Representing Dynamic Performance Information in Grid Environments with the Network Weather Service*

Martin Swany and Rich Wolski
Department of Computer Science
University of California
Santa Barbara, CA 93106
{*swany,rich*}*@cs.ucsb.edu*

Abstract

In this paper, we discuss requirements for integrating dynamic performance information from the Network Weather Service (NWS) into the Grid Information Service infrastructure (GIS). We describe the object model that NWS uses internally and provide some rationale for its structure. Finally, we present the NWS's implementation of a caching LDAP daemon that integrates NWS information into the reference GIS – the Globus MDS.

1 Introduction

The goal of Computational Grid systems is to provide the infrastructure necessary so that applications and their users can aggregate resources dynamically [8]. Grid applications should be able to draw computational "power" from a pool of "generators" much in the same way that appliances draw electrical power from a set of cooperating and interconnected power utilities. To realize this metaphor, the Computational Grid infrastructure must include a set of robust, high-performance, and cooperating information services. Grid users and application-schedulers must be able to discover *what* resources are available for a particular execution instance of an application, and determine *when* those resources will be able to deliver sufficient power to meet the application's needs.

The information necessary to answer the question of what resources are configured into the Grid is relatively static or slowly changing. However, to answer the question of when a resource will have sufficient capacity to support an application requires a time-sensitive *prediction* of future performance levels. Since Grid resources are shared, the performance they can deliver varies, possibly frequently, as a result of contention. Also, resources may become unavailable because of transient software failures, hardware failures, etc. As such, the Grid information services must be able to gather and manage "dynamic" data – data that has an often short lifetime of utility, or which is updated frequently.

Initial efforts to use a single information service such as the Globus Metacomputing Directory Service [3] (which is based on the Lightweight Directory Access Protocol – LDAP) as both a storage and and retrieval mechanism for static and dynamic data proved unsuccessful. Retrieval performance was good, but the load generated by frequent storage of dynamic performance measurement data caused the information system as a whole to become unusable. This is natural given the original intent of LDAP, i.e. as a repository for relatively static data. Other systems, such as the Network Weather Service (NWS) [20] have been designed explicitly to store and manage dynamic data. However, they often must sacrifice query performance as they must gather data from widely scattered data repositories. Further, the NWS uses non-standard wire protocols and data representations internally, and presents its data to interested clients via an NWS-specific API. Clearly, it is undesirable to require Grid users to use two different information systems, each with its own API, for static and dynamic data.

This paper describes our attempts to provide a unified information service for Grid systems that manages both static and dynamic data effectively. This service is based on the Globus MDS-2 [3] architecture and the NWS [20]. The implementation we have developed presents a uniform Grid information interface to users with the performance necessary to support Grid schedulers such as those described in [2, 14, 15]. At the same time, it serves as a reference implementation for constructing a dynamic information management system according to the specifications of the architecture.

Our work attempts to resolve a number of conflicting re-

*This work was supported, in part, by a grant from the National Science Foundation's NGS program (EIA-9975020)

quirements for Grid information systems as they exist today. First, the Grid information system must present the data with semantics that afford sufficiently rich expressiveness and flexibility, but at the same time it must be able to gather and deliver the data in an efficient manner. The performance of the Grid information system — like the performance of Grid applications it serves — matters. Second, while we propose a unified service, we foresee the necessity to present the data in multiple formats or encodings. The format translations must also be sensitive to overhead and not impose a significant performance penalty.

Specifically, in this paper we:

1. Describe a new data model for dynamic Grid information and detail its relation to the NWS and the extant GIS (which is based in large part on the MDS)

2. Present a caching NWS server that binds this object model to LDAP syntax and integrates it into the MDS server hierarchy

2 System Requirements

Due to the nature of Grid environments, the information infrastructure of a distributed system has a number of unique design considerations. Grid systems can be described as consisting of *federated* resources, which is to say that there is no central administration of the distributed "system" as a whole. Likewise, there is no single operating system or programming interface that can be thought of as "the" standard, or ubiquitously available. This heterogeneity poses a challenge to any Grid-wide infrastructure.

