Peter Dinda, Northwestern U and Beth Plale, Georgia Tech
February 2001

Information Services
Request for Comments: GWD-GIS-012-1
Obsoletes: none
Category: Informational

A Unified Relational Approach to Grid Information Services

Abstract

This first white paper of the relational Grid information services task group outlines an approach to Grid information services (GIS) that is based on the relational data model. While conceived as operating stand-alone, the system proposed herein could also be viewed as a complementary system, or as a set of services that would complement and interoperate with systems built according to the standards emerging from the Grid Information Services working group and the Grid Performance working group.

There is much that is excellent about the evolving standards of these two groups. Their data models are simple, their implementations will be easy to use, and the data they will contain will decompose naturally across administrative domains. However, there are many forms of highly desirable queries that will be difficult or expensive to support in these systems. Generally, such queries involve constraints on compositions of information ("joins"). "Give me n hosts that together have a total memory of one gigabyte" is an example. In addition, we expect that queries will evolve as the Grid evolves, so it is vital that the set of efficient queries not be decided a priori, as it is with the current models. We also see that there are *two* evolving standards—two incompatible ways at getting at information about Grid resources. Finally, it is essentially the case that highly dynamic information is not represented in the GIS system because the latter is unlikely to be able to support the necessary update rates.

We intend to address these concerns by a unified relational approach to Grid Information Services. By "relational" we mean that we will start with the full functionality of a relational database system and "build down" to a practical GIS system that still provides most of the benefits of the RDBMS, and in the process analyze the desirability and feasibility of such features as transactions and ACID properties. Existing approaches to building GIS systems have essentially been to start from a directory service with a hierarchical data model and "build up" to a practical GIS system. We believe that a "build down" approach will result in a fast and flexible GIS system. By "unified" we mean that we will provide one highly flexible query model and language for Grid information, both inside and outside the formal bounds of the data storage structure. no matter how dynamic.

We envision that a unified relational GIS system will consist of the following components:

- Appropriate extensible schemas and indices for Grid information, including an extensible type hierarchy for data objects, and the definition of a set of core types.
- Support for high update rates and freshness constraints required of any GIS system, including streaming updates.
- Support for sophisticated compositional queries involving joins, including a new primitive, the non-deterministic, time-bounded query.
- Support for data streams as relations, including "SQL queries over data streams" functionality.
- Decentralized administration, multiple administrative domains, and distributed data.
- Canned queries to make the system usable for non-SQL users.

This whitepaper is both a high level roadmap of where a relational approach would take us, and an implementation based on existing software artifacts and systems. Early feedback on the whitepaper has indicated that splitting the paper along these lines might be advantageous.

1 Introduction

The proliferation of bandwidth and computational cycles is driving rapid change in high performance parallel and distributed computing, making possible a shared large scale wide-area computational infrastructure, a concept which has been named the Grid [12, 13]. Simultaneously, traditional applications are growing increasingly ambitious, and new kinds of applications are poised to dramatically expand the user pool of the Grid.

As the scale and diversity of the Grid's resources, applications, and users continues to explode, the amount of information needed to keep track of them grows commensurately and becomes increasingly dynamic. Simultaneously, applications need to pose and answer increasingly powerful queries over this information in order to exploit Grid resources well and satisfy users. We believe that the rapidly rising tension between these opposing trends demands a new way of managing, updating, and querying Grid information.

We intend to address these trends through a unified relational approach to this important directory services problem. Our approach is unified in that we treat the whole spectrum of Grid information, from static to dynamic, in the same way from the perspective of users and managers. Our approach is relational in that we apply the relational data model to Grid information, and that we leverage existing relational database and transaction processing technology to achieve our goals. We believe that this approach holds the key to meeting the following fundamental requirements for a Grid information service:

- Efficient and scalable support for a large and a rapidly growing number of data objects,
- Elegant representation of complex evolving relationships between data objects,
- High performance support for frequent and complex updates to data objects, their attributes, and their relationships, including for objects that may belong to different administrative domains,
- High performance support for complex and evolving queries on data objects and their relationships, in particular for compositional queries, and
- High performance, integrated support for data streams.

Although our focus is on Grid information, we believe that work in this area will generalize beyond the Grid as other directory services problems face similar trends. We also believe that our approach is complementary to the LDAP approach also under consideration in the Grid Information Systems and Grid Performance working groups. Although we intend to develop our approach independently, we are quite willing to explore ways in which a relational GIS system can interoperate with more traditional directory service models.

2 Grid information as a directory service

"The Grid" was envisioned several years ago as a global compute platform for large-scale applications where typical barriers to using thousands of resources simultaneously, such as administrative domain difficulties, are lowered by community consensus and support for Grid technologies based on open standards [10]. Much of the work of Grid-based computing is done by researchers tied to the Global Grid Forum [13] (Global GF), a community-initiated forum of individual researchers and practitioners. Analogous with the Internet/internet convention, "a grid" (note lowercase) refers to a distributed system that uses Grid standards while not necessarily being a part of "the Grid." Also, we will often refer to "computational grids" or "grid-based computing", by which we mean grids used for computational purposes, as opposed to for other purposes such as data access. This is not intended to be exclusionary, but rather to focus discussion on the kind of grid that we know best. Grid information services are a necessary component of other kinds of grids and grid-based processes.

Grids are necessarily awash in information. Applications must be able to locate data sources and suitable computational nodes on which to run. Schedulers require information about utilization and load in order to make their decisions. The owners of some resources need supply and demand information in order to set prices. Grid information occupies a spectrum: some information is relatively static (host configurations, for example), while other information is quite dynamic (host load, for example). We refer to an item of Grid information as a data object, or object. We intensionally use this general term to avoid too rapidly binding to a specific model of information representation.

A Grid Information Service (GIS) is a directory service that consists of a set of objects, relationships between objects, and systems needed to query and update the objects and relationships. An object describes an entity in the Grid. It has a unique identifier, possibly a timestamp, and a set of attributes. The relationships between objects vary depending on the data model used to structure the objects. A hierarchical (tree) or object-oriented data model makes extensive use of the parent-child ("is-a") relationship. For example, an organization and a person working for the organization are related via a parent-child relationship. The flat-table organization of the relational database management lends itself to stronger reliance on the peer-to-peer ("part-of") relationship between tables. For example, two computers and the network path that connects them are related via a peer-to-peer relationship. An update to the attributes of an object causes the irrecoverable replacement of the existing attribute values.

