Key Concepts and Services of a Grid Information Service

Beth Plale
Computer Science Dept.
Indiana University
Bloomington, IN

Peter Dinda
Computer Science Dept.
Northwestern University
Evansville, IL

Gregor von Laszewski
MCS
Argonne National Lab
Chicago, IL

Abstract

The term "Grid" has become common parlance among parallel and distributed computer scientists to denote a middleware infrastructure for wide-area scientific and engineering computing. The discussion of standards and best practices is ongoing in the Grid Forum community to identify key pieces of the middleware infrastructure. One of the areas under discussion is different models for grid information services.

The contribution of this paper is a taxonomy of the grid information services. That is, of key functionality that a grid information service must provide. The paper also presents use cases as means to begin to understand user needs. Weaved throughout is a discussion of two general models of information representation, and the impact of the model on the services provided.

1 Introduction

The Grid [8] denotes a middleware infrastructure for wide-area scientific and engineering computing characterized by multiple administrative domains and geographically broad distribution of resources and users. Current research is directed toward establishing the key services that make up the Grid. Though the grid community agrees that IPv4/IPv6 is a good current and immediate future choice for the underlying network protocol, there is ongoing discussion about middleware services such as global naming, remote process execution, authentication, and security. Solutions implementing multiple key services exist in the Condor [2], Globus [6], and Legion [9] systems. While Globus currently enjoys broad support in the European community, no one solution has emerged as the standard. The topic of this paper is one of those key services, the grid information service.

A grid information service (GIS) is the Grid equivalent of the directory service. It is a software component, whether singular or distributed, that maintains information about people, software, services, and hardware that participate in a computational grid, and makes that information available upon request. A grid information service can also provide binding, discovery, lookup, and data protection. While directory services are generally focused at the network level, that is, mapping a hostname to an IP address, or a location transparent name to its host-specific equivalent, a grid information service must represent a richer set of entities having richer relationships between them and more dynamic information. For instance, whereas a machine's IP address is quite stable, the current average CPU load over multiple CPUs in an SMP can be as dynamic as an information service will allow.

The authors represent two sides of the grid information service discussion. In one view, grid resource information is best served by a hierarchical representation, and on the other side, that a relational or flat table representation is more suitable. A key requirement to understanding the points of view is an understanding of the key services. That is, the kinds of information to be represented, the usage patterns on that data, and the replication, distribution, and authorization policies.

The contribution of this paper is a taxonomy of grid information services. The paper also presents use cases as means to understand user needs. In the discussion of concepts and services we remark on how the different features manifest themselves in two types of directory services, an LDAP service such as MDS-1 and MDS-2 [3], and a relational database approach, such as is promoted in [4].

2 Services of Grid Information Service

2.1 Terms

A grid information service provides information about entities in a grid. An *entity* is something of value to a computational grid. Entities can be services, such as software; serve as a resource, such as a compute cluster, or exist as a person, group, or organization. A list of possible grid entities is shown in Figure 1. An entity has representation in a grid information service as an object. An *object* is a representation of a real-world entity. Objects are often described by a set of value-attribute pairs. 'Object' is used in its most general sense, that is, there is no assumption about any particular data model. An *object class* is a template or type to which an object belongs. An object is an instantiation of the object class; that is, it is a concrete representation of a single real world Grid entity. Finally, a *relation* is a link between two objects. Two workstations may be related in that they both exist as nodes in a cluster.

An object exists in the GIS on behalf of an entity if the entity meets the following four criteria:

1. Usefulness: the entity can be described succinctly in a manner that captures key attributes.

2. Uniqueness: the entity is distinguishable from other entities of the same type

3. Persistence: the entity is long lasting. The attributes of an entity may not transient, however.

4. Generality: the entity has value to multiple applications or users.

Many Grid entities meet these criteria and a number of specific entities already either exist or have been proposed. These are listed in Figure 1.

