Architecture for a Grid Operating System

Klaus Krauter and Muthucumaru Maheswaran

Advanced Networking Research Laboratory
Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2, Canada
krauter@cs.umanitoba.ca masheswar@cs.umanitoba.ca

Abstract. A Grid architecture is proposed that is motivated by the large-scale routing principles in the Internet to provide an extensible, high-performance, scalable, and secure Grid. Central to the proposed architecture is middleware called the Grid operating system (GridOS). This paper describes the components of the GridOS. The GridOS includes several novel ideas including (i) a flexible naming scheme called "Gridspaces", (ii) a service mobility protocol, and (iii) a highly decentralized Grid scheduling mechanism called the router-allocator.

1 Introduction

A Grid is a computing and data handling virtual system formed by aggregating the diverse services provided by distributed resources to synthesize problem solving environments. Some of the issues in Grid architecture are: (a) supporting adaptability, extensibility, and scalability, (b) allowing systems with different administrative policies to inter-operate while preserving site autonomy, (c) co-allocating resources, and (d) supporting quality of service.

In this paper we present a novel, highly decentralized, architecture for Grid computing that borrows features from Internet routing architectures. The central component of the architecture is the Grid operating system (GridOS). The GridOS is middleware that runs on all machines constituting the Grid. Because a Grid can have machines that range from high-performance supercomputers to handheld personal digital assistants and networks that range from gigabit-rate fiber networks to low-speed wireless LANs, it is necessary for the GridOS to be adaptable. The GridOS is designed in a modular fashion around a kernel so that the resident functionality can be changed on the fly. This design naturally supports extensibility and adaptability.

The GridOS design includes several novel ideas including (i) a flexible naming scheme called the "Gridspaces", (ii) a service mobility protocol, and (iii) a highly decentralized Grid scheduling mechanism called a router-allocator. The Gridspaces concept supports aggregation of resource names based on attributes. This enables a hierarchical resource discovery scheme that is more scalable than a flat scheme. In our architecture, the GridOS runs on each node that participates in the Grid. Therefore, it is essential that the GridOS is lightweight so that the overhead is minimal and at the same time powerful enough to support different services. One way of achieving this requirement is to dynamically instantiate services on demand. Our GridOS provides a service mobility protocol to migrate/replicate services depending on the demand. Another innovation of the GridOS is the use of request routing to decentralize resource allocation.

Section 2 describes the major features of our Grid architecture. Section 3 presents our approach to naming in the Grid. In Section 4, we examine some approaches for resource management for our architecture. Section 5 describes the architecture of the fundamental building block in our GridOS, the Grid Kernel. Section 6 presents related work.

2 Grid Architecture

In our architecture, the Grid consists of endsystems that provide resources to the Grid. Heterogeneous networks may connect the endsystems. Endsystems can range from a wireless PDA to a large cluster of supercomputers. The non-bandwidth resources such as disk capacity, computing cycles, and database services are provided by endsystems. Clients on other endsystems may use these resources The network links that interconnect the endsystems provide bandwidth resources. The Grid is responsible for managing endsystem resources, bandwidth resources, and meeting quality of service levels. Endsystems compose a series of resource requests that are routed to other endsystems that fulfill the resource requests. Specialized nodes called router-allocators examine resource requests and route them either directly to another endsystem or to another router-allocator.

Because the Grid in envisioned to scale up to Internet proportions, it is essential for the Grid to have a scalable architecture. This issue is addressed in our architecture by leveraging some important concepts from Internet routing [14]. The nodes (endsystems and router-allocators) are grouped into "autonomous systems" called Grid Domains. The nodes in a Grid domain have common resource management polices and are under the same administrative authority.