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

1. The entity can be described succinctly.

2. The entity can be uniquely described by a set of attributes.

Grid entity	Description
organizations	accountable bodies and owners of resources
people	resource administrators, resource providers, GIS administrators
compute resources	memory, CPU information, cache, benchmark results, heart-beat, boot time, number of users, load
communication resources	link capacity, switch capacity, error rate, drop rate
software packages	BLAS, LAPACK, etc.
event producers	generators of event streams
event channels	propagators of event streams
event dictionaries	databases of commonly used event types
instruments	radar systems, telescopes, etc
network paths	available bandwidth and expected latency
network topologies	hosts, switches, routers
wireless devices	wireless hosts, wavepoints, cells, etc.
virtual organizations	groups of collaborators
...	...

Figure 1: Non-exhaustive list of Grid entities with data objects in GIS.

3. The entity is long lasting.
4. The entity has value to the broader computational grid community.

The first criterion assures the object is useful, that it actually captures key attributes about an entity in the computational grid, that a user of the GIS can figure out what it is. The second criterion assures that the user can distinguish between entities with similar properties. In combination, these criteria assure that a user can intelligently select one entity or group of entities over another. By the third criterion, we mean that we only capture entities that persist beyond the lifetime of any one application. It is important to note, however, that the properties of an object may change very frequently. For example, a host is an entity for which we would include an object in the GIS. However, the "load" property of that object may change very frequently. The final criterion ensures that the entity represented in the information service provides value to more than one application. 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.

Although the criteria described above seem oriented to defining exclusion, they more importantly define inclusion as well. Any describable, unique entity that has lasting value to the broader computational grid community should be represented in the GIS. This inclusion is irrespective of qualities like frequent update rate that might hinder performance of some implementations. Unfortunately, the limitations of the directory services systems upon which current GIS systems are based make it difficult to achieve this ideal.

3 The needs of Grid applications

Although there are many consumers of Grid information, it is parallel and distributed applications and the software systems that map them to the Grid that make the most demands of the GIS. In the following, we'll talk about these demands in the context of traditional parallel and distributed scientific and business applications, non-traditional interactive applications, and other Grid software systems such as schedulers and prediction services. In effect, the following presents a series of use-cases that are ill-served by existing GIS systems and for which we believe a relational GIS system is more appropriate.

3.1 Traditional parallel applications

Grid-based computing allows scientists to attack far larger problems than could be addressed on traditional supercomputers or clusters. Their applications must exploit parallelism, partitioning their tasks or data and mapping them to Grid resources. Deciding how to do so requires detailed information about these resources, their connectivity, and their load.

Consider even 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 in a dedicated cluster. 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 say 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 want to say things such as "tell me when the load on any one of these four hosts is at least 25% different from the average across all the machines" so that it can trigger a load balancing step when this occurs.

There are three things to notice from this example. First, the application's queries compose information about multiple resources (data objects in the GIS) in unique, application-specific ways. Second, the data objects on which the queries depend are dynamic to different degrees. Host memory changes more slowly than upgrades to network paths, while host load fluctuates far more quickly than either. The implication is that some data objects in the GIS will be updated quite frequently. Finally, this application needs to query information streams (*e.g.*, for host load values) in addition to information in a database. None of these requirements interacts well with the directory service systems upon which current GIS systems are based.

3.2 Traditional distributed applications

At first glance, it may appear traditional distributed applications (those based on distributed transactions, RPC, and/or virtual synchrony) place less stringent demands on a GIS system. It might appear, for example, that without parallelism, the need for compositional queries is ameliorated. However, this is not the case—in fact, traditional distributed applications also place stringent demands on GIS.

In the same manner as a task-parallel parallel application, a workflow-style distributed application needs to create a virtual datapath for requests on top of the grid's resources. This induces a mapping process very similar to what we discussed above. 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." Of course, the application must be able to predict its performance on each of the processors, a non-trivial problem to be sure. A detailed compositional query is not sufficient for success, but it is necessary.

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.

One such application is CMU's Dv [2, 18], a framework for constructing interactive scientific visualizations for wide-area environments. The Dv programmer can trivially (almost mechanically) transform a sequential C++ vtk (vtk [28] is an extremely popular toolkit for building visualizations) program into a Dv program. The logical view of a Dv program is as a flowgraph. 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 knows how to assign 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.

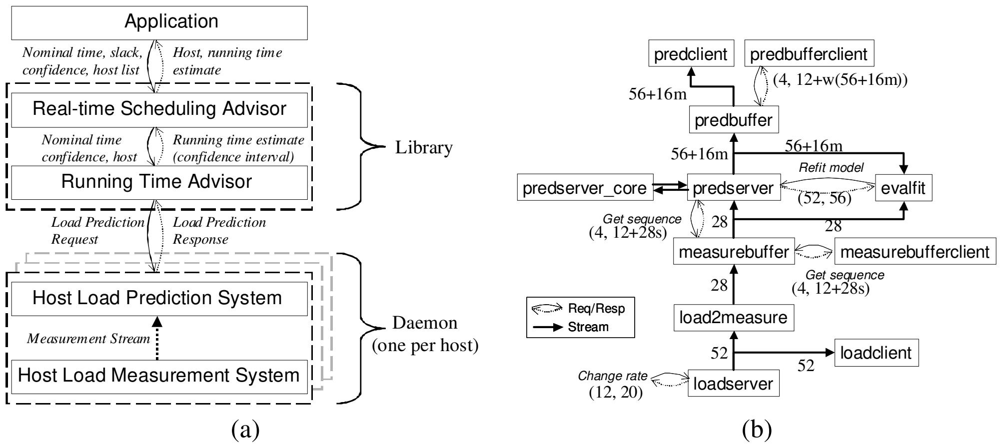


Figure 2: RPS system: (a) conceptual view, (b) constellation of components that must be managed.

Many kinds of schedulers are possible in Dv. Indeed, the framework was designed to make it easy to evaluate different schedulers. With a static scheduler, one that makes all decisions at instantiation time, the GIS needs are similar to those of the workflow-style applications discussed above. With a highly dynamic scheduler, one that reevaluates the schedule on each hop of the active frame, stream-oriented information on multiple hosts, much like in the load-balancing data-parallel program described above, is needed. We have also explored predictive real-time schedulers, which require that we be able to find prediction pipelines, as described below.

Beyond visualization, the global connectivity of grid-based computing has made possible "distributed laboratories" [24] where scientists collaborate by joint control of scientific instruments, such as computer models, used during investigation. Achieving real-time behavior on computational grids is even more critical when physical instruments are involved, and it induces the same requirements for GIS systems.