A *data model* describes the structure of objects and the relationships between objects in a data repository. There are four basic data models: hierarchical, network, relational, and object-oriented. The four models are nearly indistinguishable in their representation of entity descriptions; it is in the level of support for relationships that they differ. As the hierarchical and network models share much in common, and similarly the relational and object-oriented model share much, we focus our attention on the hierarchical and relational models.

The hierarchical model represents an object name space as one or more rooted trees. One object type serves as the root, and all other entities are related to a root. Traditionally the hierarchical model has support for a single relationship, that of parent to child. As such, relationships are not named and navigation is accomplished by specifying entity names. LDAP and XML are hierarchical; both extend the hierarchical model with 'aliases' or 'pointers', essentially references to other objects. The reference relationship increases the expressive power of the model, but the links can be costly to traverse during query execution.

The relational data model represents information as flat tables, or relations[1]. A relation is an object type; the objects themselves are tuples defined by a set of value/attribute pairs. Relations represent both entities and relationships between entities. The ability of the relational data model to represent any relationship between entities (not just the parent-child relationship) makes it superior to the hierarchical model for expressing complex relationships. In the following sections we detail the key services of a GIS.

2.2 Update Interface

An update to an object is the the irrecoverable replacement of the existing attribute values with new values. This update-in-place is in contrast to a versioning scheme [11]; the latter of which benefits from easier failure recovery. The update interface should allow the creation of object classes. Extending built-in types with attributes that are specific to an individual may or may not be supported. In an object-oriented representation this would be done with subclassing or inheritance.

Information services fall into one of three categories with respect to updates:

- *Read-only repositories* have no standard means of changing the information in them. Updates are usually accomplished through some other interface than the standard interface, such as changes to a configuration file. DNS serves in this capacity. DNS is read-only; it updates information it discovers, but the external interface is query (read) only.
- *Read-mostly repositories* allow updates to occur, but are optimized for reads. This may, for instance, manifest itself in relatively slow consistency-updating protocols, or time consuming restructuring in the face of multiple additions or repositions in the data structure. LDAP falls into this category.
- *Read-write repositories* assume that updates and read operations occur with same order of magnitude. A relational database management system is a read-write repository.

Updates to an object in the GIS should be under some degree of control of the owner of the entity being represented. That is, the owner should be able to initiate changes to the object when the resource changes (e.g., a cluster is retired, a file server upgraded, a compute server gets another network interface). Update fre-

[1]Can be thought of as flat tables of attributes where the attributes are simple (no arrays, records, etc.)

quency should be supported at a rate that is perceived by the customer as timely. This notion of timeliness can vary among resources, but the owner of a supercomputer or fast network path for instance, needs an accurate up-to-date description of the resource. The more accurate, the more likely the resource (and the information service) will be used by others.

Grid Entity	*Description*
organizations	accountable bodies, resource owners
people	resource admins, resource providers, GIS admins
physical resources	compute resources, network interfaces
services	job manager, load leveler, other GIS'
comm resources	link capacity, switch capacity, error rate, drop rate
software pkgs	BLAS, LAPACK, etc.
event producers	event stream generators
event channels	event stream propagators
event dictionary	database event types
instruments	radar systems, telescopes, etc
network paths	avail bandwidth, expected latency
network topologies	hosts, switches, routers
wireless devices	wireless hosts, wavepoints, cells, etc.
virtual orgs	groups of collaborators
...	...

Table 1: Types of Grid entities requiring representation in a grid information service.

2.3 Query Interface

The query interface defines the way in which information is retrieved from the information service. Information is retrieved by other grid services, such as a job scheduler; grid applications, such as an application that plans computational resource needs based on location of data sets; and users. The groups needing query access are not necessarily mutually exclusive with the groups needing update access. The owner of a large cluster is likely to be interested in obtaining descriptions of other large clusters. We suspect, however, that the numbers of users querying the information service are considerably larger than those needing update rights.