The resource management protocols are based on a datagram model. Similar to Internet routers the router-allocators do not maintain state for any resource requests that pass through them. Resource state and scheduling information is maintained on endsystems rather than on the router-allocators. Router-allocators route requests to likely endsystems but endsystems are free to reject the resource request. Resource information from endsystems and bandwidth utilization on links is periodically transmitted from the endsystems to the router-allocators. The router-allocators use this information to construct a soft state database [21] on resources so they can adapt to resource utilization and link congestion. This approach provides fault tolerance and self-healing capabilities for the Grid. In addition to this database which is contains short-lived information the router-allocators also maintain a long-lived resource capability database. Border router-allocators provide resource aggregation facilities and implement inter Grid domain resource management polices.

Router-allocators apply policy rules to resource management and the distribution of resource management information. Border router-allocators implement inter domain policies and all router-allocators within a domain implement intra domain policies.o

The endsystems and router-allocators run the GridOS which is middleware with a modular architecture. The central component of the GridOS is the Grid Kernel. The GridOS has a base set of components including the Grid Kernel and additional modules plug into the base environment to provide added functionality. This approach is similar to the extendable router architecture described in [5]. The GridOS is not a full-fledged operating system rather it is a set of processes that runs on each machine to form the middleware. The Grid Kernel is small and lightweight providing basic services for the extension modules. The Grid Kernel provides protocol processing, module management and coordination framework. The modules provide resource management functions such as scheduling policies, resource monitoring, resource accounting, and specialized resource discovery functions. The set of modules that is attached to the Grid Kernel determines the level of functionality of the corresponding GridOS. Thus the GridOS for an endsystem differs from the GridOS for a router-allocator. The security and integrity of the GridOS must be ensured so the loading and migration mechanisms will be integrated with the security management features.

3 Naming in a Grid System

In distributed systems, a name space is defined as a set of names that conform to the syntactic and semantic rules of the naming system [23]. This section, describes generalized name space for Grids called a Gridspace.

The Grid manages a widely distributed set of resources across multiple administrative domains. The naming of the objects it manages is important to the overall scalability and reliability of the Grid. The Gridspace must interoperate with or extend schemes such as DNS or LDAP. The management of Gridspace content will be specific to the objects that are being managed, thus the Gridspace management must be as extensible as the rest of the Grid architecture. The approaches described in [1] and [24] motivate our approach.

A Gridname is assigned to each object that is managed by a router-allocator. A Gridname is a set of name specifiers. A name specifier may be either an attribute-value pair or a hierarchy of attribute-value pairs. When the name specifier is a hierarchy, the upper level attribute-value pair sets up the context for the lower level attribute-value pairs. A Gridname reference is a pointer that refers to an object that is managed by another router-allocator, i.e., a reference to a Gridname in another Gridspace. A summary of the name specifiers associated with the corresponding Gridname may be associated with the Gridname reference.

A Gridspace is defined as a set consisting of Gridnames for managed objects, Gridname references to objects in other Gridspaces, and references to other Gridspaces. Gridnames within a Gridspace must be unique to that Gridspace. Thus all objects in the Grid can be uniquely named using a hierarchical naming scheme. Each router-allocator has a Gridspace and a Gridspace manager associated with it. The Gridspace manager is responsible for effecting operations on the Gridspace.

A Gridspace manager manages the content and consistency of a Gridspace. The Gridspace manager enforces no semantics on the contents of a Gridspace or the operations on those contents other than naming constraints and common operations on Gridnames. The GridOS on the managed objects will have Gridspace agents that are responsible for sending management messages to the Gridspace managers. These agent-manager transactions may be performed on detecting a change on the managed object or at predefined intervals. The predefined time intervals may be much larger for these transactions compared to the ones used for status information updates. The agent-manager model we use in the Gridspaces is similar to the model used in the simple network management protocol.

At startup, a router-allocator has an associated Gridspace and Gridspace manager. The Gridspace will be initialized to contain Gridnames for the objects managed by the router-allocator, i.e., the Gridspace would not have references to other Gridnames or references to other Gridspaces at startup. Then the router-allocators exchange the Gridnames between them. When a router-allocator receives a foreign Gridspace it will create reference entries to each of the Gridnames found in the foreign Gridspace. This is only true if the foreign Gridspace is in the same Grid domain as the receiving router-allocator. At the convergence of this process, each router-allocator within a Grid domain will have the complete knowledge of the managed objects within the domain in their Gridspace. Because the Gridnames describe objects using long-lived attributes frequent update messages are not needed to keep the Gridspace consistent.