3.4 Other Grid software systems

In addition to querying them directly, applications also rely on GIS systems indirectly, making use of other Grid software that uses GIS. We give two important examples here: systems that provide a view on highly dynamic Grid information that is not within the purview of GIS, and schedulers that map applications to Grid resources, both within individual sites and across them.

Some important grid information does not fall within the model we introduced in Section 2, either because it is too ephemeral or is of interest only to a small set of applications at any one time. Furthermore, some information is exceedingly fine grain and thus very difficult to incorporate into any GIS system. A number of systems have been proposed or developed to provide this information, including NWS [35], RPS [8], and Remos [19]. Currently, the Grid Performance Working Group [15] is attempting to create standards so that such systems can interoperate.

While the data generated by these systems may not be in the domain of the GIS, the structure and location(s) of their instances often is. For example, RPS is a system that provides time-series predictions of fine-grain measurement streams. While RPS is conceptually quite simple (See Figure 2(a)), 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 2(b). Each component (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 (for example) "run the (expensive) predictor on *that* machine."

With this flexibility comes the need to manage these constellations of components and their relationships. This task is clearly within the purview of a GIS system. RPS is not alone in its needs. Remos and NWS 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. At least one Grid scheduling researcher has taken it upon herself to develop a system, GridSearcher, that can answer a small subset of such queries by basically providing extra database functionality over an existing GIS system that provides no such functionality [27].

3.5 Resource owners

The owner of a supercomputer, fast network path, software library, dataset, or other resource must be able to advertise the resource to potential consumers. This is true regardless of the economic model (or lack thereof) placed on Grid resources. Furthermore, he must be able to quickly update the description of his resource, regardless of administrative domain. Indeed, the more accurate and up-to-date the description of the resource, the more likely the resource is to be used by others. Conversely, the better the picture one has of competing resources, the more able one is to quickly adjust to changing market realities. Finally, the resource owner must be able to control access to and even visibility of the resource depending on the prospective user.

4 Limitations of current directory services

As current GIS systems [9] and proposals [14] are layered on top of general purpose directory services systems, it is important for the reader to understand the limitations of these systems and how they affect what a GIS system can do for applications and other consumers of Grid information.

Traditionally, directories were employed to manage descriptive information about relatively static objects, ranging from hostname/IP-address mappings to the office locations and telephone numbers of company employees. The data in directory services today is frequently accessed but rarely updated. The storage model is typically hierarchical, that is, data objects are related to their "parent" objects. A path exists from the root object to each leaf object. We again intensionally use the general term "data object" to avoid too rapidly binding to a specific model of information. A data object in a hierarchical model is a relative path, beginning with possibly a non-root node, and a terminating node. In an object-oriented model, it is an object accessed via some path. In a relational model, it is a row of a table. Commonly used wide-area directory standards include the X.500 Directory Service [32], Lightweight Directory Access Protocol [23], Internet's Domain Name System(DNS) [3] and DEC's Global Name Service (GNS) [17]. More recent proposals include Intentional Naming Systems [1] and Active Names [33].

As Grid resources and applications evolve, they are placing increasing demands on directory services, demands that current directory services are not equipped to handle. These demands take the following five forms.

1. Rapidly growing numbers of data objects.

2. Increasingly complex relationships among data objects.

3. Increasingly frequent updates to objects and their relationships (i.e., more "dynamic objects".)

4. Rapidly escalating demand for sophisticated queries, especially over related objects.

5. Growing interest in querying data streams.

Given the inclusion criteria for Grid entities that we laid out in Section 2 and the rapid growth of interest in Grid computing, it is clear that the number of data objects is exploding. Furthermore, the success of systems like Globus [11], as well as ad hoc large scale computational efforts such as DESCHALL [7] and SETI@HOME [30] suggest that the Grid (and thus the number of objects in the GIS) is poised for even more rapid expansion. Economics are also beginning to come into play, for example, in such systems as Legion [16]. The efficiency of a computational economy, regardless of whether follows a traditional model [34, 5] or the currently-in-vogue "gift economy" model [26] is strongly dependent on the quality and quantity of information that the GIS system can offer. Traditional directory services are not designed to support such large datasets. Perhaps the largest of these systems, DNS, contains only about a hundred million host objects, each with one attribute (their IP address). By our criteria, a GIS should support queries over many more attributes of each host, as well as over the network paths that connect them.

As the number of objects stored in the GIS grows, the relationships between them also grow in complexity, and new relationships are found to have value and therefore must be represented. For example, it has been shown that link-level information about networks, which can be collected scalably and efficiently, can be composed into good estimates of path-level characteristics that themselves can not be collected scalably or efficiently [20]. This introduces a new, extremely important relationship that links objects in the GIS. Other such relationships may appear as research continues. Unfortunately, most directory services support only the hierarchical model, which means that the GIS designer must decide which relationships are important early and encode them as an explicit link in the data model. Woe to the GIS designer who is not prescient, because modifying a schema after-the-fact can be broadly disruptive.

Updates to Grid data objects are becoming increasingly frequent both because there are simply more data objects (and relationships) and because more and more dynamic kinds of data are being introduced. The update frequency of data objects must be determined by the needs of applications—within the limits of what the directory service can support. We anticipate that frequently updated, or dynamic objects, will eventually form the majority of objects a GIS system handles. Unfortunately, the existing directory services (i.e., LDAP v3 [23]) are well known to perform poorly in the presence of frequent updates.

A related problem with using existing directory services as the basis of GIS systems is their limited and restrictive query language: the language is procedural, lacks sophisticated processing semantics, and requires that the user have explicit knowledge of the tree structure. First, a procedural query language embodies an execution order in the query, whereas a declarative query language, such as SQL, states *what* is to be retrieved but leaves the *how* up to the query engine. Thus significant gains in query efficiency via query optimization can be realized by the latter that are denied to the former. Second, users of the GIS will increasingly require more sophisticated queries that extract data from over the entire data domain. Hierarchical query languages lack the ability to do such sophisticated processing on the data. Key are the need to join tables on a user specified condition, and perform an aggregate computation over the results. Placing the onus of processing complex query results (i.e., a join) on a user seems to be an unnecessary throwback to an earlier time. Finally, the user must have explicit knowledge of the structure of the tree. The syntax of an LDAP-style query language requires that the user state the starting, or top-most, node of a query's domain (*i.e.* search space). The attributes appearing in the query must stated in terms of a pathname relative to that starting node. Writing accurate queries becomes difficult, and changes to the tree topology can have a broad and costly impact on users.

