A Taxonomy and Survey of Grid Resource Management Systems

Klaus Krauter[1], Rajkumar Buyya[2], and Muthucumaru Maheswaran[1]
Advanced Networking Research Laboratory[1]
Department of Computer Science
University of Manitoba, Canada
krauter@cs.umanitoba.ca, maheswar@cs.umanitoba.ca

School of Computer Science and Software Engineering[2]
Monash University
Melbourne, Australia
rajkumar@csse.monash.edu.au

Technical Report: University of Manitoba (TR-2000/18) and Monash University (TR-2000/80)

Abstract – *The resource management system is the central component of 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, we develop a comprehensive taxonomy for describing resource management architectures. We use this taxonomy in identifying approaches followed in the implementation of real resource management systems for large-scale network computing systems known as Grids. We use the taxonomy and the survey results to identify architectural approaches that have not been fully explored in the research.*

Keywords: *Metacomputing, Grids, Resource Management, Scheduling, and Internet Computing.*

1. Introduction

A *network computing system* (NCS) is a virtual computer formed by a networked set of heterogeneous machines that agree to share their local resources with each other. A *Grid* is a very large-scale network computing system that scales to Internet size environments with machines distributed across multiple organizations and administrative domains. Machines in a Grid are typically grouped into autonomous administrative domains that communicate via high-speed communication links. A good overview of Grid technology is presented in [21].

The *resource management system* (RMS) is central to the operation of a Grid. *Resources* are the entities such as processors and storage that are managed by the RMS. The set of services provided by a RMS varies depending on the intended purpose of the Grid. The basic function of a RMS is to accept requests for resources from machines within the Grid and assign specific machine resources to a request from the overall pool of Grid resources for which the user has access permission. A RMS matches requests to resources, schedules the matched resources, and executes the requests using the scheduled resources. *Jobs* are the entities that utilize resources to execute network applications in the Grid environment. In this paper we will use the terms job and resource request interchangeably. The process of matching, scheduling, and execution is performed so that some metric of aggregate *quality of service* (QoS) delivered to the requestor is maximized [23]. This paper develops a taxonomy that describes the RMS architectures for a Grid.

In the next section the motivation for our taxonomy is developed. We follow this by developing a classification of Grid systems based on the design objectives and target applications since these greatly influence the architecture of the Grid RMS. Requirements and an abstract model for resource management systems are presented in the next section. The requirements and the abstract model are used as a basis to develop our taxonomy. We then describe our taxonomy for heterogeneous network computing systems. A survey of representative network computing and Grid systems is provided followed by some conclusions and suggestions for further research.

2. Motivation for the Taxonomy

A distributed computing scheduling taxonomy is presented in [26]. This taxonomy includes static scheduling techniques that we do not address and does not consider scheduling, state estimation, and resource models separately when classifying dynamic scheduling. The taxonomy for dynamic scheduling presented in [27] only considers two aspects of resource management, scheduling and state estimation. In our taxonomy, we provide a classification of resource models and examine scheduling and state estimation at a finer level. Several advances in distributed resource management since the publication of [27] have also been incorporated into our taxonomy.

A taxonomy for *heterogeneous computing* environments is presented in [29]. The taxonomy covers application model, target platform model, and mapping heuristic model. We do not cover application models or differentiate on target platform model since our focus is on issues relevant to the designers of RMS rather than application and toolkit designers.

Several taxonomies for characterizing a distributed system are presented in [25] and [28]. The EM^3 taxonomy in [25] classifies a heterogeneous computing system based on the number of execution modes and machine models. An extended version of the taxonomy developed in [26] is also presented in [25] to characterize the scheduling algorithms in heterogeneous computing systems. Our taxonomy focuses on RMS design issues and thus differs from the taxonomies presented in [25]. The taxonomy presented in [28] 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 [28].

3. Grid Systems

3.1. System Elements

A resource management system manages the elements that comprise the computing environment. Thus it is important to describe the elements that comprise a Grid. Computing systems are evolving from relatively uniform monolithic elements uniformly connected by low speed communication links to highly diverse specialized elements connected by diverse high-speed communication links. For example, today’s networked systems contain personal computers, personal digital assistants, parallel computers, network routers, network switches, clustered server farms, and network attached storage devices such as fiber channel RAID, automated tape libraries, and CD-RW/DVD jukeboxes.

In this paper the system elements that comprise a Grid will be categorized as being either a processing element, a network element, or a storage element. Processing elements can be further classified into uniprocessor, multiprocessor, cluster, and parallel processing elements. Another possible classification depends on whether the processing element is fixed or mobile. Network elements are the routers, switches, voice gateways, virtual private network devices and firewalls. Storage elements are the network attached storage devices such as fiber channel RAID devices, automated CD-ROM/DVD jukeboxes, tape libraries, or a dedicated database machine. Note that a dedicated storage element may sometimes have an attached computing element through which older style storage elements such as NFS storage servers are attached to the network. Any particular physical machine in a network computing system can be placed into at least one of these categories based on its intended function within the overall system.

3.2. Classification

The design objectives, target applications, and machine environment for a Grid determine the architecture of the RMS. In this paper, the design objectives are grouped into three different themes: improving application performance, data access, and enhanced services. Network computing systems with similar design objectives usually have similar characteristics. Using these themes, we can group Grid systems into several categories as shown in Figure 1.

The *computational Grid* category denotes systems that have a higher aggregate computational capacity available for single applications than the capacity of any constituent machine in the system. These can be further subdivided into *distributed supercomputing* and *high throughput* categories depending on how the aggregate capacity is utilized. A distributed supercomputing Grid executes the application in parallel on multiple machines to reduce the completion time of a job. Typically, applications that require a distributed supercomputer are grand challenge problems. Examples of grand challenge problems are fluid dynamics, weather modeling, nuclear simulation, molecular modeling, and complex financial analysis. A high throughput Grid increases the completion rate of a stream of jobs. Applications that involve parameter-study to explore a range of possible design scenarios such as ASIC or processor design verification tests would be run on a high throughput Grid [9].

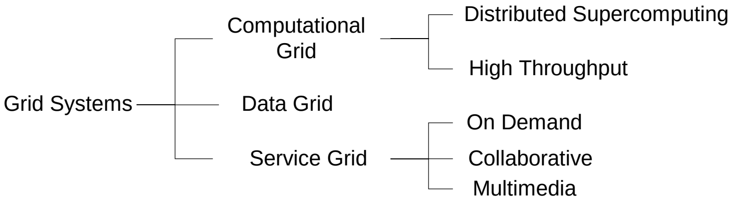

Figure 1: Grid Systems Taxonomy.

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 would be special purpose data mining that correlates information from multiple different data sources and then processes it further. For example, high energy physics applications need to process massive data generated (in the order of GB/second) by an experiment that will be using the LHC (Large Hadron Collider) from 2005 onwards [13]. The two popular DataGrid initiatives, CERN DataGrid [13] and Globus [52], 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 in *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. A multimedia Grid provides an infrastructure for real-time multimedia applications. This requires supporting quality of service across multiple different machines whereas a multimedia application on a single dedicated machine can be deployed without QoS [49]. 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 system.

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.

4. Resource Management Systems

4.1. Requirements

A resource management system that is ubiquitous and able to deal with the diverse requirements of network aware applications needs to schedule and control resources on any element in the network computing system environment. A network application would specify the type of resources required such as the various types of processing elements, storage elements, and the communication requirements between the elements. The type and topology of the resources is not sufficient to describe the requirements of the application. The application also needs to specify the required minimum resource levels. This ranges from network *quality of service* (QoS), to processor speed, to data storage requirements on a storage element. In this paper, we will label these as quality of service requirements on Grid system elements.

The network application cannot always specify the resources required since this is dependent on the input parameters to the application and the execution. In many cases the application can provide only a partial specification of the resources and required quality of service. For example, a collaborative workspace application could put strict quality of service requirements on the audio-visual streams for the human participants but loosely define the quality of service required for the visual rendering of the data.

The resource management system should predict the impact that an application's request will have on the overall resource pool and quality of service guarantees already given to other applications. This prediction can be done either by using past behavior of the application on similar input values or by using heuristics. The other option is to ignore predictions and develop the resource usage mechanisms according to some externally defined policy rules.

In order to provide QoS, a resource management system must be able to perform admission control and policing. Admission control is the process of determining if a resource request can be honored. Policing is ensuring the contract for resource utilization agreed upon between the resource management system and the requesting application has not exceeded. All resource managers implicitly perform some level of admission control during the scheduling process. The requirement to police the resource utilization impacts the design of the RMS. Issues that need to be addressed are the rate at which the policing action is performed and the degree to which the RMS can control resource utilization. This determines the quality of service guarantees a resource manager can offer for any particular resource.