Given early experiences with prototype, publicly available LDAP-based implementations, the Network Weather Service moved away from using LDAP databases to store information and focused on providing retrieval access to data using LDAP as a delivery mechanism. In general, we believe that Grid information systems should separate the *presentation* of the data from the *storage* and management functions necessary to serve the data.

Another way to view this dichotomy is the notion of separating the *semantics* from the *syntax*. The architecture of the object model for a system defines its semantics. If two systems used different syntaxes to communicate, but used compatible object models, then devising an efficient translator to facilitate intercommunication is feasible. Again, we contend that the important issues center around data organization, not data presentation. However, the organization should not preclude any particular presentation and should translate efficiently.

2.1 Data Management Requirements

To implement such a system, it is clear that we need an efficient internal data management infrastructure. The key realization is that the data organization is critical to this goal.

In identifying representations and relationships of the data atoms associated with a Grid Performance Information Service, it became obvious that the use of the Lightweight Directory Access Protocol (LDAP) constrained the object definitions and relationships that could be realized. In particular, LDAP defines a hierarchical name-space and generally uses a hierarchical database for storage. This is a natural consequence of the original intent of X.500 [11], the precursor to LDAP. We find that this organization can be effective in certain circumstances, but limiting in others. For instance, some data atoms appear naturally in multiple places in such a tree structure, but it is not acceptable to duplicate the storage of this information.

On the other hand, there is some support within the Grid infrastructure community for relational database techniques as a means of implementing GIS infrastructure [4, 16]. We note that the consistency requirements of relational systems are often not feasible in dynamically changing Grid environments, nor are they universally necessary. Certain properties of relational databases are attractive, however.

We propose the definition (if not representation) of each datum in a normalized form, specifically, the third normal form (3NF) as the basis of an efficiently implementable compromise. This representation insures that redundant information is minimized. However to mitigate any negative performance impact, we submit that a clear definition of aggregate data types must exist so that denormalized versions can be easily identified. By reducing the data in this fashion, we will be able to perform mechanical translation between objects from different systems in cases where their representations are different. For instance, we must take great pains to avoid multi-valued attributes in "base" objects, as these will not be appropriate for storage in relational databases without further normalization.

This approach also facilitates object translation. Since there are many opinions about the appropriate granularity of performance information, we believe that the only way to adequately support the myriad of applications and users is to be able to easily add new schemas to the storage system and that users should be free to design their own views without substantial re-engineering.

2.2 Presentation Requirements

While it is undesirable to force Grid application programmers to code to different APIs for static and dynamic data because the latter is more load-intensive to gather and manage, it is important for the Grid to support multiple presentation formats to ease programming and legacy code integration. Building on the generality and flexibility of the data management structure described above, we contend that this can be supported efficiently.

In this vein, many groups have expressed interest in presenting the data via a Structured Query Language (SQL) interface [4]. Moreover, we anticipate a binding to Structure of Management Information version 2 (SMIv2) as defined in RFC 2578 [12].

As such, we have identified the following presentation formats as important to the Grid community:

- LDAP
- XML
- SQL
- SNMP
- EveryWare [19] (Native) Messages

That is, no matter how the data is gathered, stored, replicated, etc., it should be accessible in any of these presentation formats according to the needs of the application or its user.

2.3 Performance Requirements

There are also performance requirements for such a system that have become apparent as we have gained operational experience in Grid testbeds such as the Grid Application Development Software (GrADS) infrastructure [1]. These requirements have driven the requirements articulated above, and are enabled to some degree by the generality of the components. In particular, since application resource selectors and schedulers operate at runtime, their performance directly impacts application execution as overhead. To maximize the efficiency of such Grid scheduling tools, the information system itself must be as efficient as possible.

To achieve the necessary performance goals, we believe that several design principles are general and inherent to the implementation of any Grid information system. First, this system must support caching. Scalability of Internet-wide systems such as the Domain Name System (DNS) have demonstrated that caching is essential for scalability. As these systems generally deal with information with much lower update frequency than our system mandates, our caching infrastructure must be substantially different.

In the same way that there is no single mechanism to deliver performance data to the user, there is no consensus about what information is relevant. More importantly, different applications require different levels of abstraction. In short, the formation and caching of various derived data types, potentially at various levels within the hierarchy, must be supported by our system.