Existing directory services also do not provide support for data streams, a type of data that we expect will become increasingly common as the Grid evolves. For example, as networks improve, Grid schedulers will probably want to schedule smaller and smaller units of work. To decide where to place that work, they will make use of increasingly dynamic information, including such information as prediction streams for RPS-like systems. Why should the scheduler developer have to deal with an entirely different data model and query model for such stream data? We believe that data streams should be unified with other Grid information to the greatest extent possible when the meet the inclusion criteria we outlined previously.

5 Proposed approach

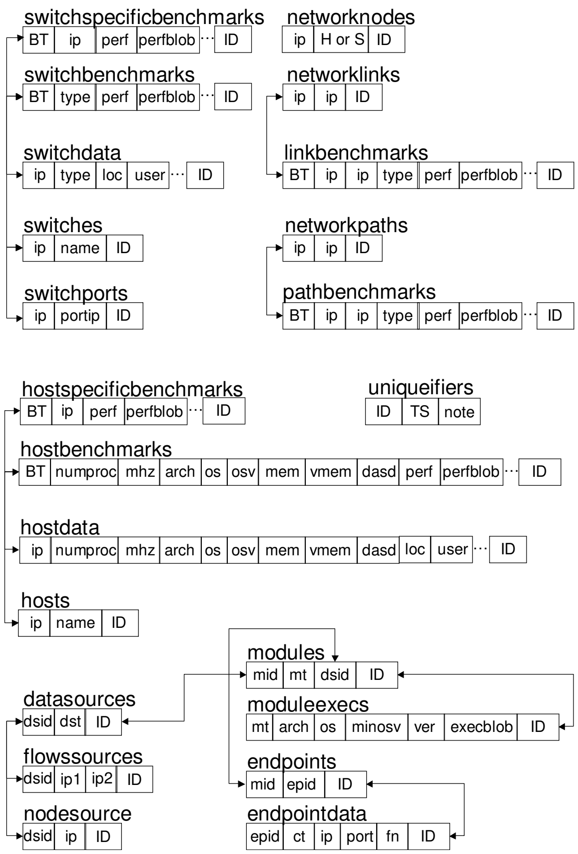
At the highest level of abstraction, our plan is to investigate and build Grid information services that meet the application needs and avoid the limitations that we laid out earlier. We intend to do so by using a different starting point. Instead of starting with commodity directory services and "working our way up" to a GIS system, we will start with a full-strength commodity relational database system and "work our way down" to a practical GIS system that can better meet the needs of applications than extant systems and proposals. We believe that doing so, while challenging, is entirely feasible and will furthermore lead to research results that will generalize beyond GIS.

5.1 Prototype system

We will begin by implementing a proof-of-concept GIS system that is based on a scalable commodity relational database management system (RDBMS) that supports transactions and distribution of data. At this point, we believe this will likely be Oracle. However, neither our design or our research are specific to the underlying RDBMS system. The system we build will provide the basis for trying out our research results in practice.

By building on a powerful database system, we will be starting on the traditional database high ground, only shunning features when absolutely necessary. In particular, we shall start with the following features.

- Full ACID properties.
    - Atomicity: Updates are all-or-nothing (transactions).
    - Consistency: The integrity of data and relationships is always preserved. The database is transformed from one valid state to another.



(a)

```
select
   ed.protocol, h.name, ed.port, m.name
from
   host as h, module as m,
   endpoints as e, endpointdata as ed
where
   h.name = "kanga" and
   ed.datatype = LOAD_MEASUREMENT and
   h.ip = m.ip and
   m.mid = e.mid and e.epid = ed.epid
```

(b)

```
select nondeterministically
   host1.name, host2.name, host3.name, host4.name,
   hd1.mem+hd2.mem+hd3.mem+hd4.mem as TotalMem,
from
   hosts as host1, hostdata as hd1,
   hosts as host2, hostdata as hd2,
   hosts as host3, hostdata as hd3,
   hosts as host4, hostdata as hd4
where
   host1.ip=hd1.ip and  host2.ip=hd2.ip and
      host3.ip=hd3.ip and host4.ip=hd4.ip and
   hd1.mem+hd2.mem+hd3.mem+hd4.mem>=512 and
   hd1.mem+hd2.mem+hd3.mem+hd4.mem<=1024 and
   host1.ip!=host2.ip and host1.ip!=host3.ip and
      host1.ip!=host4.ip and host2.ip!=host3.ip and
      host2.ip!=host4.ip and host3.ip!=host4.ip
order by
   TotalMem desc
limit
   1
inlessthan
   5 seconds
usingheuristic
   prefer_depth_first
```

(c)

Figure 3: Schema and queries: (a) a schema for hosts, switches, network properties, and RPS-like software, (b) a simple deterministic query that finds all sources of load information on a particular host, and (c) a time-bounded non-deterministic query using a heuristic.

  - Isolation: Queries and updates see a consistent picture of the database, independent of other queries and updates.
  - Durability: Committed transactions are permanent.
- Full SQL query support: Queries are written using the expressiveness of the full relational calculus (arbitrary set operations, to a first approximation).
- Efficient query optimization and evaluation: queries are planned and evaluated as efficiently as possible.
- Scalability to extremely large transaction processing environments.
- Support for distributed databases.

In support of the last point, we plan to replicate the prototype system at least once. At the present time, we have a small Oracle system described in the following section and in the appendix.

5.2 A schema and indices for Grid information

The focus of this part of our work will be to work out a suitable schema for representing computational grid data, as well as a reasonable set of indices to make common queries fast. Designing the schema will require thoughtful consideration to current Grid resources and how they are evolving. Fortunately, as we are both members of the Grid Forum, we are well poised to answer the questions that arise here.

A tentative schema is presented in Figure 3(a). The appendix provides a detailed description of this schema, as well as of the system and RDBMS software on which it is currently implemented. This schema was initially developed to support the management of RPS components for host load and network bandwidth prediction. We are not wedded to this schema. We present it to make specific our discussion and to provide the basis for examples. The schema structures information about hosts, network switches, as well as software systems composed out of RPS components. This schema is, at this point, only a design study. Figure 3(b) shows an example query on the schema. The query is trying to find all sources of load information for the host kanga. This is an example of what we call *deterministic query*. Figure 3(c) shows a *time-bounded non-deterministic query*, one of focuses of our approach. We will discuss this further in a later section, but we note here that it is very important to our goal of making it practical to use an RDBMS as the basis for a GIS system.