The border router-allocators summarize their Gridspaces and advertise it to the border router-allocators of the other domains. The summarization process virtualizes the Gridnames, i.e., the Gridnames in the exported Gridspace may not have corresponding physical objects. The virtual Gridnames in the exported Gridspace reference indicate the "capability" of the managed objects to the other Gridspaces.

The Gridspace can be considered as a persistent distributed database that is weakly consistent. Suppose a resource is no longer available at an endsystem it takes the endsystem some time before Gridname entry in the associated Gridspace is updated. Although other Gridspaces have references to they may have stale summary of the name specifiers associated with the Gridname references. This makes the database weakly consistent. The weak consistency is handled by the resource management protocol that uses the Gridspace information as shown in the next section.

The approaches described in [1], [18], and [24] are being examined for potential mechanisms that can be used to efficiently implement our notion of Gridspaces. We are also investigating how to interface the Gridspaces to existing namespaces and databases in an efficient and high performance way.

4 Resource Management in a Grid System

Resource management in our architecture extends the query routing approach in [17] to the general resource management problem. The resource management protocols are based on datagrams that are exchanged between nodes. The protocols can be functionally split into resource dissemination, resource discovery and resource scheduling. Dissemination pushes information about resources from endsystems that have resources to other nodes. Resource discovery pulls resource information from other nodes to an endsystem. Resource scheduling protocols are used to assign resources to requests including co-allocation of resources across different nodes.

The resource management protocols are extensible with extensions provided by the modules that are loaded on top of the kernel. Extensibility is a problem for nodes that receive extended message content and do not have the extension modules. Possible solutions are to dynamically load extensions into the node across the network or ignore the content altogether.

The endsystems are the sources and sinks of all resource management protocol datagrams in the Grid. Resource requests are generated by endsystems that other endsystems fulfill by providing their resources. Router-allocators route resource messages between endsystems and other router-allocators. Border router-allocators apply domain resource management policies for inter domain resource messages. The scheduling of resources is performed by the endsystems. All resource management information is maintained as soft state.

Figure 1 illustrates the request routing and data dissemination processes inside a router-allocator. The two processes while sharing common data structures are functionally split. The shaded region indicates the components involved in request routing and allocation and the unshaded region indicates the components involved in data dissemination.

4.1 Resource Status Dissemination

A large number of nodes may make uniform status dissemination too costly . Heterogeneity in resource characteristics and resource requests may make it unnecessary to uniformly disseminate the status [19].

One of the major overheads of dissemination are messages that keep the distributed status database consistent. We can use different properties of the Grid environment to minimize update messages. Resource attributes are split into two classes: short-lived attributes and long-lived attributes. Short-lived attributes are disseminated through the Grid status registry mechanism shown in Figure 1. Long-lived attributes are disseminated via Gridspaces. Consequently, it is possible to have different update policies for Gridspaces and Grid status registries. One approach is to use frequent update messages for the Grid status registry but to restrict the update message propagation area. Alternatively, Gridspaces may be updated less frequently but on a larger propagation area. The extent of update message propagation for the Grid status registry may be

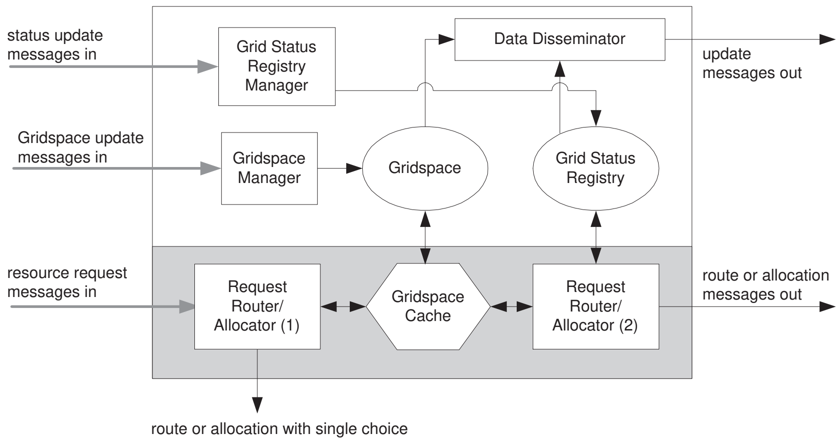

Fig. 1. Request routing and data dissemination in a router-allocator

decided based on the importance and uniqueness of the resource or based on administrative policies [19].