Therefore, we require an abstract set of objects and messages that can be rendered in a variety of wire protocols. In addition, we also require a simple set of state machine definitions that define the semantics of a given object's receipt and define the expected response.

Finally, these objects must be general enough to be represented in GOS [7], RFC 2252 LDAP Schema [18], SMIv2, and GMA [17] XML forms as well as being stored in a commercial Relational Database Management System (RDBMS). By unifying these mechanisms we afford ourselves maximum flexibility for future extensions.

3 Object Model

In this section, we describe the object model used by the NWS. First, we identify the objects themselves. Then we present both hierarchical and relational representations of the objects, since both organizations are relevant. It is important to note that a hierarchical representation is simply a certain type of relation (i.e, one parent, many children) and that when we show the Directory Information Tree- (DIT-) style diagrams that they could easily be shown as Entity Relationship Diagrams (ERDs).

There are two types of objects in our data model: *registration objects* and *data objects*. Registration objects are used by the NWS to keep track of the data and entities that it manages. While these classes are derived by the needs of the NWS, we believe that they form the core of a general object model for managing dynamic data. That is, we believe all performance data management systems will require these objects (or ones like them) at the very least.

Data objects contain the information that clients consume. A resource discovery mechanism, for example, may only query the system for data objects without regard for how they are registered (i.e. without querying the registration objects). By including both data objects and registration objects however, it is possible to optimize explicitly how the data is accessed by a particular application or Grid programming tool. For example, registration information about where data objects are stored, the medium that is used to store them, replication information, etc. can be used to optimize the retrieval process.

In our object definitions, we make use of the ISO standard Object Identifier (OID) [10] to identify the objects and, in some cases, their relationships. This use is consistent with proposals in the Grid Forum's Grid Information Services working group [6] and will be described in this section.

Also, it is important to note that there are multiple hierarchies in use in this object model. First, there is the "inheritance” hierarchy. This is analogous to the object oriented concept of the same name. Next, there is the EventType hierarchy. With this, the OID that uniquely identifies a given type of Event is grouped with related Events, either by measured item or defining party. Finally, there is a special type of relation that is represented in a hierarchical manner in the DIT.

3.1 Object Definitions

This section defines all the major object classes.

3.1.1 Registration Objects

- **GridDaemon** – This object contains registration information about a supporting or participating process. The daemon is expected to periodically refresh this object.
- **GridEventType/GridMeasurementType** – This object defines a type of event or measured/sampled value. Each object has an OID. We organize these OIDs in a hierarchy so that similar values are rooted at the same place in the tree. In this way searches can be made on a “longest match” basis or can be “wildcarded” to discover the types of measurements available for a given target.
- **GridSkill** – This object represents a *GridDaemon*’s ability to produce a *GridEventType* or serve some other function in the NWS infrastructure.
- **GridControl** – This object represents the entity responsible for ongoing activities. This can be a timer, coordination protocol, (pseudo-)random trigger, or value-based event handler.
- **GridTarget** – This represents a process, host or group of hosts. It is the “subject” of an event. This is the root of objects that allow a user to conveniently refer to groups of hosts or processes that are “interesting”.
- **GridActivity** – This object contains any parameters necessary for an ongoing measurement. Like GridTarget more specific instantiations of this object are tailored for specific sensor activities. Also like GridTarget, activities can be easily created by the end user as needed.
- **GridTopology** – This object is a collection of *GridDaemon*s. They can be ordered in various ways including rings, stars and meshes. A GridTopology can contain other GridTopology objects.
- **GridSeries** – This object describes historical performance information about a resource. By referencing event/measurement objects “under” (or related to) this object, event data can be presented in its most normalized form and treated as a time-series. This object also contains administrative information about where the event stream is stored.