With the schema in place, we plan to populate the database using data available from Globus MDS, NWS, RPS, Remos, and other sources. We also plan to examine the queries and updates typically provided to these systems, as well as to test the demand for more sophisticated queries in the Grid community. The goal is to develop a set of benchmark queries and updates, or, ideally, a workload generator for the system. Based on the benchmarks or the workload generator, we will develop a set of appropriate indices for the schema. Depending on the nature of queries, we may also develop a system that automatically chooses appropriate indices to generate given different constraints on resource, etc. Once we have the baseline RDBMS-based GIS system, schema, data, and indices, we will investigate the scalability of the system, comparing it to the more common LDAP-based systems and proposals. We envision four levels of scaling: lab, department, school, and multi-site.

5.3 Type hierarchy

The relational data model provides only primitive data types (column types) such as integers, strings, and blobs. More complex types are produced by creating relations between these types, either implicitly (rows) or explicitly (via set operations). While this lets us build arbitrarily complex types (and provides a mechanism for type extensibility—we can add new columns to tables), the types are all implicit. We intend to provide a mechanism for making types explicit, even for primitive data. The tentative schema discussed above already has some support for this, in that rows or columns have explicit type tags.

Defining the types and their relationships, as well as providing a community-based process for extending types is one aspect of our plan that overlaps considerably with the Grid Information Services and Grid Performance working groups. At this point, we believe that we can use and help define the hierarchical types being defined in these groups.

5.4 Dynamic objects

As noted earlier, a principle challenge to GIS systems is the rapid growth in the number and frequency of updates to objects in the database. We plan to empirically study supportable update rates for the relational implementation compared to a hierarchical implementation. We also plan to study how an RDBMS-based GIS system can support higher update rates. At this point, we believe that many updates will come in the form of streams, and that we ought to be able to exploit this stream nature to achieve higher update rates. How this approach might interact with transactions is an open question.

Updating the GIS is the process of changing the attributes of one or more objects. In the remainder of this section, we shall limit our discussion to updates to a single object. It is important to note, however, that atomic update to multiple objects demands transactional properties; these complex update transactions have the potential to drastically slow the system. We intend to expand the investigation described below to explore the tradeoffs between transactions and update rates. We categorize the entities represented in the GIS according to the how frequently their attributes change and according to the timeliness properties needed of their attribute data. At this point, we believe there are essentially two categories: stable grid entities, and dynamic grid entities.

Stable grid entities are major resources, such as compute engines, organizations, etc. These entities are very long-lived, changes to their attributes are infrequent. and changes in their attributes can be propagated relatively slowly. For instance, additional memory is added to a supercomputer quite infrequently. When memory is added, that information need not be propagated to consumers as rapidly as other information for scheduling based on available memory is likely to occur only over relatively long timescales. Updates to the objects of stable entities and propagation of those updates in a distributed database system can be thus be done at a lower priority than updates to objects that change more frequently or require more rapid dissemination of their changes. Also, from a user interface point of view, these updates can probably be accomplished relatively simply via a "canned" or interactive program or script.

Dynamic grid objects, on the other hand, have either more frequent updates or require more rapid dissemination of their changes. As an extreme example, consider a network path: the end-to-end path between two machines clearly meets the criteria for inclusion in the GIS as it is of interest to a broad set of users and the choice of one path over another can have significant performance implications. Relevant path attributes might include current bandwidth and predicted bandwidth, both values could see updates with a frequency matching the entities' ability to generate new values: on the order of milliseconds. Furthermore, a program is not interested in stale updates of network information, even if those updates come at a high frequency. The GIS must assure that updates are rapidly propagated to consumers.

We posit that the total number of highly dynamic objects will far exceed the number of stable objects and the rate of update to these objects must be flexible to meet the needs of the timeliness of the data contained in the attributes. A delay of 12 hours in updating the availability of LAPACK on a machine is far less a sin than is a 15 min. delay in updating the current bandwidth of a network path. Based on this supposition, we maintain the following.

- The updating mechanisms supported by GIS must go beyond hand-coded programs or scripts.
- Because of the projected number of dynamic objects, updates to objects must be pushed to the GIS, not pulled by GIS.
- The GIS must be prepared to push updates to consumers, or at least provide trigger conditions.
- The GIS must impose limits on the rate at which data is pushed to it. Too frequent updates will result in stale data being provided to applications.
- Processing to filter update messages to reduce update frequency must be done upstream of the GIS.
- The GIS could benefit from the CORBA-like event stream model proposed by the Grid Performance Working. The publish/subscribe model inherent in event channels could be used to allow GIS to selectively subscribe to the event channels of producers publishing update messages.

With the schema in place, we will explore how to support dynamic objects well. We intend to begin by explaining and quantifying the tension between the user's need for timely data and the realistic update rates that a GIS can handle. We plan to study this tension not only in the context of our RDBMS-based system, but also in the context of a traditional LDAP-based system. This is an important comparison as directory services of the future need to support higher update rates and though LDAP is known to suffer poor performance under frequent updates [29] and it has been shown in [4] that a relational database can be tuned to attain very good performance at extremely high update rates, the apples-to-apples comparison will provide invaluable insight to the Grid community.

We will extend the services provided by our prototype system to include support for updating dynamic objects. We envision a service based on data streams implemented on top of an event-channel communication infrastructure. We anticipate that some changes to the schema language (how to describe a schema such as that of Figure 3(a) to a database system) will be required to express the update frequency of dynamic objects. We will test our work by scaling our prototype database to realistic grid computing workloads to determine realistic upper bounds on input processing.

5.5 Time-bounded non-deterministic queries with optional heuristics

The nature of GIS queries will be somewhat different from that of more commonplace queries in relational database systems. In particular, we expect that GIS queries will require joins over numerous tables at distributed locations. This has the potential for introducing serious performance problems, both for particular queries and also for the GIS as a whole. However, the nature of the answers that GIS queriers require is also different.

Our plan is to exploit these different properties to achieve efficiency both from the point of view of the system and the user. There will be frequent occasions when a GIS user will be more interested in a timely response than in seeing all the results from a query. Similarly, there will be many times when the GIS system is overburdened and can not permit a query to continue, but would prefer not to dispose of the results it has computed thus far. To capture these user constraints and system limits, we plan to extend the query model. Our extension is quite simple: we will permit the database to respond to a query non-deterministically in exchange for a deadline on the search process.

