent view of the Grid. Globus is placed into the push resource dissemination category since the resource information is initially periodically pushed from the resource providers. Resource discovery is performed by querying MDS.

Globus supports soft QoS via resource reservation [42]. The predefined Globus scheduling policies can be extended by using application level schedulers such as Nimrod/G, AppLeS, and Condor/G. The Globus scheduler in the absence of application level scheduler has a decentralized organization with an *ad hoc* extensible scheduling policy.

Javelin: Java parallel computing

Javelin [43] is a Java based infrastructure for Internet-wide parallel computing. Javelin consists of clients that seek resources, hosts that offer resources, and brokers that coordinate the allocations. Javelin supports piecework and branch and bound computational models. In the piecework model, adaptively parallel computations are decomposed into a set of sub-computations. The sub-computations are each autonomous in terms of communication, apart from scheduling work and communicating results. This model is suitable for parameter-sweep (master-work) applications such as ray tracing and Monte Carlo simulations. The branch-and-bound model achieves scalability and fault-tolerance by integrating distributed deterministic work stealing with a distributed deterministic eager scheduler. An additional fault-tolerance mechanism is implemented for replacing hosts that have failed or retreated.

The Javelin system can be considered a computational Grid for high-throughput computing. It has a hierarchical RMS organization where each broker manages a tree of hosts. The resource model is simple fixed objects with graph based namespace organization. The resources are simply the hosts that are attached to a broker.

Any host that wants to be part of Javelin contacts the JavelinBNS system, a Javelin information backbone that maintains a list of available brokers. The host communicates with brokers, chooses a suitable broker, and then becomes part of the broker's managed resources. Thus the information store is a network directory implemented by JavelinBNS components. Hosts and brokers update each other as a result of scheduling work thus Javelin uses demand-based resource dissemination. Resource discovery uses the decentralized query based approach since queries are handled by the distributed set of brokers. Javelin follows a decentralized approach in scheduling with a fixed application oriented scheduling policy.

Generic Object Platform Infrastructure (GOPI)

The GOPI project developed a platform based on CORBA with RM-ODP extension that provides an extensible architecture for adaptive multimedia applications [44,45]. GOPI provides an API and core services that are extended using network protocol extensions called application specific protocols. These extensions are stacked on top of transport protocols and implement application specific scheduling policies with the scheduling framework provided by GOPI. The resource namespace is based on the RM-ODP computational model and is specified using CORBA IDL. Reflective middleware and open bindings support QoS annotation on interfaces and the ability for an application to inspect and adapt its behavior to the underlying network.

The organization is flat with an extensible object-oriented resource model since it is based on CORBA. Thus the resource namespace is a graph. The system provides hard QoS if the underlying operating systems and network provide support. The research has been focused on delivering multimedia application support and thus lacks infrastructure for a resource information directory, resource discovery protocol, and resource dissemination protocol. A scheduler framework is provided into which application specific schedulers are loaded. The scheduler framework and scheduling extensions operate on a per node basis thus the scheduler organization is decentralized with an *ad hoc* extensible scheduling policy. State estimation and rescheduling are determined by the application specification extensions and thus cannot be classified.

Legion: A grid operating system

Legion [46] is an object-based metasystem that provides the software infrastructure for a Grid. In a Legion-based Grid, objects represent the different components of the Grid. The Legion objects are defined and managed by the corresponding class or metaclass. Classes create new instances, schedule them for execution, activate or deactivate the object, and provide state information to client objects. Each object is an active process that responds to method invocations from other objects within the system. Legion defines an API for object interaction, but does not specify the programming language or communication protocol.

Although Legion appears as a complete vertically integrated system, its architecture follows the hierarchical model. It uses an object-based information store organization through the Collection objects. Collections periodically pull resource state information from host objects. Host objects track load and users can call the individual host directly to get the resource information. Information about multiple objects is aggregated into Collection objects. Users or system administrators can organize collections into suitable arrangements.

All classes in Legion are organized hierarchically with LegionClass at the top and the host and vault classes at the bottom. It supports a mechanism to control the load on hosts. It provides resource reservation capability and the ability for application level schedulers to perform periodic or batch scheduling. Legion machine architecture is hierarchical with decentralized scheduler. Legion supplies default system oriented scheduling policies, but it allows policy extensibility via a structured scheduling extension interface.

Metacomputing Online (MOL)

The MOL [47] system follows a toolkit approach with the MOL-kernel as the central component. The MOL-kernel services provide resource management supporting dynamic communication, fault management, and access provision.