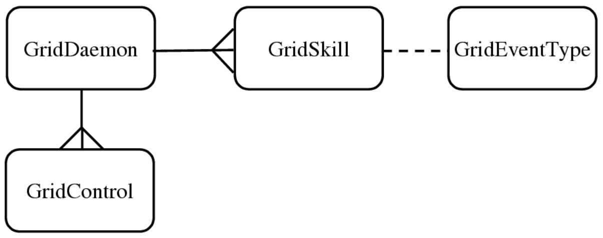

Figure 1. Partial Entity Relationship Diagram (ERD) of objects

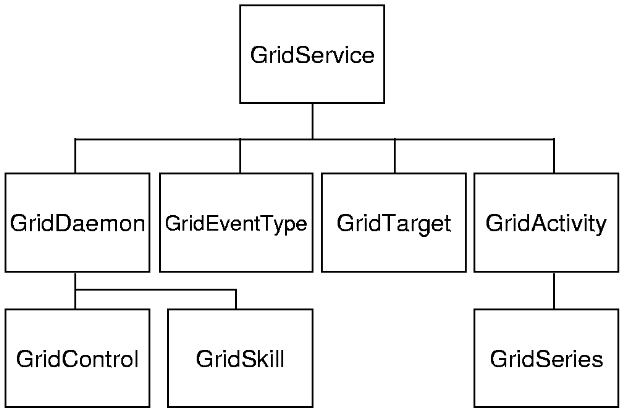

Figure 2. Object Hierarchy represented as a Directory Information Tree (DIT)

3.1.2 Data Objects

- **GridEvent/GridMeasurement** – A *GridEvent* is uniquely identified by a *GridTarget*, a *GridEventType*, and a timestamp. In certain cases, the GridTarget and GridEventType can be specified initially, or out of band, and a stream of GridEvents may just contain a timestamp and a value. One way in which one can get such a stream of “normalized” events is via the *GridSeries* object.

3.2 The GridEventType Hierarchy

The GridEventType hierarchy is a key to the NWS’s extensibility and for fostering collaborative measurement on the Grid. Because the definition of an OID implicitly delegates a subtree of OIDs (any OID prefixed by that parent OID can be used) this organization naturally facilitates easy expansion.

Our initial OID prefix is **iso . org . dod . internet . private . enterprise . University of Tennessee . NWS (1.3.6.1.4.1.13.31)** . However, we anticipate using **iso . org . dod . internet . private . enterprise . Grid (1.3.6.1.4.1.6757)** as well once the mechanisms for registration in that space are complete.

As mentioned above, the purpose of this OID hierarchy is twofold. First, we use OIDs to uniquely specify a given type of event in a lightweight fashion (i.e. the description of the event need not be included in the event itself.) So, this is effectively a just unique name, but it does have hierarchical structure. The second function of this OID-based name is to allow us to group similar measurements together. In this way, a user would be able to discover network measurements available for the **GridTarget** that she is interested by querying at a relevant OID root and then choosing the most appropriate measurement from those returned.

Following are some example **GridEventType** OIDs. Note that {**base**} is used as a shorthand, since this hierarchy can be trivially re-rooted.

- **{base}.perf.net.tcp ({base}.1.1.1.1)**
- **{base}.perf.net.udp ({base}.1.1.1.2)**
- **{base}.perf.host.cpu.available ({base}.1.2.1.1)**

For instance, one could make a query for objects of type {base}.perf.net.tcp. This could return objects of {base}.perf.net.tcp.foo and {base}.perf.net.tcp.bar, and allow the user to make a decision as to which event type(s) are desired.

3.3 Object Relations

The entity relationship diagram (ERD) in Figure 1 illustrates how some of these objects are related to one another. This allows them to be consistently stored in a SQL database or to be queried via SQL. This is not a complete diagram, but the design of the system allows a full representation of objects in this format.

3.4 Hierarchical Organization

Thinking of the objects in terms of their hierarchical organization is useful for understanding the relationships between objects within LDAP frameworks and is also relevant outside the scope of LDAP. In fact, many XML documents make heavy use of object hierarchy with nested labels (and indentation for human consumption.) The basic object hierarchy is shown in Figure 2.