A RMS performs accrediting, reclamation, allocation and naming of resources. Accrediting is the ability to track resource usage to the jobs. Reclamation is the ability to reclaim a resource when the job that was using the resource has finished. Allocation is the ability to allocate resource to jobs in a way that does not violate imposed resource limits. The naming of resources is done via one or more namespaces that uniquely identify managed resources within the Grid. Jobs use the namespaces to identify the resources that are required to carry out the computation.

The RMS is also responsible for ensuring the integrity of the underlying resource and thus enforces the security of resources. The resource management system operates in conjunction with a security manager. The resource manager authenticates and authorizes all resource requests with the security manager using the credentials provided by the job. The resource manager does not ensure the integrity of the credentials or manage the authorization lists since this is the function of the design of the security system and security manager. A secure channel for exchanging information between the RMS and the security manager is assumed to be part of underlying Grid infrastructure.

4.2. Abstract Model

A resource management system in a Grid provides three fundamental services, resource dissemination, resource discovery, and scheduling resources for the execution of jobs. The RMS works in conjunction with other Grid components as shown in Figure 2. Applications use Grid toolkits to describe the application resource requirements. Grid toolkits consist of compilers, development environments, and runtime systems. The toolkits provide higher-level application oriented resource management services. The toolkits in turn issue resource requests to the RMS. The RMS uses the services of the native operating systems (including queuing systems) to execute applications. The RMS interfaces with a security component that implements the authorization, authentication and non-repudiation functions of the Grid. The RMS provides the raw resource utilization data that is used by the billing and accounting systems.

The RMS manages the allocation, scheduling, and reclamation of resources from jobs. The RMS assigns resources to jobs in response to resource requests that a job makes during its execution. In the Grid environment there are external mechanisms on nodes that can introduce jobs and such jobs are not managed by RMS, but by node OS. In this paper, we assume that the RMS is able to take this into account either by detecting external resource utilization or interfacing with the node's native operating system.

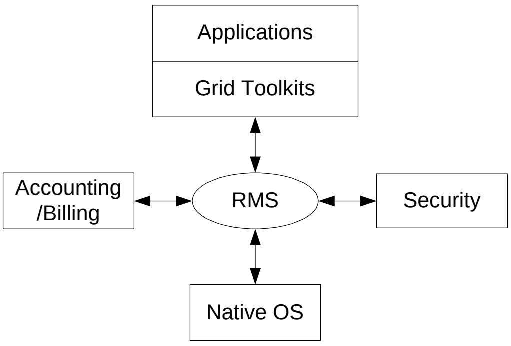

Figure 2: Resource Management System Context.

We defined aabstract model of resource management systems that provides a basis for a comparison between different RMS architectures. The model presented in Figure 3 shows the functional units and data stores that a general purpose RMS for a Grid would have. Existing resource management systems implement some of these functions but do so in very different ways. The abstract model provides a mechanism to compare and contrast the different RMS implementations.

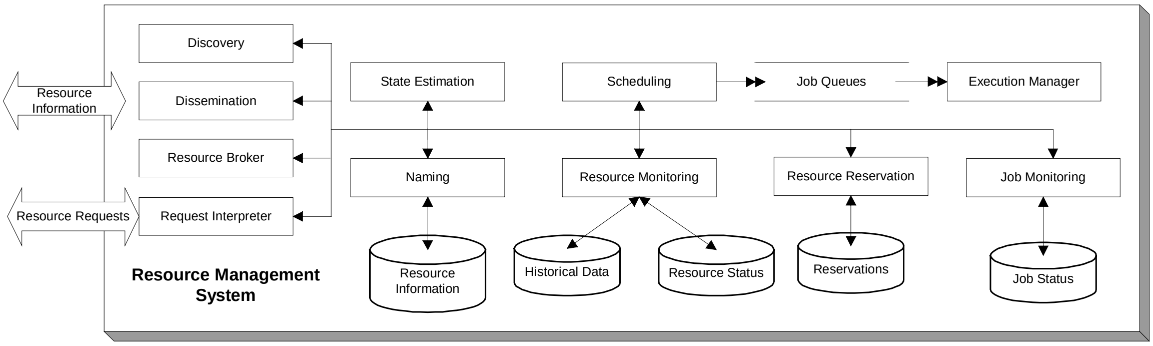

Figure 3: Resource Management System Abstract Structure.

The model describes the RMS structure as a whole without specifying which machines provide the functions. In an actual RMS, functions may run on dedicated or special purpose machines within the Grid. For example, a RMS may have a partially centralized scheduling mechanism that schedules resources for all machines in the Grid that runs on a few designated machines whereas another RMS performs scheduling on all machines concurrently. Some resource management systems do not implement some of the data stores or functional units.

There are three different types of functional units, application to RMS interfaces, RMS to native operating system or hardware environment, and internal RMS functions. The application to RMS interfaces provides services that end-user or Grid applications use to carry out their work. The RMS to native operating system or hardware environment interface provides the mechanisms that the RMS uses to implement resource management functions. The internal RMS functions are implemented as part of providing the resource management service. The request interpreter function provides the application to RMS interfaces. The resource dissemination, resource discovery, and resource broker functions provide internal RMS functions but may be directly accessible to applications albeit through a more restrictive interface than the RMS itself. It is up to the implementation to decide whether there are two different interfaces provided to these functions, one for the applications and the other for the RMS itself. For the purposes of the abstract model the request interpreter will function as the mediator between the application and the other RMS components. The RMS to native operating system interfaces are provided by the execution manager, job monitoring, and resource monitoring functions. Internal RMS functions are provided by the resource naming, scheduling, resource reservation and state estimation.

Resource information is distributed between machines in the Grid using a resource information protocol. This protocol is implemented by the resource and dissemination functions. In some RMS architectures that use market based scheduling approaches, a specialized resource broker functions exists that implements a trading protocol [9][10]. Other RMS architectures could also potentially use a specialized resource broker that provides alternative application oriented resource presentations for the discovery and dissemination functions.

Application resource requests are described using a resource description language or protocol that is parsed by the resource interpreter into the internal formats used by the other RMS functions. Existing systems provide different interfaces for resource reservation but in our abstract model, this is considered an internal function of the RMS since conceptually immediate resource requests and reservation resource requests are not different. The request interpreter passes reservation requests on to the reservation function that maintains the database of resource reservations. The resource request protocol is bi-directional since applications should be able to effectively negotiate resources and associated service levels and thus requires feedback on requests from the RMS.

The resource dissemination and resource discovery functions provide the means by which machines within the Grid are able to form a view of the available resources and their state. In some RMS architectures there is no distinction made between these functions since they are implemented using a single protocol or function. There is, however, a significant difference between dissemination of resource information and the discovery of resource information. Resource dissemination provides information about machine resources or a pointer to a source of information resources. Resource discovery is the mechanism by which applications find resource information.

For example, an RMS could use a replicated network directory that contains resource information. The resource dissemination function 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 network address and resource discovery consists of querying the central directory. The two examples could both be implemented using a network directory such as LDAP but would result in quite different RMS architectures.

The resource naming function is an internal function that enforces the namespace rules for the resources and maintains a database of resource information. The structure, content, and maintenance of the resource database are important differentiating factors between different RMS. The implementation of the resource database is a separate concern from the implementation of the naming function and is less important. The naming function interacts with the resource dissemination, discovery, and request interpreter so design choices in the namespace significantly affect the design and implementation of these other functions. For example, a flat namespace or a hierarchical namespace could both be implemented using a network directory for resource description databases. A flat namespace would impose a significantly higher level of messaging between machines in the Grid even with extensive caching.

As the request interpreter accepts requests for resources, they are turned into jobs that are scheduled and executed by the internal functions in the RMS. The job queue abstracts the implementation choices made for scheduling algorithms. The scheduling function examines the jobs queue and decides the state of the jobs in the queue. The scheduling function uses the current information provided by the job status, resource status, and state estimation function to make its scheduling decisions. The state estimation uses the current state information and a historical database to provide information to the scheduling algorithm. The structure of the job queue is dependent on the scheduling algorithm and varies from a simple FIFO queue to multiple priority ordered queues. The execution manager function is responsible for controlling the execution of the jobs on the machine. In some systems, the execution manager does not control the execution of the jobs on a machine other than initiating the job using the native operating system services. The job monitoring and resource monitoring functions update their respective databases.

5. Resource Management System Taxonomy

The taxonomy classifies resource management systems 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 we classify a RMS 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 the rescheduling.

In the following sections we will use the terms *resource requestor* to identify the machine that is requesting a resource, *resource provider* to identify the machine that is providing the resource, and *resource controller* to identify the machine that is responsible for allocating the resource. Current research Grid systems have machines that function in one or more of these roles.

Previous taxonomies focused on distributed and centralized variants of the different attributes of heterogeneous network computing systems. Grids are highly scaleable wide-area computing systems so current research implementations use some variant of distributed mechanisms. Thus we omit classifications of centralized mechanisms since these are not an area of interest.

5.1. 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 we need to distinguish how the resource requestors, resource providers, and the resource controllers organize their dialogue.