A typical GIS user will probably be less interested in getting all the answers from a query than in getting an answer within a limited time. The problem is how to express these constraints. We plan to extend the SQL query model and select statement to do so. First, we will allow the user to state that a query is non-deterministic. This means that the database is permitted to return any subset of the full query results. The second extension is to introduce a time bound, an upper limit on how much time the database should work in constructing its non-deterministic answer. The final extension is to allow the user to specify an optional search heuristic. The search heuristic and non-determinism are not intended to be at loggerheads—the heuristic may only guide search, the system may still return different results each time a query is issued with the same heuristic. Figure 3(c) shows an example of a query that uses all of these extensions. We will also permit the system to return partial results for a deterministic query, as long as it makes it clear to the user that his deterministic query has been downgraded to a time-bounded non-deterministic query.

The result of a time-bounded non-deterministic query is a non-deterministic subset of the full query results that would have been returned had the query been allowed to run its duration. The subset could be a random sample, or it could reflect preferences provided by the user. The declarative nature of SQL is an important enabler for this research in that it describes what is wanted, and not how to find it. We believe that time-bounded non-deterministic queries can be implemented as heuristically-guided stochastic search on top of an existing database system. While work has been done in the database community on returning partial results, particularly in extending SQL to enable query writers to explicitly limit the cardinality of a result [6], to "get some answers quickly, and perhaps more later" [31], and to do statistical random sampling of databases [22], we are sure that by focusing on the specific needs of GIS systems and users, we can produce specific solutions that provide better performance and more flexibility with less overhead.

It is important to note that we are not promising a solution applicable to database query processing in general, but rather to specific query problems that arise in the context of applying database technology to the specific application of a GIS system. In addition to the need for time-bounded queries, query processing for GIS systems differs radically from typical databases in the query cost metric used to measure the performance of the query processing. Whereas in a typical database, the cost metric is to minimize the total cost (CPU and I/O cost), in a GIS system we further must minimize or bound the bandwidth consumed, and the response time to the user.

We believe that these extensions can be done without modification to the underlying RDBMS system, as many of these systems already provide support for randomness. We envision a re-writing stage in which queries in our extended SQL will be transformed into (perhaps much larger) queries in ANSI SQL.

5.6 Unified query model

There are times when information is needed from entities that either fail to meet the criteria for inclusion in a GIS system, or else whose inclusion appears impractical. Data streams, a prominent example, are a commonly occurring grid entity that does not meet the strict criteria, but has high value to a small set of users. A data stream is an event flow from one or more suppliers, usually a single consumer, and possibly intermediate computation to transform, aggregate, or filter data. Underneath a data stream are suppliers, consumers, and event channels as intervening objects, using the CORBA event channel terminology. Data streams typically exist for the duration of a particular application, and support visualization (Dv active frames form such a stream), performance measurement (RPS measurements and predictions form such streams), or general event-based communication [21].

The grid community, particularly the Grid Performance Working Group, is working on schemes for extract information from event streams. These schemes are conceptually quite different from those used to extracting information from a GIS system. We propose a simpler, unified model that views data streams as data objects that happen not to be resident in the GIS. These objects can then be queried using SQL just as other objects in the GIS. We plan to accomplish this by viewing a data stream as a set of relations, where an event is a tuple belonging to one of the relations. This work leverages off the dQUOB system [25], wherein an SQL query is evaluated purely over data streams instead of tables in a database. Queries can be managed remotely, inserted into a stream at runtime, and optimized at runtime in response to information gathered about the data and from the environment.

We will design a unified relational model objects inside the GIS and data streams outside the GIS, and implement the model within our GIS system. These dynamic queries over data streams will also provide support for non-deterministic, time-bounded queries discussed above.

It will be possible to associate computation with each query to refine or transform the data. In this way, a query can create a "view", that is, a new type of relation that can serve as input to another query, thus allowing multiple and successively refined views on the data. The beauty of the unified model comes in the enhanced ties between views on data objects in the GIS and data streams outside the GIS. Since an event channel is defined as a relation, a data stream has a description of the composite of relations, hence has a natural description in relational terms. If it meets the other criteria for inclusion, it should itself be an entity with object representation in the GIS.

On the flip side, an SQL query could be written over a combination of GIS relations and data stream relations, since the interface to the two is the same. It is not the goal of this whitepaper to undertake the question of supporting queries simultaneously requesting information from a database and data stream, but we do intend to explore the implications of representing data streams in the GIS.

5.7 Decentralized administration

One clear benefit to the hierarchical data model used in current and proposed GIS systems is that it naturally decomposes across administrative boundaries. Indeed, LDAP even has the notion of delegating a query to a server responsible for a particular subtree. RDBMS systems are not designed with this kind of decentralized administration, or even the notion of multiple administrative domains in mind. We acknowledge that this is an issue that requires consideration. We are only beginning to consider it.

5.8 Interoperability

Our approach envisions a unified relational GIS system as a relatively stand-alone system, providing both high speed updates and sophisticated compositional queries to all users. However, we fully expect that this will not be the only mode of operation. We envision two other ways in which the system could be used. The first is as an indexing node within the context of the GIS working group's evolving standard. The second is a value-add service, which would collect data from other GIS and performance monitoring systems and then provide more powerful query service on that data.

6 Conclusion

The relational Grid information services task group is in the early stages of developing a system and a standard for a unified relational GIS. We believe that the approach we have outlined here is a powerful and productive one.

This first white paper of the task group outlines an approach to Grid information services (GIS) that is based on the relational data model. While conceived as operating stand-alone, the system proposed herein could also be viewed as a complementary system, or as a set of services that would complement and interoperate with systems built according to the standards emerging from the Grid Information Services working group and the Grid Performance working group.

We are interested in expanding task group membership and welcome ideas and discussion. For more information, please check out our web page: http://www.cs.nwu.edu/∼pdinda/relational-gis.

Appendix A: Discussion of schema

Figure 3 presented our tentative schema in graphical form. In this appendix, we show how a database can be created from the schema using the SQL language. The specific target database here is Oracle, although, to the best of our knowledge, these SQL calls should work on any database that supports ANSI-standard SQL.

Perl scripts for creating and performing simple queries against this database are available from our web page (http://www.cs.nwu.edu/∼pdinda/relational-gis). It is important to note that this is a rapidly evolving schema, and thus will change soon. In addition, it currently has only minimal integrity constraints and other sanity checks. Tables with similarly or identically named columns are implicitly associated.