In addition to disseminating resource status information it is also necessary to disseminate information regarding the services offered by the different Grid nodes. Including the Gridnames that describe the services in the respective Gridspaces performs a transparent dissemination of service information. This feature aids the service mobility that is described below.

To enhance scalability , our architecture divides the overall system into Grid domains. Border router-allocators are responsible for connecting domains to form the Grid and disseminating data between domains. Administrative policies are used to restrict the content of the data dissemination messages. Border router-allocators also aggregate the Gridspace update messages using the Gridspace aggregation policies. Updating the Grid status registry dependents on the policy. One policy may update status only within a Grid domain and another policy would update based on a dynamically chosen network distance.

4.2 Resource Discovery

Resource discovery in distributed systems is typically based on a distributed database approach or a mobile agent approach. In our architecture, we use a hybrid approach. For example, the Gridspace is maintained as a weakly consistent distributed database. Because of the way Gridspaces aggregate the attributes of the managed objects there may not be detailed information for a remote object. Once a resource discovery query walks through the aggregated Gridspace that entry is cached in the Gridspace cache. This can be considered as an agent based approach. However, the benefit of this approach over the agent based approach is that the subsequent queries that may be interested in the same object need not incur the full overhead until the cached entries are timed out.

4.3 Resource Scheduling

Resource scheduling is done in a distributed way using both the router-allocators and endsystems. Endsystems are responsible for the scheduling of the resources on their node. Endsystems also generate requests for resources elsewhere in the Grid. When a resource request arrives at a router-allocator, the router-allocator consults the Gridspace to determine the best way to handle the resource request.

The router-allocator may process the request in several ways. If it determines that a remote resource is best for handling the request it will forward the request to the corresponding router-allocator. Alternatively, the router-allocator may find a loaded Grid scheduling service and delegate the request processing. This scheduling service may be implemented using wide-area schedulers such as Globus, Gallop, or MSHN. The scheduling services are registered with the Gridspace at the router-allocator and the request routing mechanism will find the service when it queries the Gridspace. In some situations, it may be necessary to schedule a request over several different resources (referred to as co-allocation or multi-scheduling in [8]).

Scheduling uses a three-phase protocol between the endsystems providing the resources and the endsystems requesting the resources. The protocol also follows an end-to-end allocation model rather than following an in-core allocation model. More specifically, the router-allocators just facilitate the resource scheduling by guiding the resource requests to the most appropriate endsystem. They do not keep any state information that keep track of the request-to-resource mappings and make any binding decisions.

When a resource request arrives at a router-allocator, it consults its Gridspace and Grid status registry as shown in Figure 1 and may decided to split the original request into a number of smaller requests and route the request fragments on to other endsystems and other router-allocators. This is a recursive process that can cross Grid domains. The endsystems then send a response indicating if the resource can be scheduled. The responses then return to the originating endsystem, which then sends out a scheduling request to the endsystems that responded to the resource request. This completes the three-way handshake for scheduling resources.

There are different ways of routing the acknowledgements and scheduling request messages. The request acknowledgements could be sent directly to the originating system or via the router-allocators. Sending the messages via the router-allocators allows the dynamic updating of Grid status registries without waiting for the periodic resource dissemination messages. There are two different options for the message flow when scheduling resources. The first option is directly send scheduling messages from the requesting endsystem to the endsystem providing the resources. The second option is to route the messages along the router-allocators. The first option does not inform the router-allocators of resource allocation. The second option enables the router-allocator to update the Grid status registry entries.