For example, a resource requestor 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 requestor 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 we use a different approach and characterize the organization structure and resource management implementation separately. This section describes the organization and the next section describes the resource management implementation.

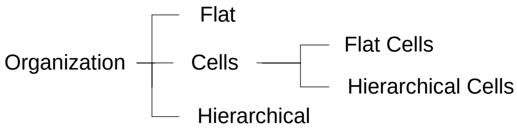

Figure 4: Organization Taxonomy.

In a flat organization all machines can directly communicate with each other without going through an intermediary. In a hierarchal organization machines in 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 a flat or hierarchical 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.

5.2. 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 management components are even more important than the scheduling components since 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.

5.2.1 Resource Model

The resource model determines how applications and the RMS describe and manage Grid resources. Figure 5 shows the resource model taxonomy. The taxonomy focuses on how the resources and operations on the resources are described. It is important to determine if the resource descriptions and resource status data store are integrated with their operations in an active scheme or if they function as passive data with operations being defined by other components in the RMS.

In a schema based approach the data that comprises a resource is described in a description language along with some integrity constraints. In some systems the schema language is integrated with a query language. The schema languages are further characterized by the ability to extend the schemas. In a fixed schema all elements of resource description are defined and cannot be extended. In an extensible scheme new schema types for resource description 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 scheme 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. It may be difficult to implement a high performance extensible object model scheme. In current fixed object model schemes the object models are typically very basic and provide very few primitive operations on resources.

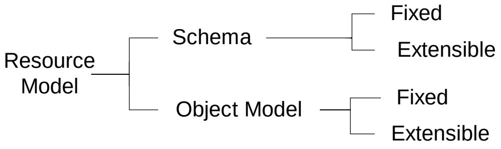

Figure 5: Resource Model Taxonomy.

5.2.2 Resource Namespace Organization

The RMS for a Grid system provides a global namespace for resources required by network applications. The organization of the resource namespace influences the design of the resource management protocols and affects the discovery methods. For example in a flat namespace the use of agents to discover resources would require some sort of global strategy to partition the search space in order to reduce redundant searching of the same information. 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 (LAN segments) or logical (DNS domains) network structure of the Grid. 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.

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. 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.

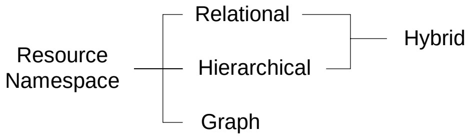

Figure 6: Namespace Organization Taxonomy.

5.2.3 Quality of Service Support

The level of support for extensive QoS for all components of the Grid will become more important as network applications becoming more sophisticated and demanding. Our notion of QoS is not limited to network bandwidth but extends to the processing and storage capabilities of the nodes in the Grid. Thus we focus on the degree that a Grid can provide end-to-end QoS across all components rather than QoS only on the network as is typically done. 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 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.

Another aspect of QoS is the ability for a Grid application to negotiate a service level agreement (SLA) with the RMS. In some cases an application can accept lower overall performance or an inexact match between a specified resource and the actual resource supplied. This idea of general QoS or SLA can be extended to placing bounds on resource discovery or other grid provided services such as data migration. For example, if an application requires a resource the RMS could return similar resources within a local administrative domain. If a particular resource is required the discovery process could be extended outside of the local administrative domain. An underlying assumption of this scenario is that there is a cost differential applied when crossing administrative domains when doing discovery. This may be particularly true of agent based resource discovery.

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 does not allow applications to specify QoS requirements in resource requests does not support QoS. It can be argued that scheduling is a form of admission control but if the resource request interface provides no way to specify required resource service levels it does not support QoS. A RMS that provides explicit QoS attributes for resource requests but cannot enforce service levels via policing provides soft QoS support. Most current Grid systems provide soft QoS since most non real-time operating systems do not allow the specification of service levels for running jobs and thus cannot enforce non-network QoS guarantees. Hard QoS support is provided when all nodes in the Grid can police the service levels guaranteed by the RMS.

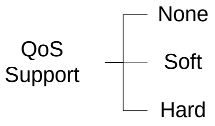

Figure 7: QoS Support Taxonomy.

5.2.4 Resource Information Store Organization

The resource description, resource status, and resource reservation data store organization help characterize the overall performance of the RMS. The organization determines the cost of implementing the resource management protocols since resource dissemination and discovery may be provided by the data store implementation. Figure 8 shows the taxonomy for 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. Network directories data stores are based on X.500/LDAP standards or utilize specialized distributed database implementation. 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.

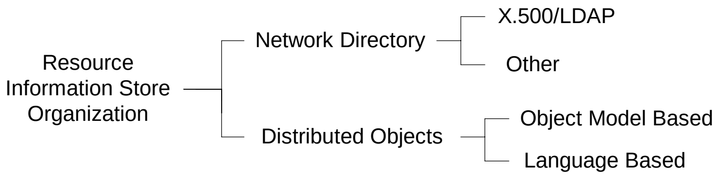

Figure 8: Resource Information Store Taxonomy.

5.2.5 Resource Discovery and Dissemination

A major function of a RMS in a Grid is to provide a mechanism for resources in the Grid to be discovered and utilized by network application. Resource discovery and dissemination provide 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. The overhead of matching resources and applications determines the efficiency of the RMS and determines the maximum resource utilization that a RMS can achieve in the Grid environment. 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, which uses its query engine to execute the query against the database contents. Query based system 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 approach. Agent based approaches send active code fragments across machines in the Grid that are interpreted locally on each machine. 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. Most agent systems are based on an underlying mobile code environment like Java. Agent based resource discovery is inherently distributed.

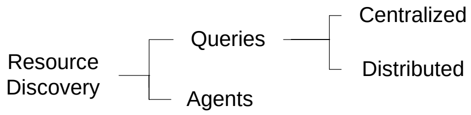

Figure 9: Resource Discovery Taxonomy.

Resource dissemination is categorized by the approach taken to updating the resource information. The mechanisms used to implement dissemination determine the amount of data that is sent between machines in the Grid. The latency of resource information across machines in the Grid is also determined by the dissemination mechanism. Note that it is possible to have more than one dissemination mechanism. 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.

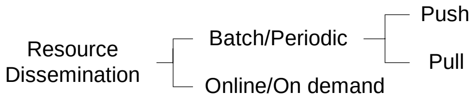

Figure 10: Resource Dissemination Taxonomy.

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. 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.

5.2.6 Valid Combinations

The resource model used by a RMS determines many of the other classifications in the taxonomy. The choice between a schema based model and an object-oriented model for resources is very fundamental. Schema based models use either relational or hierarchical namespaces for the resources. Large scale Grid RMS that use a schema model will use a hybrid namespace where the relations are grouped hierarchically allowing efficient distribution and replication of resource information. Object oriented models use a graph-based namespace.

The choice of schema model typically results in using a network directory information store organization since this is highly efficient mapping of the logical content of the resource information store to the physical implementation. It is possible to use a distributed object store organization for a schema but this would be less inefficient than using network directories which have been optimized for storing schemas in a distributed network environment. Conversely object models would use a distributed object store organization since the efficient mapping of an object to a network directory typically requires translating the object representation to the network directory supported schema. This has been difficult to implement in an effective and efficient manner.

The current state of the art in operating systems and networking equipment makes the support of QoS difficult. Many network devices do not support QoS and those that do support only a few differentiated service classes. The concept of QoS is not directly supported in operating systems other than dedicated real time systems. The other complication is that the Grid RMS operates at the application level rather than at the operating system level. Existing mechanisms such as application level scheduling priorities do not allow the direct specification of QoS such as allocating 80% of the processor cycles to an application. Other issues arise with the attribution of native operating system level activity to individual jobs. Thus soft QoS is the best that a Grid RMS can offer today unless it runs on a specially engineered network and node platform, in which case hard QoS can be offered to jobs.

Resource discovery is either a distributed query or agent based. Centralized queries will not work in a large scale distributed system such as the Grid. Future Grid RMS may provide a hybrid environment where distributed queries give partial results about resource information with full resource information being obtaining by using agents. There are no current Grid systems that we are aware of that use this approach.

Resource dissemination needs to be scalable and efficient. There are a number of different data stores in the RMS that contain information that needs to be distributed across the Grid. It is possible to use different schemes to distribute the information based on the frequency with which the data in the store changes. For example, the resource and job status information may change frequently which could result in a periodic push of information through the Grid. The addition of new resources to the Grid is infrequent and could be performed either on demand or periodically pulled depending on the implementation of the relevant data store. For example, network directories typically have a replication protocol to synchronize content.

Based on this discussion we end up with the following valid combinations for the resource model, namespace organization, and the resource information store organization; *schema model with a hybrid namespace using a network directory* or an *object model with graph namespace using distributed objects*. All QoS schemes are valid. Resource discovery schemes are either *distributed queries* or *agents.* All schemes are valid for resource dissemination with the potential to have more than one scheme used concurrently in a RMS for different types of information.