In this schema, each row of each table is a unique data object with a time stamp noting its insertion. The uniqueifiers table keeps track of the GUIDs, and maps them to their timestamp and a short note:

```
CREATE TABLE uniqueifiers
(
   id   NUMBER(12) NOT NULL PRIMARY KEY,
   ts   CHAR(15),
   note VARCHAR2(255)
);
```

The network topology is represented by three tables, consisting of nodes, links, and paths.

```
CREATE TABLE networknodes
(
   ip   NUMBER(12),
   hors CHAR(1),
   id   NUMBER(12) NOT NULL PRIMARY KEY
```

```
);

CREATE TABLE networklinks
(
   ip1  NUMBER(12),
   ip2  NUMBER(12),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE networkpaths
(
   ip1 NUMBER(12),
   ip2 NUMBER(12),
   id  NUMBER(12) NOT NULL PRIMARY KEY
);
```

In the above, a network node identifies whether it is a host or a switch using the "hors" field.

Hosts are described by two tables:

```
CREATE TABLE hosts
(
   ip   NUMBER(12),
   name VARCHAR(256),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE hostdata
(
   ip      NUMBER(12),
   numproc NUMBER(10),
   mhz     NUMBER(10),
   arch    CHAR(16),
   os      CHAR(16),
   osv     NUMBER(10),
   mem     NUMBER(10),
   vmem    NUMBER(10),
   dasd    NUMBER(10),
   loc     VARCHAR(256),
   usr     VARCHAR(256),
   id      NUMBER(12) NOT NULL PRIMARY KEY
);
```

The hosts table contains a minimal set of information about the host, merely its name and IP address. The hostdata table contains an extensible set of information about the host, such as its computational capacity, disk storage, versions of different software, its location, and its owner. The IP address of the host associates it with its network node entry and with data about the host.

Network switches are described by three tables.

```
CREATE TABLE switches
(
   ip   NUMBER(12),a
   name VARCHAR2(255),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);
```

```
CREATE TABLE switchports
(
   ip     NUMBER(12),
   portip NUMBER(10),
   id     NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE switchdata
(
   ip   NUMBER(12),
   type NUMBER(10),
   loc  VARCHAR2(255),
   usr  VARCHAR2(255),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);
```

Like a host, a switch is considered to have a distinguished ip address, but it may also have several other ip addresses, one for each of its ports. The distinguished ip address of a switch associates it with its network node entry, with its other ports, and with its entry in the switchdata table, which provides more information about the switch, such as its type and location.

A particular kind of switch or host may have certain performance characteristics, or benchmark information that could be shared. The switchbenchmarks and hostbenchmarks tables associate the results of such benchmarks with the hardware and software used:

```
CREATE TABLE switchbenchmarks
(
   bt       NUMBER(10),
   type     NUMBER(10),
   perf     NUMBER(10),
   perfblob BLOB,
   id       NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE hostbenchmarks
(
   bt       NUMBER(10),
   numproc  NUMBER(10),
   mhz      NUMBER(10),
   arch     CHAR(16),
   os       CHAR(16),
   osv      NUMBER(10),
   mem      NUMBER(10),
   vmem     NUMBER(10),
   dasd     NUMBER(10),
   perf     NUMBER(10),
   perfblob BLOB,
   id       NUMBER(12) NOT NULL PRIMARY KEY
);
```

In the above, “bt” refers to a benchmark type, which is some type within a future standardized hierarchy of benchmarks. Furthermore, a benchmark must collapse its performance measurement into a floating point number, perf. It may also provide more detailed information within an opaque binary large object.

In addition, *specific* hosts and switches may have been benchmarked at certain points in time:

```
CREATE TABLE hostspecificbenchmarks
```

```
(
   bt       NUMBER(10),
   ip       NUMBER(12),
   perf     NUMBER(10),
   perfblob BLOB,
   id       NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE switchspecificbenchmarks
(
  bt       NUMBER(10),
  ip       NUMBER(12),
  perf     NUMBER(10),
  perfblob BLOB,
  id       NUMBER(12) NOT NULL PRIMARY KEY
);
```

It is important to note that both the mapping from hardware type to benchmark and from specific node to benchmark is many-to-many.

Analogously, there are also benchmarks for specific links and paths within the network. These are likely eventually to be dynamic information provided through a streaming interface.

```
CREATE TABLE linkbenchmarks
(
   bt       NUMBER(10),
   ip1      NUMBER(12),
   ip2      NUMBER(12),
   type     NUMBER(10),
   perf     NUMBER(10),
   perfblob BLOB,
   id       NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE pathbenchmarks
(
   bt       NUMBER(10),
   ip1      NUMBER(12),
   ip2      NUMBER(12),
   type     NUMBER(10),
   perf     NUMBER(10),
   perfblob BLOB,
   id       NUMBER(12) NOT NULL PRIMARY KEY
);
```

The remainder of the tables in the schema are designed for managing the configurations of stream-oriented software such as RPS prediction systems.

Streams are sourced from data sources:

```
CREATE TABLE datasources
(
   dsid NUMBER(10),
   dst  NUMBER(10),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);
```

A data source has a type and an identifier. Dsts are types in a future standardized hierachy.

Two current examples of data sources within the context of RPS are flow sources, which are measurements of network bandwidth using Remos, and nodesources, which are measurements of host load using RPS-internal sensors.

```
CREATE TABLE flowsources
(
   dsid NUMBER(10),
   ip1  NUMBER(12),
   ip2  NUMBER(12)
   id   NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE nodesources
(
   dsid NUMBER(10),
   ip   NUMBER(12)
   id   NUMBER(12) NOT NULL PRIMARY KEY
);
```

RPS-like systems consist of components, or modules, that attach to each other in various ways. The modules table provides a list of the currently running modules:

```
CREATE TABLE modules
(
   mid  NUMBER(10),
   mt   NUMBER(10),
   dsid NUMBER(10),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);
```

Each runing module has a unique identifier, a type, and the identifier of the underlying data source on whose data stream it is operating. Again, module types are types with a future standardized type hierarchy. Modules can be found in the moduleexecs table, which has executable blobs of the different module types for different architectures and system software:

```
CREATE TABLE moduleexecs
(
   mt       NUMBER(10),
   arch     CHAR(16),
   os       CHAR(16),
   minosv   NUMBER(10),
   ver      NUMBER(10),
   execblob BLOB,
   id       NUMBER(12) NOT NULL PRIMARY KEY
);
```

Modules interface to the rest of the world through endpoints. Endpoints are listed separately in a table that associates endpoint ids with module ids, and another table that associates endpoint ids with a description of the endpoints.

```
CREATE TABLE endpoints
(
   mid  NUMBER(10),
   epid NUMBER(10)
   id   NUMBER(12) NOT NULL PRIMARY KEY
);

CREATE TABLE endpointdata
```

```
(
   epid NUMBER(10),
   ct   NUMBER(10),
   ip   NUMBER(12),
   port NUMBER(10),
   fn   VARCHAR(256),
   id   NUMBER(12) NOT NULL PRIMARY KEY
);
```

Endpoints may support differnt communication types (udp, tcp), and usually have a port associated with them.