In LDAP, this creates a hierarchy of the following sort.

```
dn: daemon=myhost.my.edu:8090, hn=myhost.my.edu,
    dc=my, dc=edu, o=Grid
    objectClass:  GridDaemon
    version:      version
    owner:        username
    started:      timestamp
    port:         port
    type:         NwsMemory,NwsSensor,NwsForecaster
```

```
dn: skill=tcpBandwidth, daemon=myhost.my.edu:8090,
    hn=myhost.my.edu, dc=my, dc=edu, o=Grid
    objectClass:  GridSkill
    type:         a GridEventType
    param:        parameters
    param-range:  parameter ranges
```

```
dn: control=my-clique, daemon=myhost.my.edu:8090,
    hn=myhost.my.edu, dc=my, dc=edu, o=Grid
    objectclass:  GridControl
    param:        parameters
    param-value:  parameter values
    activity:     dn of GridActivity
```

3.5 Registration in the Grid

This infrastructure supports the NWS as a stand-alone system. However, the NWS is also able to register itself in the MDS [3] using the Grid Resource Registration Protocol (GRRP). The daemon should identify itself as service=GridDaemon under the hn=host.name, dn=domain, dn=edu, o=Grid branch of the MDS DIT using any server already running there. Further, each GridDaemon should respond to a Grid Information Protocol (GRIP) with its hierarchy as follows.

```
dn: MdsService=myhost.my.edu:8090, hn=myhost.my.edu,
    dc=my, dc=edu, o=Grid
    objectClass:        MdsService
    Mds-Service-Type:   GridDaemon
    Mds-Service-port:   port-number
    Mds-Service-hn:     hostname
```

The **GridSkill** and **GridSkill** objects described above would then be returned as children of this object rather than the **GridDaemon** object.

Aside from performance activities, clients and other GIS servers can “subscribe” to events (including future GRRP registrations) or create user-specific GridActivity objects. In fact we can control registration of a GridDaemon to an

NWS GRIS by loading these control objects when the daemon starts, so that prior to receiving any invitations, the GridDaemon will register with designated servers.

4 Implementation

In this section, we describe the process architecture we have used to integrate the NWS with the MDS. The NWS consists of a number of separate but cooperating processes that are organized into subsystems. All processes communicate via statically typed messages that are transferred via the sockets interface using TCP/IP. Currently, there are five subsystems.

- **Sensor subsystem** — All performance measurements are gathered by *sensors* which introduce data into the system via a well-defined message format.
- **Persistent State subsystem** — To aid robustness, sensors should be stateless. Any data that needs to survive a system restart, hardware failure, etc. can be stored remotely in one of many persistent state servers.
- **Forecasting subsystem** — Forecasts can either be generated in the client API by calls to a forecasting library, or in separate forecasting processes. The advantage to having forecasting servers is that both the computational and memory burden introduced by forecasting can be off-loaded from the client, if necessary.
- **Reporting subsystem** — Sensor measurement and forecast data can be presented in a variety of formats. The Reporting subsystem includes command-line tools, Java tools, and HTML/CGI tools for extracting and visualizing the data managed by the other subsystems.
- **Naming subsystem** — All NWS processes register their contact information (IP address, and port number), their characteristics (process type, when it was started, by what user id, version number, etc.), and the names of any persistent objects or files they manage with a name service.

Our original intention was for the internal architecture of the NWS to be opaque to its clients. As such, we designed and implemented custom protocols for all management functions. In particular, we implemented our own name server and registration protocols. At the time we were developing the initial implementation of the system, we experimented with the open versions of LDAP that were available and found that they did not provide the robustness or the performance levels adequate to our needs.

Since then, however, the publicly available versions of LDAP have improved dramatically. In addition, it has become clear that our user community is willing to optimize

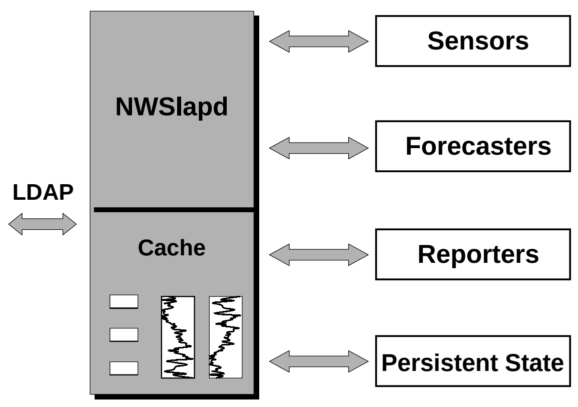

Figure 3. Combined LDAP registry and NWS data cache.

their use of the system to gain performance. To do so requires name server information that our initial API did not export (e.g. sensor topologies). Rather that reworking the API and the name server, we decided to replace it with LDAP. For this reason, we have developed the registration objects described in Section 3. Our intention is to use the MDS itself as the name service for the NWS. Not only does the MDS provide a wider user community with access to the system (it installs with the standard Globus distribution), but tools such as the GrADS [1, 9] Resource Selector that take advantage of internal NWS structural information for performance can also have access to this information via a single LDAP information system interface.