5.3. Scheduling

This section contains the taxonomies that describe how the scheduling components of the RMS are organized, the degree of extensibility, rescheduling approaches, and state estimation. We do not address the scheduling algorithms since many of the current scheduling approaches for networking and parallel computing jobs are also applicable to a Grid RMS. The scheduling components are internal to the RMS but determine the external Grid performance as determined by applications and also the measured utilization of the resources provided by the Grid.

The scheduling components of the Grid decide which resources are assigned to which jobs. The assigning of resources to jobs is a policy decision. The policy can be explicitly specified via an external rule base or a programmable interface. The policy can be implicitly implemented by choice of state estimation algorithms and rescheduling approach as well. Both approaches can also provide a way for external programs to override the scheduling policy. The scheduling taxonomy presented here captures the scheduling features of a Grid but the actual combination of the features determine how resources are assigned to jobs.

5.3.1 Scheduler Organization

Previous taxonomies focused on the degree of centralization in the scheduler. Current Grid systems use decentralized scheduling schemes where more than one machine in the Grid can perform scheduling and they use a distributed scheme for resource information dissemination.

In our approach we focus upon the resource controller function, the mapping of resource controller to machine and the number of logical request and job queues used by the RMS at the Grid level. The resource controller is responsible for allocation and scheduling of the resource providers. A particular machine in the Grid may support one or more resource providers, a resource controller and a resource requestor. The resource controllers interact with each other to match providers to a request and then schedule the resources provided by the resource providers. Once a schedule has been determined the resource providers and the resource requestors are informed of the schedule. Figure 11 shows the scheduler organization taxonomy.

In a centralized controller scheme there are one or more machines that provide the resource controller function for resource requestors and resource providers. All requests are routed to the resource controller machines that schedule the resource provider machines. Logically there is one central request and job queue that is distributed among the designated resource controllers in the Grid. In a hierarchical controller approach the resource controllers are organized as a hierarchy. Resource controllers in higher levels of the hierarchy schedule bigger units of resource providers with lower level resource controllers scheduling smaller units of resource providers until individual resource providers are scheduled. Logically there is one request and job queue per hierarchy level. A one level hierarchical controller organization and a centralized controller organization can be considered to be equivalent. In a fully decentralized approach there are no dedicated resource controller machines, the resource requestors and resource providers directly determine the allocation and scheduling of resources. Logically there are as many different request and job queues as there are resource requestors and resource providers in the Grid.

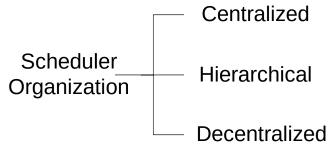

Figure 11: Scheduler Organization Taxonomy.

5.3.2 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 [10]. 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.

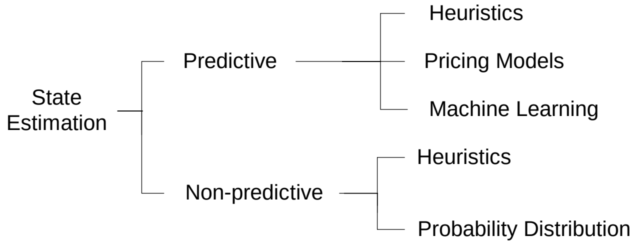

Figure 12: State Estimation Taxonomy.

5.3.3 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. Periodic or batch rescheduling approaches group resource requests and system events and process them 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.

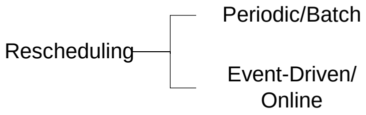

Figure 13: Rescheduling Taxonomy.

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.

5.3.4 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. In a large Grid system with many different administrative domains we can expect different resource utilization policies. Thus it is unlikely that an unalterable scheduling policy will suffice for the different needs.

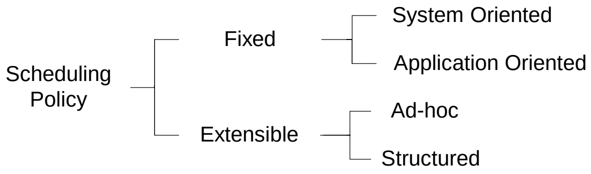

Figure 14: Scheduling Policy Taxonomy.

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. 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.

5.3.5 Valid Combinations

Unlike the taxonomies for the resources for a RMS, the scheduling taxonomies are orthogonal to each other. Thus different combination of scheduler organization, state estimation, rescheduling, and scheduling policy classifications can be implemented in a Grid RMS. In current Grid systems it is possible to change portions of the scheduling subsystem without affecting the other components of the RMS or the other scheduling components. For example, the Nimrod/G super-scheduler was used in conjunction with the existing Globus scheduling and resource components.

6. Grid Resource Management Systems Survey

There are many existing Grid and network computing projects currently underway with many new projects starting all the time. The example systems survey is no exhaustive, but comprehensive enough to have case systems so that most classifications in the Grid RMS taxonomy are covered (Figure 1). We are trying to capture as many different classifications of the resource taxonomies and scheduling taxonomies as possible. In the following survey we will give a brief description of each system and then classify the resource management system attributes according to our taxonomy. We will focus our attention on the RMS aspect of the surveyed system rather than the overall Grid level characteristics of a system. A summary of architectural design choices made by a few popular Grid resource management systems is shown in Table 1. Attributes that are not described in a section for a system are not part of the surveyed system or provided by some other component such as the underlying Grid fabric or operating systems. The goal of this section is to demonstrate the use of the taxonomy to classify a large number of examples from the literature particularly recent works in grid resource management.

Table 1: Grid resource management systems and their architecture choices.

System	Grid Type	Organization	Resources	Scheduling
2K	On Demand Service Grid	Flat (Hierarchical in the Future)	Extensible object model, graph namespace, soft network QoS, object model store, agent based discovery, online/demand dissemination	One level hierarchical scheduler for network resources, decentralized scheduler for other resources
AppLeS	Computational Grid (scheduling)		Uses resource model provided by the underlying Globus, Legion, or Netsolve middleware services	Centralized scheduler, predictive heuristic state estimation, online rescheduling, fixed application oriented policy
Bond	On Demand Service Grid	Flat	Extensible object model, graph namespace, hard QoS, language based object store, agent based discovery, periodic push dissemination	Decentralized scheduler, predictive pricing models, online rescheduling, fixed application oriented policy
CERN DataGrid	Data Grid Computational Grid	Hierarchical	Extensible schema model, hierarchical namespace, no QoS, LDAP network directory store, distributed query-based discovery, periodic push dissemination.	Hierarchical schedulers, extensible scheduling policy
Condor	Computational Grid	Flat	Extensible schema model, hybrid namespace, no QoS, other network directory store, centralized query based discovery, periodic push dissemination	Centralized scheduler
Darwin	Network Oriented Service Grid	Hierarchical	Fixed schema model, graph namespace, hard QoS	Hierarchical scheduler, non-predictive state estimation, online rescheduling, fixed system oriented policy.
Globus	Grid Toolkit (for developing computational, data, & service Grids)	Hierarchical Cells	Extensible schema model, hierarchical namespace, soft QoS, LDAP network directory store, distributed query based discovery, periodic push dissemination	It offers lower level services for resource allocation or co-allocation including resource reservation. Higher-level tools (like Nimrod/G) perform scheduling.

Javelin	Computational Grid	Hierarchical	Fixed object model, graph namespace, soft QoS, other network directory store, distributed query based discovery, periodic push dissemination	Decentralized scheduler, fixed application oriented policy
Lancaster **DMRG**	Multimedia Service Grid	Flat	Extensible object model, graph namespace, hard QoS	Decentralized scheduler, ad-hoc extensible policy
Legion	Computational Grid	Flat Hierarchical	Extensible object model, graph namespace, soft QoS, object model store, distributed query based discovery, periodic pull dissemination.	Hierarchical scheduler, ad-hoc extensible scheduling policies
MOL	Computational Grid	Hierarchical Cells	Extensible schema model, hierarchical namespace, no QoS, object model store, distributed query based discovery, periodic push dissemination	Decentralized scheduler, extensible ad-hoc scheduling policies
MSHN	Computational & Service Grids	Flat	Fixed schema model, hybrid namespace, hard QoS, other network directory store, distributed query based discovery, online dissemination	Centralized scheduler, predictive heuristics for state estimation, event driven rescheduling, fixed system oriented scheduling policy
NetSolve	Computational & Service Grids	Hierarchical	Extensible schema model, hierarchical namespace, soft QoS, centralized query based discovery, periodic push dissemination.	NetSolve Agents perform scheduling.
Nimrod/G & GRACE	High-throughput Computational and Service Grids.	Hierarchical Cells	Uses resource model provided by the underlying Globus or Legion middleware services. It offers both soft and hard QoS depending on their availability on computational nodes.	Application-level scheduling policies driven by computational economy and deadline. It follows hierarchically distributed scheduling model.
Ninf	Computational & Service Grid	Hierarchical	Extensible schema model, relational namespace, no QoS, centralized query based resource discovery, periodic push for dissemination.	Decentralized scheduler,
Ninja	On Demand Service Grid	Hierarchical	Extensible object model, graph namespace, no QoS, language based store, distributed query based discovery, periodic push dissemination	Not applicable
PUNCH	Computational & on Demand Service Grids	Hierarchical	Extensible schema model, hybrid namespace, soft QoS, distributed query based discovery, periodic push dissemination.	Both Hierarchical and decentralized approach in scheduler organization. PUNCH employs non-preemptive, decentralized, adaptive, sender-initiated scheduling.