The sophistication of the query interface determines the ease with which a user can write queries, the efficiency of query execution, and the frequency of queries. Relational and object oriented databases support a standardized and powerful access method like Structured Query Language (SQL). SQL queries can be complex, reducing the total number of queries a user must submit to get a desired answer, and because SQL is declarative, queries are optimized so are highly efficient. Older hierarchical models such as LDAP use simplified access protocols. The queries are restricted to simple expressions expressed in a procedural language. Because of the hierarchical structure of LDAP and XML, a user must possess intimate knowledge of the tree structure to write queries [10].

2.4 Distribution and Replication

The distribution of a grid information service is the partitioning of the data space and distribution of the parts across a wide area. DNS illustrates distribution well. The name space is hierarchically organized into a tree of domains which are divided into non-overlapping zones. The names in a zone are handled by a single name server. Resolution of a single name can involve DNS servers in several zones, as the name is resolved from general (*e.g.*, .edu) to specific (*.cs*). Where not all sites have all the information, mechanisms should exist to get the information to the requester, even when it is not available at the site originally asked.

The hierarchical data model is well suited to the distribution needs of the computational grid because the rooted tree lends itself well to partitioning into subtrees by administrative domains. Partitioning a RDBMS is not as intuitive; it requires slicing tables by rows and parceling out the sets of rows.

MDS-2 employs a unique distribution scheme; it logically and physically separates the data space and general indexes (the GRIS,) from additional user-defined indexes, (the GIIS.) The purpose is to give the user more efficient access to multiple resource repositories (*i.e.*, GRIS').

Replication is the duplication of data across multiple sites. Replication increases availability and helps balance workload between replicas. In widely dispersed systems like the grid, replication can also reduce communication latency. Replication brings with it the problem of replica consistency. MySQL, for instance, supports strong consistency with a write-through policy on updates. But write-through can be problematic without some kind of soft-state policy like leases [15] for recovery of partial failures. While strong consistency may not be practical in a grid information system, weaker policies result in outdated information being returned which in turn could result in less optimal scheduling decisions being made by a job scheduler.

2.5 Security

Security deals with both trusting the information and trusting the users that access or update the information. In particular, the directory service must support authorization, authentication, and integrity. *Authorization* is the verification that users are allowed to perform requested operations on the data objects. The resource owner must be able to control access to and even visibility of the resource depending on the prospective user. *Authentication* is the assurance that the opposite party (machine or person) really is who he/she claims to be. And *integrity* asks the question "why do we trust a piece of information to be correct?"

There are several categories of authorization:

- Open access: Anyone can get the information,
- Property-based access: Access because of what one is, or where one is. For example, access could be limited to those on "same network", "physically present" or with "resolvable DNS name,"
- Identity-based access: Access because of who one is, or successfully claims to be. Provided through username/password, certificates. This type of access is backed up by a layer specifying what one has access to given their identity, and
- Token-based access: Access because of what one possesses. For instance, hardware tokens, smartcards, certificates, capability keys. Access is given to all who can present that credential, without caring about their identity.

The integrity design space is also broad:

- Because it's in the repository (and therefore must have been authorized). This is perimeter (or Eggshell) integrity,
- Because it contains internal integrity checks, usually involving digital signatures by verifiable identities This is item integrity, and
- Because it fits other available information, and causes the right things to happen when used. This is hopeful integrity.

The most common approaches to authorization are identity-based access and open access; token-based access is commonly used informally in, for example, password-protected FTP or Web sites where the password is shared between all members of a group. In best practices recommended by the Global Grid Forum community, authorization and authentication are achieved through X.509 Certificates with support for delegation to a proxy. The appropriate level of trust to adopt requires evaluating the cost of implementing the integrity, the cost of having the integrity break, and the impact of cost on doing business.

3 Sample Use Cases

3.1 Traditional parallel applications

A simple, but commonplace data decomposition problem in data-parallel SPMD programs—how to partition the rows of a matrix into blocks, and then assign those blocks to processors so that computation done on the matrix according to the owner-computes rule will achieve high levels of speedup. On a dedicated parallel machine or cluster, the static properties of the resources that are needed to solve this problem are known implicitly—the programmer knows what processors the cluster uses and the topology and capabilities of its communication network. The dynamic properties of the resources are a non-issue. On a non-dedicated cluster, life is more complex, but still, the scope of the information that is needed is limited.