At the same time, we wish to provide measurement and forecast data through the MDS, in effect, using the MDS as another component to the NWS reporting system. Having worked extensively as part of the GrADS development efforts [14], we have developed a set of caching policies for NWS data that greatly optimize performance. As such, we have implemented the MDS interface to the NWS as a daemon process (called *NWSlapd*) that provides a caching LDAP interface to the NWS. Figure 3 depicts this architecture.

The various NWS subsystem processes register with the NWSlapd. Data from the persistent state subsystem (which is stored as a time-series) is also kept by a caching layer within the NWSlapd process. When an LDAP query for NWS data is sent to the MDS, the request is passed to the NWSlapd via an LDAP referral [18]. If the data is in the cache, it is served immediately. If not, the registration objects stored in the NWSlapd are consulted for the location of the persistent state server responsible for the data, and a request to that server is generated using the NWS wire

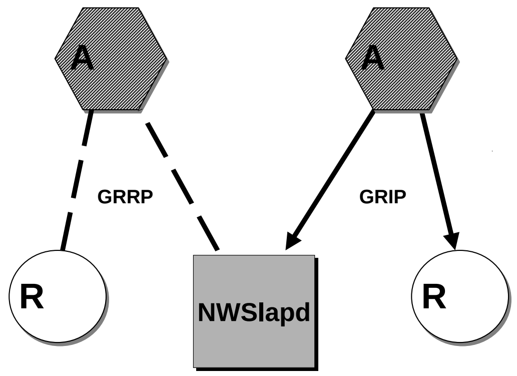

Figure 4. NWSlapd registers with a VO Aggregate directory (*A*) with GRRP and responds to GRIP requests like other Resources (*R*). That is, NWSlapd functions as a GRIS.

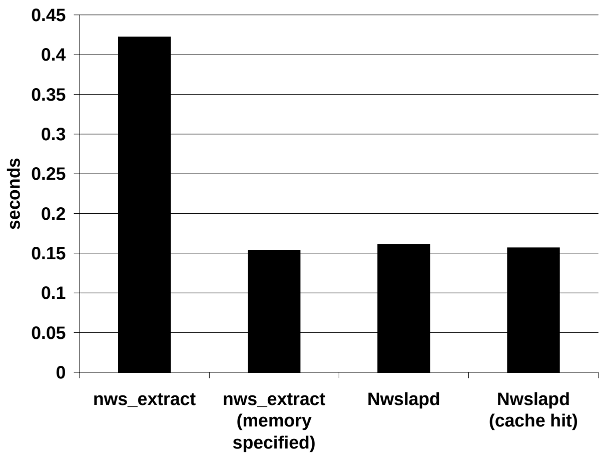

Figure 5. Single datum queries to local infrastructure, NWSlapd

protocols [1]. The data is cached, converted to LDAP, and returned to the requesting client.

Cached data is periodically refreshed by the NWSlapd. Each registered event series is related to an activity, which contains a periodicity attribute. When the data for an event series is cached, the NWSlapd spawns a thread internally that is responsible for adding any new data to the cache according to the periodicity with which it is being generated by its sensor. Cache entries are replaced either LRU or when a configurable timeout threshold is reached.

Note that the daemon is modularized so that the caching layer is separate from the LDAP registration parsing and protocol processing. Our hope is that such a modularity will allow the code to be used as a template for other presentation formats such as XML. Also, because the NWSlapd is based on the code for OpenLDAP [13], it should be possible to use the Slurpd implementation to provide replication. As such, each Virtual Organization (in the Globus sense — see [5]) can have its own NWSlapd which serves as an Grid Resource Information Service (GRIS) node in that organization's MDS hierarchy. Figure 4 depicts that relationship.