6.1. 2K: A Distributed Operating System

2K is a distributed operating system [40], [42] that provides a flexible and adaptable architecture for providing distributed services across a wide variety of platforms ranging from a Palm PDA to large scale Unix and Microsoft Windows platforms. The emphasis in the 2K project is on providing a flexible and extensible operating system environment for the development and deployment of distributed service applications. High performance grand challenge applications do not seem to be a target of the 2K project.

The core of the 2K system is a dynamic reflective CORBA object request broker (ORB) called *dynamicTAO* that is an extension of the *TAO* ORB [43]. 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 machine organization. In [41] an extension to the current flat model to hierarchical model is described using CORBA traders and Name servers. 2K 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 (DSRT) Dynamic Soft Real Time scheduling at each resource provider resulting. 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. They do not appear to support any scheduling policy component in the 2K system.

6.2. AppLeS: A Network Enabled Scheduler

The AppLeS [1] (Application Level Scheduling) project at the University of California, San Diego primarily focuses on developing scheduling agents for individual applications on production computational Grids. It uses the services of Network Weather Service (NWS) to monitor changes in performance of resources dynamically. AppLeS agents use static and dynamic application and system information while selecting viable set of resources and resource configurations. It interacts with other resource management systems such as Globus, Legion, and NetSolve to implement application tasks. The applications have embedded AppLeS agents and thus become self-schedulable on the Grid. The concept of AppLeS has been applied to many application areas including Magnetohydrodynamics [2], Gene Sequence Comparison, satellite radar images visualization, and Tomography [3].

Another effort within AppLeS project framework is the development of AppLeS templates (similar to Nimrod/G framework and resource broker, but not budget and deadline scheduling). These templates can be applied to a number of applications, but they need to be structurally similar and have the same computational model. Templates have been developed for application classes such as master/slave and parameter studies.

As the focus of AppLeS project is on scheduling, 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 (like in Nimrod/G). AppLeS schedulers do not offer QoS support and build on whatever resource model offered by the underlying systems. AppLeS can be considered to have predictive heuristic state estimation model with online rescheduling and application oriented scheduling policies.

6.3. Bond: Java Distributed Agents

Bond is a Java based object oriented middleware system for network computing [44]. Bond is based on agents [45] that communicate using KQML, the Knowledge Querying and Manipulation Language, for inter object communication. KQML is implemented using Java and 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 [46]. 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 [47] supports hard QoS and provides a general-purpose real time OS platform for multiresource reservation, scheduling and signaling. Tempo is the bandwidth manager component and schedules network bandwidth. It is implemented as an extension of Sun Solaris.

Bond is middleware that provides the infrastructure for an on-demand service Grid with a flat organization since there is no concept of autonomous domains with a border. Other organizations can implemented on top of Bond but require extended the resource management functions provided by Bond. The resource model is 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.

6.4. CERN Data Grid

CERN, the European Organization for Nuclear Research, and the High-Energy Physics (HEP) community have established a project called ‘‘Research and Technological Development for an International Data Grid” [13]. The objectives of this project are the following. Firstly, establish a Research Network that enables the development of the technology components essential for the implementation of a new worldwide Data Grid on a scale not previously attempted. Secondly, demonstrate the effectiveness of this new technology through the large-scale deployment of end-to-end application experiments involving real users. Finally, demonstrate the ability to build, connect and effectively manage large general-purpose, data intensive computer clusters constructed from low-cost commodity components. Furthermore, the project does not only cover HEP but also other scientific communities like Earth Observation and Bioinfomatics.

The CERN Data Grid project focuses on the development of middleware services in order to enable a 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 CERN Data Grid project has a multi-tier hierarchical machine organization. For example, tier-0 is CERN, which stores almost all relevant data, several tier-1 regional centers (in Italy, France, UK, USA, 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.

6.5. Condor: Cycle Stealing Technology for High Throughput Computing

Condor [14], [16] is a high-throughput computing environment developed at the University of Wisconsin at Madison, USA. It can manage a large collection of computers such as PCs, workstations, and clusters that are owned by different individuals. Although it is popularly known for harnessing idle computers CPU cycles (cycle stealing), it can be configured to share resources. The Condor environment follows a layered architecture and offers powerful and flexible resource management services for sequential and parallel applications. The Condor system pays special attention to the computer owner's rights and allocates their resources to the Condor pool as per the usage conditions defined by resource owners. Through its unique 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 machine 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. The agents are then notified of their compatibility. The compatible agents then contact each other directly and if they are satisfied, then the customer agent initiates computation on the resource.

Resource requests and offers are described in the Condor classified advertisement (ClassAd) language [15]. ClassAds use a semi-structured data model for resource description. Thus no specific schema is required by the matchmaker allowing it to work naturally in a heterogeneous environment. 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.

The matchmaker performs scheduling in a Condor pool. The matchmaker is responsible for initiating contact between compatible agents. Customer agents may advertise resource requests to multiple pools with a mechanism called flocking, allowing a computation to utilize resources distributed across different Condor pools.

The Condor system has recently been enhanced to support creation of a personal condor pools. It allows the user to include their Globus-enabled nodes into the Condor pool to create a "personal condor" pool along with public condor pool nodes. The Grid nodes that are included in personal condor pool are only accessible to the user who created the pool.

Condor can be considered as 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.

6.6. Darwin: Resource Management for Network Services

Darwin is a customizable resource management system [24] 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 required. 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. Control delegates are Java code segments that reside on the routers. 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. Local resource managers coordinate the allocation of network resources using the Beagle signaling protocol [30]. Darwin uses a hierarchical fair service curve scheduling algorithm (H-FSC) for higher level resource allocation. The H-FSC algorithm was designed to efficiently support virtual networks for distributed applications.

Darwin is a networked oriented on demand service Grid. The machine organization is a one level hierarchy since all requests are sent to a Xena request broker, which interacts with its peer request brokers in a Darwin system. The resource model is a fixed schema with hard QoS support. The resource namespace is a graph. Darwin does not provide a separate resource information store, resource discovery protocol, or resource dissemination protocol. Darwin is network oriented and much of the network resource information is available from the underlying network routers. 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.

6.7. Globus: A Toolkit for Grid Computing

Globus [5] provides software infrastructure that enables applications to view distributed heterogeneous computing resources as a single virtual machine. The Globus project is an American multi-institutional research effort that seeks to enable the construction of computational Grids. Currently the Globus researchers are working together with the High-Energy Physics and the Climate Modeling community to build a Data Grid [52]. A central element of the Globus system is the Globus Metacomputing Toolkit (GMT), which defines the basic services and capabilities required for constructing computational Grids. 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. The GMT provides a bag of services from which developers of specific tools or applications can select from to meet their own particular needs.

Globus is constructed as a layered architecture in which higher level services can be developed using the lower level core services [6]. Its emphasis is on the hierarchical integration of Grid components and their services. This feature encourages the usage of one or more lower level services in developing higher level services. Globus offers Grid information services via an LDAP-based network directory called Metacomputing Directory Services (MDS) [50].

The recent MDS (resource info store) is distributed in nature and it consists of two components: GIIS (Grid Index Information Service) and GRIS (Grid Resource Information Service). The GRIS is responsible for providing a uniform means of querying resources on a Grid for their current configuration, capabilities, and status. The directory information is provided by Globus system (running each resource) or other information providers or tools. The GIIS provides a means of knitting together arbitrary GRIS services to provide a coherent system image that can be explored or searched by Grid applications. It could list all of the resources available within a confederation of laboratories, or all of the distributed data storage systems owned by a particular agency. It can also pool information about all of the Grid resources (computation, data, networks, and instruments) in a particular research consortium, thus providing a coherent system image of that consortium’s computational Grid. The resource information providers’ use push protocol to update GRIS. The information maintained in GIIS is updated using pull protocol (i.e., GISS pulls information from multiple GRIS).