On a computational grid, the static properties of the compute and communications resources are not implicit, and their dynamic properties usually necessitate run-time adaptation. Both to initially map itself onto the grid, and then to adapt its behavior as it runs, the application needs detailed information about these resources. Furthermore, it is generally not interested in individual resources per se, but rather in compositions of them. For example, if it has been coded to run on four processors, then, at startup, it will want to ask things such as "find me a set of four unique hosts which in total have between 0.5 and 1 GB of memory and which are connected by network paths that can provide at least 2 MB/s of bandwidth with no more than 100 milliseconds of latency." As it runs, it will need to know when "the load on any one of four selected hosts is at least 25% different from the average across all hosts".

3.2 Traditional distributed applications

Workflow-style distributed applications create a virtual datapath for requests on top of the grid's resources. This induces a mapping process very similar to a traditional parallel application. The application will ask such things as "find me four processors such that if I map my datapath onto them in this manner then I can achieve a throughput of n requests per second while still keeping the latency below 10 seconds."

3.3 Non-traditional applications

Non-traditional applications, especially those that have interactivity demands that can be expressed as real-time constraints, are emerging to take advantage of the explosion of resources provided by grid-based computing. Examples of these include interactive scientific visualization, distributed laboratories, and computer-aided design. These applications are composed of interacting programs, actuators, sensors, remote data sources, and distributed users and typically have high I/O or computation demands necessitating the inclusion of at least one high-end computational resource such as an SMP-node cluster.

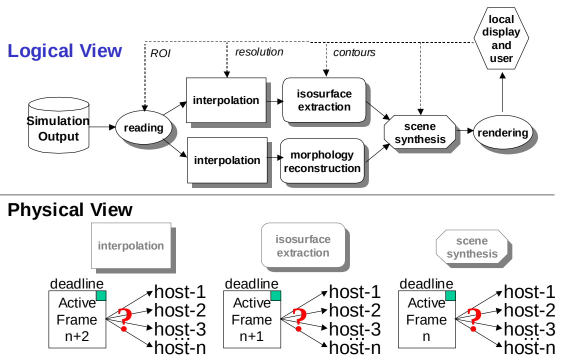

Figure 1: Example DV visualization pipeline: logical view as filters operating on stream, and physical view as active frames moving through compute resources.

Figure 1 illustrates one such application, Dv [1]. Dv is a framework for constructing interactive scientific visualizations for wide-area environments. The Dv programmer can trivially transform a sequential C++ vtk (vtk [13] is a popular toolkit for building visualizations) program into a Dv program. The logical view of a Dv program is as a flowgraph, as shown in the top of the Figure 1. When the user requests that some region of interest in a remote database be transformed via the flowgraph to a display on his workstation, an "active frame" is sent to the remote database. Embedded in the active frame is the code that implements the flowgraph, and a scheduler that assigns nodes on the flowgraph to hosts in the Grid. The active frame reads the region of interest from the database, and then propagates back to the user, executing flowgraph nodes and reevaluating its schedule as each flowgraph node is executed. The goal of the scheduler is to choose where to execute the flowgraph nodes such that and end-to-end deadline can be met with high probability.

3.4 Applications with Indirect Reliance on Grid Information Service

Applications exist that rely on systems indirectly, making use of other Grid software that uses the directory service. Two important examples are systems that provide network information and schedulers that map applications to Grid resources.

While the information generated by these systems may not meet the strict criteria for inclusion in a grid information system, the structure and location(s) of

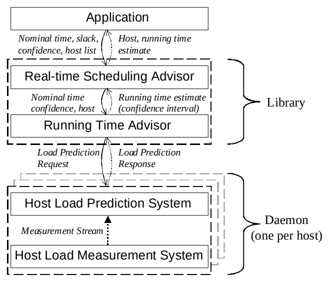
Application
Nominal time, slack, confidence, host list
Host, running time estimate
Real-time Scheduling Advisor
Nominal time confidence, host
Running time estimate (confidence interval)
Running Time Advisor
Library
Load Prediction Request
Load Prediction Response
Host Load Prediction System
Measurement Stream
Host Load Measurement System
Daemon (one per host)

Figure 2: RPS system: conceptual view.