The MOL-kernel follows a three-tier architecture consisting of resource abstraction, management, and access layers containing *resource module* (RM), *center management modules* (CMMs), and *access module* (AM) respectively along with customizable and predefined event handlers. The RMs encapsulate metacomputing resources or services. Different RMs in an institution are coordinated by a CMM. The CMM is responsible for keeping the network components and resources in a consistent state. CMM also acts as a gatekeeper to control access to the resources from external networks. Large institutions can have multiple CMMs per institution.

If any MOL-kernel components fail, only one institute becomes inaccessible. As long as a single CMM is available, the MOL-kernel remains operational. The MOL-kernel dynamically reconfigures itself as the organizations come online and go offline. The MOL-kernel guarantees that the collective CMMs maintain consistent state by using a transaction-oriented protocol on top of the virtual shared memory objects associated with each CMM. Mirror instances of shared active objects are maintained at each CMM in order to make the global state available at all entry points.

Higher-level functionality of the MOL-kernel is based on typed messages and event handlers. Any module in the kernel can send messages to any other component. This is usually triggered by either a user interacting with an AM, by another incoming message or by a changed state at one of the available resources or services. The event management unit associated with each kernel module checks for incoming messages and invokes appropriate event handlers.

The MOL follows a service Grid model with hierarchical cell-based machine organization. It has a schema-based resource model with a hierarchical name space organization. The object-based resource state information is maintained in the shared objects of each CMM. The resources and services themselves utilize a push protocol for resource dissemination. The AMs/schedulers perform resource discovery and scheduling by querying shared objects.

NetSolve: A network-enabled computational kernel

Netsolve [48] is a client–agent–server paradigm based network-enabled application server. It is designed to solve computational science problems in a distributed environment. The Netsolve system integrates network resources including hardware and computational software packages into a desktop application. Netsolve clients can be written in C, FORTRAN, Matlab or Web pages to interact with the server. A Netsolve server can use any scientific package to provide its computational software. Communications between Netsolve clients, agents, and servers are performed using TCP/IP sockets. Netsolve agents can search for resources on a network, choose the best one available, execute the client request, and then return the answer to the user.

The Netsolve system is a computational Grid with hierarchical based machine organization. Netsolve agents maintain information about resources available in the network. The Netsolve servers which are the resources in a Netsolve Grid are responsible for making their existence aware to Netsolve Agents and thus use a push protocol for resource dissemination. Netsolve Agents also perform resource discovery and scheduling. An agent may request assistance of other Agents in identifying the best resources and scheduling. Thus Netsolve can be considered to use a network directory implemented by the Netsolve agents, decentralized scheduling with a fixed application oriented scheduling policy.

Nimrod/G: Resource broker and economy grid

Nimrod/G [2,11] is a Grid resource broker for managing and steering task farming applications such as parameter studies on computational Grids. It follows a computational market-based model for resource management. Nimrod/G provides support for formulation of parameter studies, a single window to manage and control experiments, resource discovery, resource trading, and scheduling. The task-farming engine of Nimrod/G coordinates resource management and results gathering. This engine can be used for creating user-defined scheduling policies. For example, ActiveSheets are used to execute Microsoft Excel computations/cells on the Grid [49]. Nimrod/G is being used as a scheduling component in a new framework called *Grid Architecture for Computational Economy* (GRACE) which is based on using economic theories for a Grid resource management system.

Nimrod/G has a hierarchical machine organization and uses a computational market model for resource management [2]. It uses the services of other systems such as Globus and Legion for resource discovery and dissemination. State estimation is performed through heuristics using historical pricing information. The scheduling policy is fixed-application oriented and is driven by user-defined requirements such as deadline and budget limitations. Load balancing is performed through periodic rescheduling.

Ninf: A network enabled server

Ninf is a client–server based network infrastructure for global computing [50]. The Ninf system functionality is similar to NetSolve. Ninf allows access to multiple remote compute and database servers. Ninf clients can semi-transparently access remote computational resources from languages such as C and FORTRAN. Programmers can build a global computing application by using the Ninf remote libraries as its components, without being aware of the complexities of the underlying system they are programming. Procedural parameters, including arrays, are marshaled and sent to the Ninf server on a remote host responsible for executing the requested library functions and returning the results. The Ninf client library calls can be synchronous or asynchronous in nature.

The key components of Ninf system include Ninf client interfaces, Ninf Metaserver, and the Ninf remote libraries. Ninf applications invoke Ninf library functions which generate requests that go to the metaserver that maintains a directory of Ninf servers in the network. The metaserver allocates resources for the remote library calls by routing them to appropriate servers by querying its information store on behalf of the client request. Thus Ninf utilizes centralized query-based resource discovery. Ninf computational resources themselves register details of available library services with the Ninf metaserver thus using a push protocol for resource dissemination. Ninf follows a flat model in machine organization, a fixed schema resource model, and has a relational name space organization. The Ninf metaserver performs some level of scheduling by routing the requests but the actual scheduling of client request is performed by the server thus resulting in a decentralized scheduler organization.