Thus MDS follows both push and pull protocol for resource dissemination. Higher-level tools such as resource brokers can perform resource discovery by querying MDS using LDAP protocols. The MDS namespace is organized hierarchically in the form of a tree structure. Globus offers QoS in the form of resource reservation [51]. It allows application level schedulers such as the Nimrod/G resource broker to extend scheduling capabilities. The resource brokers can use heuristics for state estimation while performing scheduling or re-scheduling whenever the status of the Grid changes. Globus is a Grid toolkit and thus does not supply scheduling policies, instead it allows third party resource brokers. Globus services have been used in developing many resource brokers (global schedulers) including, Nimrod/G, AppLeS, and Condor/G.

6.8. Javelin

Javelin [12] is a Java based infrastructure for internet-wide parallel computing. The three key components of Javelin system are the clients or applications, hosts, and brokers. A client is a process seeking computing resources, a host is a process offering computing resources, and a broker is a process that coordinates the allocation of computing resources. Javelin supports piecework and branch and bound models of computation. 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-worker) applications such as ray tracing and Monte Carlo simulations. The latest Javelin system, Javelin 2.0, supports branch-and-bound computations. It 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 machine organization where each broker manages a tree of hosts. Resources are simple fixed objects with graph (tree) 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 JavelinBNS system, a Javelin information backbone that maintains list of available brokers. The host then communicates with brokers and chooses suitable broker and then becomes part of the broker-managed resources. Thus the information store is a network directory implemented by JavelinBNS. Hosts and brokers update each other as a result of scheduling work thus Javelin uses demand resource dissemination. The broker manages the host-tree or resource information through a heap-like data structure. 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 using work stealing and fixed application oriented scheduling policy. Whenever a host completes an assigned job, it requests works from peers and thus load balancing is achieved.

6.9. Lancaster Distributed Multimedia Research Group

The University of Lancaster Distributed Multimedia Research Group (DMRG) has a number of projects focused on providing a high performance real time platform for distributed multimedia applications. The Generic Object Platform Infrastructure (GOPI) project developed a platform based on CORBA with RM-ODP extension that provides an extensible architecture for adaptive multimedia applications [35][36]. 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 Lancaster DRMG systems together form what can be considered a multimedia service Grid. 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 as in the case in the SUMO project. The research has been focused on delivering multimedia application support and thus lacks infrastructure for resource information directory, resource discovery, and resource dissemination protocols. A scheduler framework is provided into which application specific schedulers are loaded. The scheduler framework and scheduling extensions operate on a per nodes basis thus the scheduler organized 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.

6.10. Legion: A Grid Operating System

Legion [4] is an object-based metasystem or Grid operating system developed at the University of Virginia. Legion provides the software infrastructure so that a system of heterogeneous, geographically distributed, high performance machines can seamlessly interact. Legion provides application users with a single, coherent, virtual machine. The Legion system is organized into classes and metaclasses.

Legion objects represent all components of the Grid. Legion objects are defined and managed by their class object or metaclass. Class objects 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. Objects can be deactivated and saved to persistent storage. Objects are reactivated automatically when another object wants to communicate with it. Legion defines an API for object interaction, but 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. Currently, there is a global collection named "/etc/Collection" for the system that tracks HostObjects and VaultObjects which embody the notion of persistent storage. The users or their agents can obtain information about resources by issuing queries to a Collection.

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 resource management architecture is hierarchical with decentralized scheduling policies. Legion supplies default system oriented scheduling policies, but it allows policy extensibility through resource brokers. That is, application level schedulers such as Nimrod/G [10] and AppLeS [2] can change Legion default scheduling policies to user-oriented policies such as computational economy and deadline-based scheduling.

6.11. MOL: Metacomputing Online Kernel

MOL initiative is developing technologies that aim at utilizing multiple WAN-connected high performance systems as a computational resource for solving large-scale problems that are intractable on a single supercomputer. One of the key components of MOL toolbox is the MOL-Kernel [48]. It offers basic generic infrastructure and core services for robust resource management that can be used to construct higher level services (tools and applications). MOL-Kernel services include, managing available resources of institutions (computing) centers, establishing a dynamic infrastructure for interconnecting these institutions, managing faults (network connection failures), and providing access points for users.

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 handlers. The resource modules are meant to encapsulate various metacomputing resources such as hardware (e.g., computing resources and scientific devices) and services (e.g., applications and databases). All resource modules in an institution (center) are coordinated by CMM. This module is responsible for keeping its network components/resources in a consistent state and also makes them accessible from the outside world. It essentially acts as a gatekeeper and controls the flow of data between center resources and external networks. Usually there is one CMM per institution, but it is possible to have multiple (identical) CMMs in the case of large organizations. If any of the MOL-Kernel components fails, only one institute becomes inaccessible in the worst case. Hence, there is no single-point-of-failure exist in the system. As long as a single CMM available, the MOL-kernel remains operational. That means, organizations can leave or enter the metacomputing environment as they wish. The MOL-Kernel dynamically reconfigures itself to include or exclude corresponding center (organization) resources. A significant challenge in this distributed architecture is how to guarantee that the collective CMMs always maintain consistent state. In MOL-kernel, this is achieved by using a transaction-oriented protocol on top of virtual shared memory objects associated with each CMM. In order to make the global state available at all entry points, mirror instances of shared active objects are maintained at each CMM. Whenever the state of a shared object changes, the new information is automatically distributed to the corresponding mirror instances.

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 access module, 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. There exist two types of event handlers: predefined and customizable. Some of the predefined handlers offer basic services like routing messages to other destinations, broadcasts, collecting information from a subset of available services. High level handlers can offer services like "response to load status queries" and invocation of preinstalled applications. Such high level services (are not part of kernel) and they need to be implemented as customized event handlers.

The MOL follows a service Grid model with hierarchical cell-based machine organization. It adopts the schema based resource model and hierarchical name space organization. The global state is maintained in shared objects of each CMM (i.e., object based resource information storage). The resources and services themselves announce their initial presence to MOL (push protocol in information dissemination). The access modules/schedulers perform resource discovery and scheduling by querying shared objects. Although the resource model is schema based, its primary mode is service based. For example, if users request an application (e.g. CFD-simulation) with a certain quality of service. MOL then finds those computers, which have this application, installed and asks them "which of you are powerful enough to provide the requested quality of service?" (i.e., decentralized scheduler). It then selects one or more to execute the request.

6.12. MSHN: Management System for Heterogeneous Networks

The MSHN project [37] is a collaborative effort between the United States Department of Defense (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). It is a research effort aimed at developing a resource management system for distributed heterogeneous environments. The resource manager is designed to run in a true Grid environment where each machine has its own native operating system. One unique aspect of MSHN is that it is targeted to support applications that can adapt to resource conditions in the Grid [39]. Different versions of an application may be available to run on different machine architectures or under different operating systems. In addition some applications may be able to change their resource utilization at runtime in order to adapt to changes in the Grid resource availability. MSHN supports a general concept of QoS in the scheduler design [38] but has no explicit QoS specification language. Advance reservation of resources is also supported. Applications are described to MSHN by a directed acyclic graph where the nodes are subtasks that are annotated with QoS requirements. The edges denote subtask precedence.

The MSHN architecture consists of a client library module, a resource status server, resource requirements database, and a scheduling advisor. The MSHN application resource interface approach is similar to Condor. The client library is linked into applications and the resource requests trapped and routed to the MSHN system. For example, attempting to start an application on a remote machine causes the request to transfer to MSHN for scheduling. Each machine in MSHN runs *mshd*, a daemon that provides the interfaces to the other parts of the system. The scheduling advisor provides centralized scheduling services for the MSHN system. The resource status server keeps track of the resource status and compares the scheduled resource utilization against the actual resource utilization. A rescheduling event is generated and sent to the schedule advisor if there is a significant discrepancy. The resource requirement server contains fine grained resource information about applications resource requirements. The goal is that compilers and application developers can populate the resource information database.

MSHN can be considered as a computational and service Grid with a flat machine organization. The resource model will be classified as fixed schema since it will be based on network directories. The resource namespace for similar reasons will be a hybrid namespace. MSHN will also support hard QoS. The information store will use an X.500/LDAP network directory. Resource discovery is achieved through distributed queries and resource dissemination performed online. The scheduler organization is centralized with predictive heuristics for state estimation. MSHN is investigating a number of scheduling approaches. Currently it can be classified as event driven rescheduling with a fixed system oriented scheduling policy.

6.13. NetSolve: A Network Enabled Computational Kernel

Netsolve [8] 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 software into the desktop application. It performs resource discovery and delivers solutions to the user in a directly usable and “conventional” manner (i.e., no need to develop special program code like parallel code to use high-end machines). The backend networks resources that includes supercomputers and software (i.e., it can be parallel high performance computational kernel/special purpose software library). The use of such techniques will help in hiding parallel processing complexity from the user applications and deliver the power of parallel processing to desktop users with ease.

Netsolve clients (applications) can be written in C and Fortran and use Matlab or the Web to interact with the server. A Netsolve server can use any scientific package to provide its computational software. Communications within Netsolve is performed using TCP/IP socket facility. Good performance is ensured by a load-balancing policy that enables Netsolve to use the computational resources available as efficiently as possible. Netsolve offers the ability to search for computational resources on a network, choose the best one available, solve a problem (with retry for fault-tolerance), and return the answer to the user.