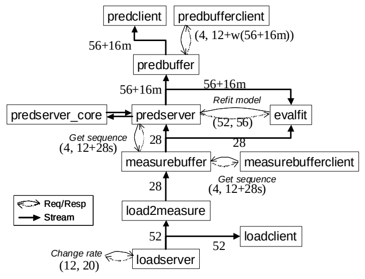

Figure 3: RPS system: constellation of components that must be managed.

their instances often does. For example, the RPS system [5] system provides time-series predictions of fine-grain measurement streams. While RPS is conceptually quite simple (see Figure 2), it is designed to be highly flexible in terms of where its components run and how they communicate. When configured to predict, for example, the load on a host, the components form a "prediction pipeline" such as can be seen in Figure 3. Each component, or box, in the figure can be run on a different host, and each communication path (arrow) can use a different underlying protocol. This flexibility is important because the costs of measurement and prediction vary widely with the resource being monitored, making it absolutely necessary to be able to say "run the (expensive) predictor on *that* machine." RPS is not alone in its needs. Remos [12] and NWS [14], also need to manage large numbers of components with complex relationships.

Grid schedulers are perhaps the most common interface that applications have and will have to the Grid. To do their work of mapping applications to Grid resources, Grid schedulers generate frequent and complex queries over diverse resources.

4 Conclusion

A clear benefit of a hierarchical name space is that it naturally decomposes across administrative boundaries. Localized control within an administrative domain is essential for a wide area solution if broad acceptance is to be achieved. Our vision of a distributed grid information service, giving nod to DNS, is a hierarchical decomposition of the global name space into a top layer under centralized control. But at the lowest level, where the resources themselves reside and the need for complex relationships is greater, administrators can be free to choose the implementation that best suits their needs. They may choose to decompose the hierarchy all the way down to the leaves, or may instead choose a non-hierarchical representation.

To enable interoperability between levels, the interface language must be expressive. The Open Grid Services Architecture (OGSA) effort [7] is a positive step toward standardization on a richer protocol.

References

[1] M. Aeschlimann, P. Dinda, L. Kallivokas, J. Lopez, B. Lowekamp, and D. O'Hallaron. Preliminary report on the design of a framework for distributed visualization. In *Int'l Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1833–1839, Las Vegas, NV, June 1999. CSREA Press.

[2] Jim Basney and Miron Livny. *High Performance Cluster Computing*. Prentice Hall PTR, 1999.

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.

[4] Peter Dinda and Beth Plale. GWD-GIS-012-1 Unified relational approach to grid information services. http://www.gridforum.org/1_GIS/gis.htm, 2001.

[5] Peter A. Dinda and David R. O'Hallaron. An extensible toolkit for resource prediction in distributed systems. Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. http://www.globus.org, January 2002.

[8] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[9] A. S. Grimshaw, W. A. Wulf, and the Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), 1997.

[10] H. V. Jagadish, Laks V. S. Lakshmanan, Tova Milo, Divesh Srivastava, and Dimitra Vista. Querying network directories. In *ACM SIGMOD Conference*, 1999.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, , and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Ninth Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[12] Bruce Lowekamp, Nancy Miller, Dean Sutherland, Thomas Gross, Peter Steenkiste, and Jaspal Subhlok. A resource monitoring system for network-aware applications. In *7th IEEE Int'l High Performance Distributed Computing (HPDC)*, pages 189–196. IEEE, July 1998.

[13] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Prentice Hall, second edition, 1998.

[14] Rich Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)*, pages 316–325, August 1997.

[15] Jian Yin, Lorenzo Alvisi, Michael Dahlin, and Calvin Lin. Using leases to support server-driven consistency in large-scale systems. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, Los Alamitos, CA, 1998. IEEE Computer Society.