The Purdue University Network Computing Hubs (PUNCH)

PUNCH [51,52] is a middleware testbed that provides operating-system services in a network-based computing environment. It provides transparent access to remote programs and resources, access control and job control functionality in a multi-user, multi-process environment. PUNCH supports a virtual Grid organization by supporting decentralized and autonomous management of resources.

The PUNCH infrastructure consists of a collection of services that allow management of applications, data, and machines distributed across wide-area networks. User applications are accessed via standard Web browsers. Applications do not have to be written in any particular language and access to source or object code is not required thus allowing PUNCH to use existing applications and Grid enabling them.

PUNCH employs a hierarchically distributed architecture with several layers. A computing portal services layer provides Web-based access to a distributed, network-computing environment. This layer primarily deals with content management and user-interface issues. A network OS layer provides distributed process management and data browsing services. An application middleware layer allows the infrastructure to interoperate with other application-level support systems such as the Message Passing Interface [53]. A virtual file system layer consists of services that provide local access to distributed data in an application-transparent manner. Finally, an OS middleware layer interfaces with local OS services available on individual machines or clusters of machines. The layers interoperate with a distributed RMS and a predictive performance modeling sub-system in order to make intelligent decisions in terms of selecting from different application implementations, data storage sites, and hardware resources. For example PUNCH can choose between a sequential or parallel implementation running on a dedicated server or a Condor pool [35].

PUNCH uses a hierarchical decentralized scheduler, predictive machine learning state estimation for matching user requests to the most appropriate Grid resources, and fixed application oriented scheduler policy.

DISCUSSION AND CONCLUSIONS

There are many different approaches and models for developing Grid resource-management systems. The systems surveyed have for the most part focused on either a computational Grid or a service Grid. The only data Grid project surveyed is the European DataGrid Project which is in the initial stages of development. The other category of system is the Grid scheduler such as Nimrod/G and AppLeS that is integrated with another Grid RMS such as Globus or Legion. These combinations are then used to create application oriented computational Grids that provide certain levels of QoS.

Extensibility of the resource model is a feature of all the surveyed RMS with the exception of Darwin and Javelin. Darwin is oriented towards network services and thus does not require extensibility and Javelin is oriented towards Internet parallel computing and thus does not require a rich resource model. The degree of extensibility is quite different between systems. Extensible schema-based models range from the semi-structured data models of Condor to the LDAP based structure for Globus. Object-based model extensibility typically follows what is available in the underlying technology, which is either CORBA or Java. The Legion system is an exception to this since it builds its object model from its own primitives. A topic for further research is to investigate the extent to which the extensible features of the various resource models are used by Grid applications. It is possible that the resource model extensions are only used internally by the RMS components.

Most systems employ a periodic push approach to resource dissemination within a hierarchical machine organization. The resource discovery approach is correlated with the resource model. Schema-based system use queries whereas object model sometimes uses an agent-based approach. There are no systems that we are aware of that use a schema-based resource model with agent-based queries. A topic of further research is to investigate the relative efficiency of different dissemination schemes in conjunction with the machine organization and resource model.

The survey indicates that most of the different aspects of the taxonomy have been explored in the different systems. The flat and hierarchical machine organizations are quite common. Cell-based organizations have so far appeared in only schema-based resource models. Different resource models have been developed but there have been no comparisons between the performances of the schemes. The scheduling aspect of the taxonomy has not been fully explored. A decentralized, online rescheduling system with an extensible scheduling policy has, to our knowledge, not been developed for a Grid system.

This paper has presented a taxonomy for Grid RMSs. Requirements for RMSs have been described and an abstract functional model has been developed. The requirements and model have been used to develop the taxonomy. The taxonomy has focused on the type of Grid system, machine organization, resource model characterization, and scheduling characterization. Representative Grid systems have been surveyed and placed into their various categories. This has helped to identify some of the key Grid resource management approaches and issues that are yet to be explored as topics of future research.