The Netsolve system follows the service Grid model with hierarchical cell-based machine organization. The Netsolve-agents act as an information repository and maintains the record of resources (hardware and software) available in the network. The resources themselves are responsible for making their existence aware to Netsolve Agent (push protocol in information dissemination). That is, whenever a new server comes-up, it sends information such as its location and services that served can offer to the Agent. The Netsolve Agent also acts as a resource broker and performs resource discovery and scheduling. The user requests are passed to the Agent that identifies the best resource, initiates computations on that resource, and returns the results. It may request assistance of other Agents in identifying the best resources and scheduling (decentralized scheduling).

6.14. Nimrod/G Resource Broker and Economy Grid

Nimrod/G [9], [10] is a Grid resource broker that allows managing and steering task farming applications (parameter studies) on computational Grids. It follows an economic (computational market) model for resource management and scheduling. It allows the study of the behavior of output variables against a range of different input scenarios. The users can easily formulate parameter studies using *declarative* parametric modeling language or GUI and run and manage the whole experiment on a global Grid with ease. The key features of Nimrod/G include: 1) support for a formulation of parameter studies, 2) a single window to manage and control experiments, 3) resource discovery, 4) trade for resources, 5) scheduling, 6) staging executables and data on Grid resource, 7) steering and data management, and 8) gathering results and presenting them to the user.

The current focus of the Nimrod/G project team is on the use of economic theories in Grid resource management and scheduling as part of a new framework called GRACE (Grid Architecture for Computational Economy). The components that make up GRACE include global scheduler (broker), bid-manager, directory server, and bid-server working closely with Grid middleware and fabrics. The GRACE infrastructure also offers generic interfaces (APIs) that the Grid tools and applications programmers can use to develop software supporting the computational economy. The Grid resource brokers such as (Nimrod/G) uses GRACE services to dynamically trade with resources owner agents to select those resources that offer low-cost access services and meet the user requirements. GRACE enabled Nimrod/G has been used for scheduling parameter sweep application’s jobs on economy Grid testbed resources spanning across four continents: Australia (Melbourne), Asia (Japan), North America (USA), and Europe (Germany) [11].

Nimrod/G follows hierarchical and computational market model in resource management [10]. It uses services of Grid middleware systems such as Globus and Legion for resource/information discovery and uses either network directory or object model based data organization. It supports both soft and hard QoS through computational economy services of GRACE infrastructure and resource reservation. The users need to define their QoS requirements explicitly (i.e., they need to define the deadline and budget as part of an experiment setup). The Grid resource estimation is performed through heuristics and historical information (load profiling) and scheduling policy is application oriented and is driven by user defined requirements such as deadline and budget limitations. The load balancing is performed through periodic rescheduling.

6.15. Ninf: A Network Enabled Server

Ninf is a client/server-based network infrastructure for global computing [7]. The Ninf system functionalities are similar to NetSolve, but they differ in the protocols and some of the APIs. It 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 efficiently marshaled and sent to the Ninf-server on a remote host responsible for executing requested library functions and send back 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 Ninf-remote libraries. When Ninf-applications invoke Ninf-library functions through APIs, the request goes to Ninf-Metaserver that maintains the information of Ninf servers in the network (using LDAP technology), and automatically allocates remote library calls dynamically on appropriate servers for load balancing or scheduling. That means, the Ninf-metaserver itself performs the resource discovery by querying its information store. Ninf-computational resources themselves register details of available library services with Ninf-metaserver (i.e., it uses push protocol in resource dissemination). Ninf follows a flat model in machine organization, schema for resource model, and relational name space organization. Ninf-Metaservers performs resource brokering or superscheduling, but actual scheduling is done using dynamic policies.

6.16. Ninja: Java Infrastructure

Ninja is a Java based infrastructure for fault tolerant, scalable, and high available Internet based applications. Ninja provides a distributed application namespaces called Multispaces [31]. Multispaces also provide service mobility so that services can migrate between nodes in a Ninja based system. The Java virtual machine with Ninja extensions and run time class libraries (an Ispace) provide a uniform execution environment between nodes. Nodes with limited resources use RMI to access non-resident services.

Components that comprise Ninja services are described using a document markup language based on XML called ISL (Interface Specification Language) [32]. Services are discovered in a Ninja system using a directory service that is called Ninja SDS (Service Discovery Service) [33]. SDS uses XML rather than attribute value pairs to describe services. Service information is periodically disseminated via multicast with plans to use agents and active messages later. The servers can be grouped into a hierarchy in order to reduce traffic loads. SDS servers are just another instance of a Ninja service and thus are accessible using standard Ninja communication mechanisms.

The Ninja system extends the Java platform in the area of performance and inter-object communication. Jaguar provides a compiler extension and run time enhancements to enhance I/O performance [34]. Ninja uses NinjaRMI, a reimplementation of Java RMI, for distributed object communication. NinjaRMI provides unicast UDP, multicast, and best effort UDP transport mechanisms that are better suited to Grid like systems than regular JavaRMI. There are some example applications such as the Ninja Jukebox, a distributed music jukebox using CD-ROM at nodes as the players and Keiretsu, an instant messaging service that integrates a variety of devices including Palm Pilots, pagers, and workstations.

Ninja can be classified as an on demand service Grid. It provides a hierarchical machine organization since there is no concept of specially designated border nodes in separate autonomous systems. The Ninja resource model is an extensible object model. Services are described in extended XML but the service definition is written in ISL, which describes the Java object interface. The namespace organization is graph based with no explicit QoS support. The resource information store is language based distributed objects since the services are all Java objects. The current resource discovery scheme is distributed queries with plans to move to agents in the future. Resource dissemination is periodic push using multicast. Ninja does not support explicit scheduling services at this time.

6.17. PUNCH: The Purdue University Network Computing Hubs

PUNCH [17][18] is a middleware testbed that provides operating system services in a network-based computing environment. It provides transparent and universal access to remote programs and resources; access-control (privacy and security) and job-control (execute, abort, and run-status) functionality in a multi-user, multi-process environment; and logical (virtual) organization and decentralized management of resources.

The PUNCH infrastructure consists of a collection of technologies and services that allow seamless management of applications, data, and machines distributed across wide-area networks. Users can access any application from anywhere 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.

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 PVM [19] and MPI [20]. 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 described layers interoperate with a distributed resource management system and a predictive performance modeling sub-system in order to make intelligent decisions in terms of selecting application implementations (e.g., sequential versus parallel), data storage sites, and hardware resources (e.g., a dedicated server versus a Condor [14] pool).

7. Discussion and Conclusion

There are many approaches/models [53] for developing Grid resource management systems. The systems we surveyed have for the most part focused on either a computational Grid or a service Grid. The only Data Grid project that we have surveyed is the CERN Data Grid, 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 with certain degree of QoS.

Any RMS that provides support for QoS, even if it is limited to network QoS, provides the basis for both computational and service Grid. A RMS that does not provide QoS is only able to provide a computational Grid. Another observation is that systems that provide QoS use a wide variety of scheduling approaches. For example 2K uses a hierarchical scheduler, Bond uses a decentralized scheduler, and MSHN uses a centralized scheduler. A topic of further research would be to correlate the scheduling attributes from different systems with different Grid sizes.

Extensibility of the resource model is a feature of all the surveyed RMS with the exception of Darwin, Javelin, and MSHN. 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 may be possible 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.

In this paper, we presented a taxonomy for Grid resource management systems. Requirements for resource management systems were described and an abstract functional model developed. The requirements and model were used to develop the taxonomy. The taxonomy focused on the type of Grid system, machine organization, resource model characterization, and scheduling characterization. Representative Grid systems were then surveyed and placed into the different categories. This helped in identifying some of the key Grid resource management approaches and issues that are yet to be explored and we expect such unexplored issues 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 Ian Foster (ANL), Jim Basney (Wisconsin), Hidemoto Nakada (ETL), Nirav Kapadia (Purdue), Heinz Stockinger (CERN), Achim Streit (Paderborn), Michael Neary (UCSB), Joern Gehring (Paderborn), John Karpovich (Virginia), Henri Casanova (UCSD), Dan Marinescu (Purdue), Andre Merzky (ZIB), Fabio Kon (UIUC), Colin Gan (Minnesota), Omer Rana (Cardiff), and Jarek Nabrzyski (Poznan) for their intellectual communications during the preparation of the manuscript.

References

[1] F. Berman and R. Wolski, *The AppLeS Project: A Status Report*, Proceedings of the 8th NEC Research Symposium, Berlin, Germany, May 1997.

[2] H. Dail, G. Obertelli, F. Berman, R. Wolski, and Andrew Grimshaw, *Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem*, Proceedings of the 9th Heterogeneous Computing Workshop, May 2000.