4.4 Service Mobility Protocol

The purpose of the service mobility protocol is to provide extensibility and adaptability to the GridOS while maintaining the lightweight nature of the Grid Kernel so that the Grid can execute on a wide variety of platforms. Consider a Grid domain with several endsystems and a single router-allocator. This router-allocator also acts as the border router-allocator because it is the sole router-allocator for the domain. As the load on the node increases it may be necessary to off-load some of the work by migrating the router-allocator to another node or replicating the router-allocator functionality to another node and sharing the load.

The GridOS provides service mobility as a built-in service at each node. This service is responsible for monitoring the mobility-enabled services and deciding when and where they should be replicated or migrated. Although our design does not allow new migration or replication policies to be added to the system it allows the Grid domain administrator to set the values of fine-tuning parameters for the migration and replication logic. In the simplest case, the mobility protocol module monitors the CPU load at the local node and when a threshold set by the domain administrator is reached, it will initiate the replication and/or migration process. It requests other potential nodes for bids for hosting the service that needs to be moved. Other considerations such as network vicinity may be considered in deciding the set of potential nodes. All nodes, i.e., the current node and the potential nodes are in the same administrative domain.

After receiving the bids, the mobility service needs to decide which node it is going to choose and whether it is going to replicate or migrate. Of it replicates the service to another node it will need to modify the Gridspaces in the local node as well as the remote node to reflect this replication. The data dissemination protocol is used to inform the Gridspaces in the other nodes about the new instance of the service. Instead of simply monitoring the CPU load at the node, it may be beneficial to use a combination of fraction of the CPU cycles delivered to the service under consideration and the demand for the service. The replication may be made further efficient if the service is replicated to a location closer to the region where the service is in high demand [25].

5 Grid Kernel Architecture

The GridOS runs on all nodes that are part of the Grid. The GridOS is organized in a modular fashion with Grid Kernel as the central component. The Grid Kernel is designed to be small, efficient and lightweight. The Grid Kernel provides only functions that are absolutely required to have a node be part of the Grid. The extension modules provide all other functions.

The kernel provides basic functions such as the processing of the resource management protocols, the module management of the extension modules, and the Grid security on a node. It is run as a user mode application using the services of the host operating system to execute and monitor jobs. The Grid kernel does not require system mode operating systems privileges but some of the extension modules may be interfacing with native operating system services that require extended privileges. The kernel does not require that the module be written in a particular language only that it provides a standard binary interface so that the proper functions can be invoked.

The kernel is structured in layers. The bottom layer, the kernel to native operating system interface is responsible for providing a uniform interface to native functions. The second layer consists of the resource protocol processing component, module management component, and a security manager component. The top layer, the module to kernel interface layer provides a uniform interface for extension modules to the services provided by the kernel and the loaded modules.

The resource protocol processing component is responsible for processing the protocol messages and protocol events. The processing involves dispatching the extension modules that handle message content. The module management component keeps track of the modules that are currently loaded on top of the kernel. It also provides the services to dynamically load and unload the modules. The Grid security manager authenticates the modules that may be loaded dynamically. The security manager implements the generic security mechanisms used by the modules. There may be a number of different security authorization and authentication modules specific to the resources used by the node but the security manager manages the overall access and control in conjunction with these modules.

The use of modules and Gridspaces on top of a common kernel provides the ability to have Grid software that is customized to the needs of individual nodes. Thus each node in the Grid only runs the modules that are required to manage the resources specific to the node. This approach reduces the overhead required for a node to be part of the Grid. For example, a workstation, supercomputer, and bandwidth broker would have very few modules in common since the resources provided by these nodes to the Grid vary widely.

The module structure for an endsystem and a router-allocator node are somewhat different. The Grid kernel and Grid Manager module are common but the router-allocator would contain routing and resource routing policy manager modules whereas the endsystem would contain resource specific manager modules. The Grid Manager module is a mandatory on all nodes since it provides the basic functions for the node to be part of the Grid.