ACKNOWLEDGEMENTS

The authors would like to acknowledge all developers of the Grid systems described in the paper. In particular, we thank Jim Basney, Henri Casanova, Geoff Coulson, Ian Foster, Colin Gan, Joern Gehring, Nirav Kapadia, John Karpovich, Fabio Kon, Jarek Nabrzyski, Hidemoto Nakada, Michael Neary, Dan Marinescu, Andre Merzky, Omer Rana, Heinz Stockinger, and Achim Streit for their help during the preparation of the manuscript.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Fransisco, CA, 1999.
2. Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications. *Proceedings of the 2nd International Workshop on Active Middleware Services (AMS '00)*, August 2000.
3. Sinha PK. *Distributed Operating Systems: Concepts and Design*. IEEE Press: New York, NY, 1997.
4. Maheswaran M. Quality of service driven resource management algorithms for network computing. *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA '99)*, June 1999; 1090–1096.
5. Casavant TL, Kuhl JG. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering* 1988; **14**(2):141–154.
6. Rotithor HG. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proceedings on Computer and Digital Techniques* 1994; **141**(1):1–10.
7. Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. *Proceedings IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1998; 330–335.
8. Ekemecic I, Tartalja I, Milutinovic V. A survey of heterogeneous computing: Concepts and systems. *Proceedings of the IEEE* 1996; **84**(8):1127–1144.
9. Kapadia NH. On the design of a demand-based network computing system: The Purdue Network Computing Hubs. *PhD Thesis*, School of Electrical and Computer Engineering, Purdue University, 1999.

10. Abramson D, Giddy J, Kotler L. High performance parametric modeling with nimrod/g: Killer application for the global grid? *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, April 2000; 520–528.
11. Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. *Proceedings of the International Conference on High Performance Computing in Asia–Pacific Region (HPC Asia 2000)*, 2000.
12. Hoscheck W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K. Data management in an international data Grid project. *Proceedings 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000)*, December 2000; 77–90.
13. Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S. The data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications* **23**(3):187–200.
14. Nahrstedt K, Chu H, Narayan S. QoS-aware resource management for distributed multimedia. *Journal of High-Speed Networking (Special Issue on Multimedia Networking)* December 2000; **7**(3–4):227–255.
15. Buyya R, Chapin S, DiNucci D. Architectural models for resource management in the Grid. *Proceedings 1st IEEE/ACM International Workshop on Grid Computing (Grid '00)*, December 2000; 18–35.
16. Vahdat A. Toward wide-area resource allocation. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 2, June 1999; 930–936.
17. Berman F. High-performance schedulers. *The Grid: Blueprint for a New Computing Infrastructure*, Foster I, Kesselman C (eds.). Morgan Kaufmann: San Fransisco, CA, 1999; 279–310.
18. Livny M, Raman R. High-throughput resource management. *The Grid: Blueprint for a New Computing Infrastructure*, Foster I, Kesselman C (eds.). Morgan Kaufmann: San Fransisco, CA, 1999; 311–338.
19. Berman F, Wolski . The AppLeS project: A status report. *Proceedings of the 8th NEC Research Symposium*, May 1997.
20. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing* 1997; **11**(2):115–128.
21. Mitzenmacher M. How useful is old information. *IEEE Transactions on Parallel and Distributed Systems* 2000; **11**(1):6–20.
22. Maheswaran M, Krauter K. A parameter-based approach to resource discovery in Grid computing systems. *Proceedings 1st IEEE/ACM International Workshop on Grid Computing (Grid '00)*, December 2000; 181–190.
23. Regehr J, Stankovic J, Humphrey M. The case for hierarchical schedulers with performance guarantees. *Technical Report CS-2000-07*, University of Virginia, 2000.
24. Zhou S, Zheng X, Wang J, Delisle P. Utopia: A load sharing facility for large, heterogeneous distributed computing systems. *Software—Practice and Experience* 1993; **23**(12):1305–1336.
25. Banga G, Druschel P, Mogul J. Resource containers: A new facility for resource management in server systems. *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI '99)*, February 1999.
26. Carvalho D, Kon F, Ballesteros F, Romn M, Campbell R, Mickunas D. Management of execution environments in 2K. *Proceedings of the 7th International Conference on Parallel and Distributed Systems (ICPADS '00)*, July 2000; 479–485.
27. Kon F, Campbell R, Mickunas M, Nahrstedt K. 2K: A distributed operation system for dynamic heterogeneous environments. *Proceedings 9th IEEE International Symposium on High Performance Distributed Computing (HPDC '00)*, August 2000; 201–210.
28. Kon F, Romn M, Liu P, Mao J, Yamane T, Magalhes LC, Campbell RH. Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '00)*, April 2000; 121–143.
29. Kon F, Yamane T, Hess C, Campbell R, Mickunas MD. Dynamic resource management and automatic configuration of distributed component systems. *Proceedings of the USENIX Conference on Object-Oriented Technologies and Systems (COOTS '01)*, February 2001.
30. Dail H, Obertelli G, Berman F, Wolski R, Grimshaw A. Application-aware scheduling of a magnetohydrodynamics application in the Legion metasystem. *Proceedings 9th IEEE Heterogeneous Computing Workshop (HCW '00)*, May 2000.
31. Smallen S, Cirne W, Frey J, Berman F, Wolski R, Su M, Kesselman C, Young S, Ellisman M. Combining workstations and supercomputers to support Grid applications: The parallel tomography experience. *Proceedings 9th IEEE Heterogeneous Computing Workshop (HCW '00)*, May 2000.
32. Boloni L, Marinescu DC. An object-oriented framework for building collaborative network agents. *Intelligent Systems and Interfaces*, Kandel A, Hoffmann K, Mlynek D, Teodorescu NH (eds.). Kluwer, 2000; 31–64.
33. Jun K, Boloni L, Palacz K, Marinescu DC. Agent-based resource discovery. *Proceedings 9th IEEE Heterogeneous Computing Workshop (HCW '00)*, October 1999.
34. Yau D, Marinescu DC, Jun K. Middleware QoS agents and native kernel schedulers for adaptive multimedia services and cluster servers. *Proceedings of the Real-Time System Symposium 99*, 1999.
35. Litzkow M, Livny M, Mutka MW. Condor—A hunter of idle workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