References

[1] ADJIE-WINOTO, W., SCHWARTZ, E., BALAKRISHNAN, H., AND LILLEY, J. The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM Symposium on Operating System Principles* (December 199).

[2] AESCHLIMANN, M., DINDA, P., KALLIVOKAS, L., LOPEZ, J., LOWEKAMP, B., AND O'HALLARON, D. Preliminary report on the design of a framework for distributed visualization. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)* (Las Vegas, NV, June 1999), CSREA Press, pp. 1833–1839.

[3] ALBITZ, P., AND LIU, C. *DNS and BIND*. O'Reilly and Associates, Inc., Sebastopol, California, 1992.

[4] BECLA, J., AND HANUSHEVSKY, A. Creating large scale database servers. In *IEEE Symposium of High Performance Distributed Computing (HPDC)* (August 2000).

[5] BUYYA, R., GIDDY, J., AND ABRAMSON, D. An economy grid archicture for service-oriented distributed computing. In *Proceedings of the 10th IEEE Heterogeneous Computing Workshop* (April 2001). To Appear.

[6] CAREY, M., AND KOSSMANN, D. On saying "enough already!" in sql. In *ACM SIGMOD Conference* (1997).

[7] CURTIN, M., AND DOLSKE, J. A brute force search of des keyspace. *;login:* (May 1998).

[8] DINDA, P. A., AND O'HALLARON, D. R. An extensible toolkit for resource prediction in distributed systems. Tech. Rep. CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[9] FITZGERALD, S., FOSTER, I., KESSELMAN, C., VON LASZEWSKI, G., SMITH, W., AND TUECKE, S. A directory service for configuring high-performance distributed computations. In *Proceedings of the Sixth IEEE Symposium on High Performance Distributed Computing (HPDC '97)* (1997), pp. 365–375.

[10] FOSTER, I., AND CARL KESSELMAN, E. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1999.

[11] FOSTER, I., AND KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications 11*, 2 (1997), 115–128.

[12] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[13] GRID FORUM. Grid forum web site. http://www.gridforum.org.

[14] GRID FORUM. Grid information servies working group. http://www.gridforum.org.

[15] GRID FORUM. Grid performance working group. http://www.gridforum.org.

[16] GRIMSHAW, A. S., AND WULF, W. A. The legion vision of a worldwide virtual computer. *Communications of the ACM 40*, 1 (1997).

[17] LAMPSON, B. W. Designing a global name service. In *4th ACM Symposium on Principles of Distributed Computing* (August 1986).

[18] LOPEZ, J., AND O'HALLARON, D. Runtime support for adaptive heavyweight services. In *Proc. of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000)* (Rochester, NY, 2000), Lecture Notes in Computer Science, Springer-Verlag. to appear.

[19] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource monitoring system for network-aware applications. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 1998), IEEE, pp. 189–196.

[20] LOWEKAMP, B., O'HALLARON, D., AND GROSS, T. Direct queries for discovering network resource properties in a distributed environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC99)* (August 1999), pp. 38–46.

[21] OLESON, V., SCHWAN, K., EISENHAUER, G., PLALE, B., PU, C., AND AMIN, D. Operational information systems - an example from the airline industry. In *Proceedings of the First Workshop on Industrial Experiences with Systems Software (WIESS '00)* (October 2000).

[22] OLKEN, F. *Random Sampling from Databases*. PhD thesis, University of California, Berkeley, 1993.

[23] ORGANIZATION, I. I. T. S. *Understanding LDAP*. IBM Corporation, 1998.

[24] PLALE, B., ELLING, V., EISENHAUER, G., SCHWAN, K., KING, D., AND MARTIN, V. Realizing distributed computational laboratories. *International Journal of Parallel and Distributed Systems and Networks 2*, 3 (1999).

[25] PLALE, B., AND SCHWAN, K. dQUOB: Managing large data flows using dynamic embedded queries. In *IEEE International High Performance Distributed Computing (HPDC)* (August 2000).

[26] RAYMOND, E. S. The cathedral and bazzar. http://www.tuxedo.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/, August 2000. (Revision 1.51).

[27] SCHOPF, J., AND MELODY, S. Gridsearcher. http://anchor.cs.nwu.edu/GridSearcher.

[28] SCHROEDER, W., MARTIN, K., AND LORENSEN, B. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*, second edition ed. Prentice Hall, 1998.

[29] SMITH, W., WAHEEL, A., MEYERS, D., AND YAN, J. An evaluation of alternative designs for a grid information service. In *IEEE Symposium of High Performance Distributed Computing (HPDC)* (August 2000).

[30] SULLIVAN, W. T., WERTHIMER, D., BOWYER, S., COBB, J., GEDYE, D., AND ANDERSON, D. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the Fifth International Conference on Bioastronomy* (1997), C. Cosmovici, S. Bowyer, and D. Werthimer, Eds., no. 161 in IAU Colloquim, Editrice Compositori, Bologna, Italy.

[31] TAN, K.-L., GOH, C. H., AND OOI, B. C. Query rewriting for SWIFT (first) answers. *IEEE Transactions on Knowledge and Data Engineering 12*, 5 (Sept/Oct 2000).

[32] UNION, I. T. Information technology – open systems interconnection – the directory: Overview of concepts, models, and services, August 1997.

[33] VAHDAT, A., DAHLIN, M., ANDERSON, T., AND AGGARWAL, A. Active names: flexible location and transport of wide-area resources. In *USENIX Symposium on Internet Technology and Systems* (October 1999).

[34] WALDSPURGER, C. A., HOGG, T., HUBERMAN, B. A., KEPHART, J. O., AND STORNETTA, W. S. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering 18*, 2 (February 1992), 103–117.

[35] WOLSKI, R. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)* (August 1997), pp. 316–325. extended version available as UCSD Technical Report TR-CS96-494.