The endsystem node would have a number of different resource managers that manage requests for a specific resource such as a specialized database or computational resource. The request routing and allocation module in the router-allocator is responsible for handling the resource requests coming in to the router-allocator. Based on the information it finds in the Gridspace and the Grid status registry, it may either send the request to another router-allocator or to an endsystem. Any administrative policies that may determine the outcome of the resource request processing are also implemented by this module.

The Grid manager module consists of a Gridspace naming module, a node monitoring module, and a service mobility module. The naming module enforces the local and global Gridspace naming rules. The node monitoring module monitors the state of the local node and notifies other modules or the kernel of state changes. The service mobility module implements the moving of modules and their associated Gridspaces from one node to another node.

Resource specific manager modules typically consists of a resource management component, a Gridspace management agent component, a resource monitoring component, and a resource specific security component. The resource management component interfaces with the native resource management functions to perform the actual scheduling of the resource. The Gridspace management agent is very similar to the simple network management protocol agent. The Gridspace management agent is responsible for tracking that status of the managed object and notifying the Gridspace manager about the relevant changes. The resource monitoring component is used to track resource utilization rates and important events for that resource. The resource specific security component interfaces to the authentication and authorization module that this resource uses to authenticate and authorize resource requests from another Grid node.

The kernel structure, module management, resource monitoring facilities, and service mobility make it possible to dynamically change the resources and behavior of the Grid. For example, there is no significant difference between a router-allocator and an endsystem node other than the modules that are loaded on top of the Grid kernel. The node that is hosting different Grid modules to make up a router-allocator may become heavily loaded. This will prompt the service mobility protocol to replicate/migrate the router-allocator functionality to other nodes thus shedding some of the load on the node. Further, the failure of a router-allocator could be detected by an endsystem and it could then load the required modules to provide this service. Endsystems may also migrate some services to other nodes within the Grid.

6 Related Work

Existing Grid computing systems can be divided into two categories, application enabling systems and user access systems. The application enabling systems provide application programmers with tools that allow applications to access globally distributed resources in an integrated fashion. Examples of such systems include ATLAS [2], Globe [13], Globus/GUSTO [6], [7], Legion [10], and ParaWeb [3]. The user access systems provide the end users of the Grid with transparent access to geographically distributed resources in a location independent manner. Examples of such systems include CCS [20], MOL [22], NetSolve [4], and PUNCH [15].

The Grid Integrated Architecture [9], an extension of the Globus/GUSTO effort, intends to provide a globally distributed uniform infrastructure. Our GridOS corresponds to part of the Grid services layer in the architecture. The major difference between our GridOS architecture and the architecture in [9] is that our architecture features a Grid with fine grain adaptability. With the service mobility protocol and the extensible Grid Kernel design, it is possible to adapt the different nodes of the Grid according to the usage requirements and the node's capacity. The Ninja project [11], [12] is building a network computing structure centered on "network documents" and implemented using Java. There are similarities between their multispaces and our Gridspaces but we focus on using Gridspaces as a weakly consistent capability database and they use multispaces as function repository. Their use of XML in network document is for dynamic application loading rather than for resource specific management extensions.

7 Conclusions

This paper presents a Grid architecture that is motivated by the large-scale routing principles in the Internet to provide an extensible, high-performance, scalable, and secure Grid. The central idea of the proposed architecture is to layer a Grid operating system on top of the resources to construct a Grid. The resources may have their own schedulers, accounting mechanisms, and security mechanisms. The GridOS interfaces with the services provided by the local resources and exports them to the Grid level. This paper describes the components of the GridOS.

The GridOS includes several novel ideas including (i) a flexible naming scheme called the "Gridspaces", (ii) a service mobility protocol, and (iii) a highly decentralized Grid scheduling mechanism called the router-allocator. The combination of flexible naming, service adaptability, service extensibility, and highly decentralized resource management results in a novel Grid architecture. A Grid system based on this architecture can execute across a widely varying resource set that may include wireless PDAs and powerful supercomputers.