36. Basney J, Livny M. Deploying a high throughput computing cluster. *High Performance Cluster Computing*, vol. 1, Buyya R (ed.). Prentice Hall: Upper Saddle River, NJ, 1999; 116–134.
37. Raman R, Livny M. Matchmaking: Distributed resource management for high throughput computing. *Proceedings 7th IEEE International Symposium on High Performance Distributed Computing*, July 1998.
38. Chandra P, Fisher A, Kosak C, Ng TSE, Steenkiste P, Takahashi E, Zhang H. Darwin: Customizable resource management for value-added network services. *Proceedings 6th IEEE International Conference on Network Protocols*, October 1998.
39. Chandra P, Fisher A, Steenkiste P. A signaling protocol for structured resource allocation. *IEEE Infocomm*, March 1999.
40. Czajkowski K, Foster I, Kesselman C, Karonis N, Martin S, Smith W, Tuecke S. A resource management architecture for metacomputing systems. *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
41. Fitzgerald S, Foster I, Kesselman C, von Laszewski G, Smith W, Tuecke S. A directory service for configuring high-performance distributed computations. *Proceedings Sixth IEEE Symposium on High Performance Distributed Computing*, 1997; 365–375.
42. Foster I, Roy A, Sander V. A quality of service architecture that combines resource reservation and application adaptation. *Proceedings of the 8th International Workshop on Quality of Service (IWQoS '00)*, June 2000.
43. Neary M, Phipps A, Richman S, Cappello P. Javalin 2.0: Java-based parallel computing on the Internet. *Proceedings of the European Parallel Computing Conference (Euro-Par 2000)*, 2000.
44. Coulson G. A configurable multimedia middleware platform. *IEEE Multimedia* 1999; **6**(1):62–76.
45. Coulson G, Clarke M. A distributed object platform infrastructure for multimedia applications. *Computer Communications* 1998; **21**(9):802–818.
46. Chapin S, Karpovich J, Grimshaw A. The Legion resource management system. *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, April 1999.
47. Gehring J, Streit A. Robust resource management for metacomputers. *Proceedings 9th IEEE International Symposium on High Performance Distributed Computing*, 2000.
48. Casanova H, Dongarra J. Netsolve: A network-enabled server for solving computational science problems. *International Journal of Supercomputer Applications and High Performance Computing* 1997; **11**(3):212–223.
49. Kotler L, Abramson D, Roe P, Mather D. Activesheets: Super-computing with spreadsheets. *Proceedings of the 2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference*, April 2001.
50. Nakada H, Sato M, Sekiguchi S. Design and implementation of Ninf: Towards a global computing infrastructure. *Future Generation Computing Systems (Metacomputing Special Issue)* October 1999; **15**(5–6):649–658.
51. Kapadia N, Fortes J. PUNCH: An architecture for web-enabled wide-area network-computing. *Cluster Computing: The Journal of Networks, Software Tools and Applications; Special Issue on High Performance Distributed Computing* September 1999; **2**(2):153–164.
52. Kapadia N, Figueiredo R, Fortes J. PUNCH: Web portal for running tools. *IEEE Micro* May–June 2000; **20**(3):38–47.
53. Gropp W, Lusk E, Doss N, Skjellum A. A high performance, portable implementation of the message passing interface (mpi) standard. *Parallel Computing* 1996; **22**(6).