[3] S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, and M. Ellisman, *Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience*, Proceedings of the 9th Heterogeneous Computing Workshop, May 2000.

[4] S. Chapin, J. Karpovich, A. Grimshaw, *The Legion Resource Management System*, Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999.

[5] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit.*, Intl Journal of Supercomputer Applications, Volume 11, No. 2, 1997.

[6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[7] H. Nakada, M. Sato, S. Sekiguchi, *Design and Implementations of Ninf: towards a Global Computing Infrastructure,* Future Generation Computing Systems, Metacomputing Special Issue, October 1999.

[8] H. Casanova and J. Dongarra, *NetSolve: A Network Server for Solving Computational Science Problems*, Intl. Journal of Supercomputing Applications and High Performance Computing, Vol. 11, Number 3, 1997.

[9] R. Buyya, D. Abramson, J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, USA, 2000.

[10] R. Buyya, J. Giddy, D. Abramson, *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 2000), Kluwer Academic Press, August 1, 2000, Pittsburgh, USA.

[11] R. Buyya, D. Abramson, J. Giddy, *An Economy Grid Architecture for Service-Oriented Grid Computing*, 10th International Heterogeneous Computing Workshop (HCW 2001) (In conjunction with IPDPS 2001), San Francisco, California, USA (submitted).

[12] M. Neary, A. Phipps, S. Richman, P. Cappello, *Javelin 2.0: Java-Based Parallel Computing on the Internet*, Proceedings of European Parallel Computing Conference (Euro-Par 2000), Germany, 2000.

[13] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, *Data Management in an International Data Grid Project,* Proceedings of the first IEEE/ACM International Workshop on Grid Computing, (Springer Verlag Press, Germany), India, 2000.

[14] M. Litzkow, M. Livny, and M. W. Mutka, *Condor - A Hunter of Idle Workstations*, Proceedings of the 8th International Conference of Distributed Computing Systems, June 1988.

[15] R. Raman and M. Livny, *Matchmaking: Distributed Resource Management for High Throughput Computing*, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.

[16] J. Basney and M. Livny, *Deploying a High Throughput Computing Cluster*, High Performance Cluster Computing, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.

[17] N. Kapadia and J. Fortes, *PUNCH: An Architecture for Web-Enabled Wide-Area Network-Computing*, Cluster Computing: The Journal of Networks, Software Tools and Applications, September 1999.

[18] N. Kapadia, R. Figueiredo, and J. Fortes, *PUNCH: Web Portal for Running Tools*, IEEE Micro, May-June, 2000.

[19] V. Sunderam, A. Geist, J. Dongarra, and R. Manchek, *The PVM Concurrent Computing System: Evolution, Experiences, and Trends*, Parallel Computing Journal, Volume 20, Number 4, April 1994.

[20] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A High-Performance, Portable Implementation of the Message Passing Interface (MPI) Standard*, Parallel Computing Journal, Volume 22, Number 6, September 1996.

[21] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[22] M. Baker, R. Buyya, D. Laforenza, *The Grid: International Efforts in Global Computing*, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000), l'Aquila, Rome, Italy, July 31 - August 6. 2000.

[23] M. Maheswaran, *Quality of service driven resource management algorithms for network computing*, International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA '99), July 1999.

[24] P. Chandra, A. Fisher, C. Kosak et al., *Darwin: Customizable Resource Management for Value-Added Network Services*. 6th IEEE International Conference on Network Protocols, 1998.

[25] I. Ekmecic, I. Tartalja, and V. Milutinovic, *A survey of heterogeneous computing: Concepts and Systems,* Proceedings of the IEEE, Vol 84, No 8, Aug 1996, pp. 1127-1144.

[26] T.L. Casavant and J. G. Kuhl, *A taxonomy of scheduling in general-purpose distributed computing systems,* IEEE Transactions on Software Engineering, Vol 14, No 2, 1988, pp. 141-154.

[27] H.G. Rotithor, *Taxonomy of dynamic task scheduling schemes in distributed computing systems*, Proceedings of Computer Digital Technology, Vol 141, No 1, January 1994, pp. 1-10.

[28] N. H. Kapadia, *On the Design of a Demand-based Network Computing System: The Purdue University Network-Computing Hubs*, PhD Thesis, Purdue University, August 1999.

[29] T. Braun, J. Siegel, et al, *A Taxonomy for describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems*, IEEE Workshop on Advances in Parallel and Distributed Systems, in Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems 1998, pp. 330-335

[30] P. Chandra, A. Fisher, et al, *A Signalling Protocol for Structured Resource Allocation*, IEEE Infocom'99, New York, March 1999.

[31] S. Gribble, M. Welsh, E. Brewer, D. Culler, *The Multispace: an Evolutionary Platform for Infrastructural Services*, Proceedings of the 1999 Usenix Annual Technical Conference, Monterey, CA, June 1999.

[32] T. Hodes and R. Katz, *A Document-based Framework for Internet Application Control*, 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, October 1999.

[33] S. Czerwiniski, B. Zhao, et al, *An Architecture for Secure Service Discovery Service*, Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99), Seattle, WA, August 1999, pp. 24-35.

[34] M. Welsh and D. Culler, *Jaguar: Enabling Efficient Communication and I/O in Java*, Concurrency: Practice and Experience, Special Issue on Java for High-Performance Applications, Decmeber 1999.

[35] G. Coulson, *A Configurable Multimedia Middleware Platform*, IEEE Multimedia Vol. 6, No. 1, January – March 1999.

[36] G. Coulson and M. Clarke, *A Distributed Object Platform Infrastructure for Multimedia Applications*, Computer Communications Vol. 21, No. 9, July 1998, pp. 802-818.

[37] D. Hensgen, T. Kidd, et al, *An Overview of MSHN: The Management System for Heterogeneous Networks*, 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, April 1999, invited.

[38] G. Xie, D. Hensgen, T. Kidd, and J. Yarger, *SAAM: An Integrated Network Architecture for Integrated Services*, Proceedings of the 6th IEEE/IFIP International Workshop on Quality of Service, Napa, CA, May 1998.

[39] M. Schnaidt, D. Hensgen, et al, *Passive, Domain-Independent, End-to-End, Message Passing Performance Monitoring to Support Adaptive Applications in MSHN*, International Symposium on High Performance Distributed Computing (HPDC), Aug. 1999.

[40] F. Kon, R. Campbell, M. Mickunas, and K. Nahrstedt, *2K: A Distributed Operating System for Dynamic Heterogeneous Environments*, 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9) August 2000.

[41] F. Kon, T. Yamane, et al., *Dynamic Resource Management and Automatic Configuration of Distributed Component System*, 6th Usenix Conference on Object-Oriented Technologies and Systems (COOTS'2001) February 2001.

[42] D. Carvalho, F. Kon, et al, *Management of Environments in 2K*, 7th International Conference on Parallel and Distributed Systems (ICPADS-2000), Iwate Japan, July 4-7 2000.

[43] F. Kon, M. Roman, et al, *Monitoring and Dynamic Configuration with the dynamicTAO Reflective ORB*, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000), New York. April 3-7, 2000.

[44] L. Boloni and D. Marinescu, *An Object-Oriented Framework for Building Collaborative Network Agents*, in *Agents in Intelligent Systems and Interfaces*, A. Kandel et al, eds, Kluewer Publishing House 1999.

[45] K. Jun, L. Boloni, K. Palacz, and D. Marinescu, *Agent-Based Resource Discovery*, Proceedings of the Heterogeneous Computing Workshop (HCW 2000), IEEE Press.

[46] L. Boloni, R. Hao, K. Jun and D. Marinescu, *Subprotocols: an object oriented solution for semantic understanding of messages in a distributed object system*.

[47] D. Yau, D. Marinescu, K. Jun, *Middleware QoS Agents and Native Kernel Schedulers for Adaptive Multimedia Services and Cluster Servers*, Proceedings of the Real-Time System Symposium 99, IEEE Press, 1999.

[48] J. Gehring and A. Streit, *Robust Resource Management for Metacomputers*, 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, USA, 2000.

[49] K. Nahrstedt, H. Chu and S. Narayan, *QoS-aware Resource Management for Distributed Multimedia*, Journal on High-Speed Networking (Special Issue on Multimedia Networking), December 1998.

[50] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, *A Directory Service for Configuring High-Performance Distributed Computations*, 6th IEEE Symp. on High-Performance Distributed Computing, 1997.

[51] I. Foster, A. Roy, and V. Sander, *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*, 8th International Workshop on Quality of Service (IWQOS 2000), June 2000.

[52] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*, Journal of Network and Computer Applications (to appear).

[53] R. Buyya, S. Chapin, D. DiNucci, *Architectural Models for Resource Management in the Grid*, First IEEE/ACM International Workshop on Grid Computing (GRID 2000), Springer Verlag LNCS Series, Germany, Dec. 17, 2000, Bangalore, India.