References

1. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system", Operating Systems Review, Vol. 34, No. 5, Dec. 1999, pp. 186-201.
2. J. E. Baldeschwieler, R. D. Blumofe, and E. A. Brewer, "ATLAS: An infrastructure for global computing", 7th ACM SIGOPS European Workshop, 1996.
3. T. Brecht, H. Sandhu, M. Shan, and J. Talbot, "ParaWeb: Towards world-wide supercomputing", 7th ACM SIGOPS European Workshop, 1996.
4. H. Casanova and J. Dongarra, "Netsolve: A network solver for solving computational science problems", Supercomputing, 1996.
5. D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, "Router plugins: A software architecture for next-generation routers", IEEE/ACM Transactions on Networking, Vol. 8, No. 1, Feb. 2000, pp. 2-15.
6. I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit", Int'l Journal of Supercomputer Application, Vol. 11, 1997.

7. I. Foster and C. Kesselman, "The Globus project: A status report", 1998 IEEE Heterogeneous Computing Workshop (HCW '98), 1998, pp. 4-18.
8. I. Foster and C. Kesselman, eds., The Grid: Blueprint for a new computing infrastructure, Morgan Kaufmann, San Francisco, CA, 1999.
9. I. Foster, "Building the Grid: Integrated services and toolkit architecture for next generation networked applications", www.Gridforum.org/building_the_Grid.htm, 1999.
10. A. S. Grimshaw, W. A. Wulf, and et. al., "The Legion vision of a world-wide virtual computer", Communications of the ACM, Vol. 40, 1997.
11. S. Gribble, M. Welsh, E, Brewer, and D. Culler, "The MultiSpace: an evolutionary platform for infrastructural services", 1999 Usenix Annual Technical Conference, 1999.
12. T. Hodes and R. Katz, "A document-based framework for Internet application control", 2nd USENIX Symposium on Internet Technologies and Systems, 1999.
13. P. Homburg, M. v. Steen, and A. S. Tennanbaum, "An architecture for a wide area distributed system", 7th ACM SIGOPS European Workshop, 1996.
14. C. Huitema, "Routing in the Internet", 2nd Edition, Prentice-Hall, Upper Saddle River, NJ, 2000.
15. 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; Special Issue on High Performance Distributed Computing. 1999.
16. K. Krauter and M. Maheswaran, "Towards a High Performance Extensible Grid Architecture", HPCS 2000, June 2000
17. P. Leach and C. Wieder, "Query routing: Applying systems thinking to Internet search", 6th Workshop in Hot Topics in Operating Systems, 1997, pp. 82-86.
18. J. Ordville and B. P. Miller, "Distributed active catalogs and meta-data caching in descriptive name services", IEEE Int'l Conference on Distributed Computing Systems, May 1993, pp. 120-129.
19. M. Maheswaran and K. Krauter, A Parameter-based Approach to Resource Discovery in Grid Computing Systems, TR-CS, Computer Science, University of Manitoba, under preparation.
20. F. Ramme, "Building a virtual machine-room a focal point in metacomputing", Future Generation Computer Systems, Vol. 11, 1995.
21. S. Raman and S. McCanne, "A model, analysis, and protocol framework for soft state-based communication", ACM SIGCOMM, 1999, pp. 15-25.
22. A. Reinefeld, R. Baraglia, T. Decker, J. Gehring, D. Laforenza, F. Ramme, T. Romke, and J. Simon, "The MOL project: An open, extensible metacomputer", 1997 IEEE Heterogeneous Computing Workshop (HCW '97), 1997, pp. 17-31.
23. P. K. Sinha, Distributed Operating Systems: Concepts and Design, IEEE Press, New York, NY, 1997.
24. A. Vahdat, M. Dahlin, T. Anderson and A. Aggarwal, "Active names: flexible location and transport of wide-area resources", USENIX Symposium on Internet Technologies and Systems, 1999.
25. T. Vaseeharan and M. Maheswaran, "Towards a novel architecture for wide-area data caching and replication", First International Conference on Internet Computing (IC 2000), June 2000