5 Performance Results

Finally, we present the performance of the NWSlapd implementation. First, we examine the wall clock time spent querying the NWS for a single datum. To measure this, we used the NWS command-line tool *nws_extract* and the generic LDAP query utility *ldapsearch*. Figure 5 shows the query times from the U. Tennessee campus, where the master NWS nameserver is located.

The four cases that are presented are a simple NWS query, an NWS query where the location of the Persistent State server (*nws_memory*) is known in advance, and the NWSlapd, both with the datum in the cache and without. We note that the NWSlapd actually provides an improvement even when the cache is empty. This is because the NWSlapd implementation caches NWS registration objects so the additional query that the native NWS tools (like nws_extract) must make to find the Persistent State server is unnecessary. We observe that when that additional query is eliminated (by specifying the "nws_memory" in which the data is stored), the times are quite similar.

Next, we examine the case where the client and the cache are located on another campus (UC Santa Barbara in this case.) Both campuses are connected to Internet2 and enjoy a relatively high-bandwidth connection with latency of about 120ms. Figure 6 shows these results. It is plain to see that accessing data from a local cache provides significant performance improvement. Surprisingly, accessing data via the NWSlapd is more efficient than going directly to the data store. However, the *nws_extract* still makes one request to the NWS nameserver, so the additional exchange accounts for the performance difference.

Finally, we test the efficacy of our caching mechanism in an application environment, using the software infrastructure from the GrADS [1] project. As described in recent GrADS work [14], this software makes requests for $2n^2$ data elements from the NWS to form a full table of bandwidth and latency measurements between all pairs of hosts (and the available CPU and memory on each host.) While

[1]Future versions of the NWS may, like LDAP, use the ASN.1 [10] Basic Encoding Rules (BER) as an encoding scheme to facilitate this transfer.

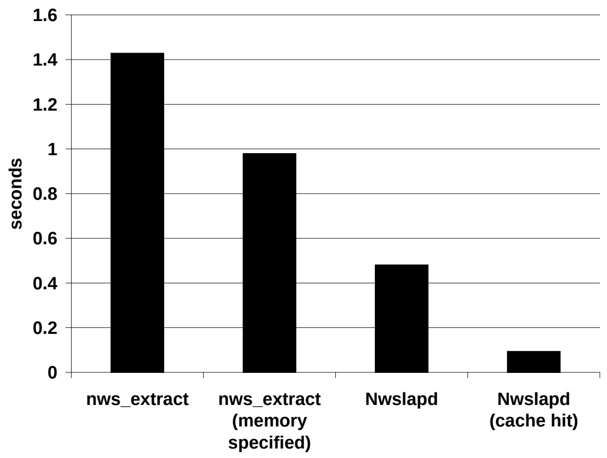

Figure 6. Single datum queries to remote infrastructure, with local NWSlapd

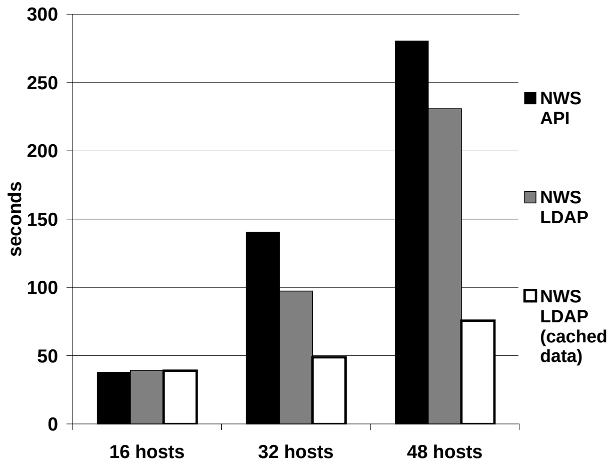

Figure 7. GrADS ScaLAPACK Resource Selector, local infrastructure

we will discuss options that could ameliorate the need for that many requests, in the general case, a full map is often required. Further, we feel that these requests stressed the system regardless of the application requirements.

In Figure 7 we show the query times for various numbers of hosts (**n**). We show the direct access to the NWS via the NWS API (the C language interface to the NWS) and access via the caching LDAP server. We show cases in which the cache is loaded by prior access and when it is not.

Note that the NWS LDAP daemon actually performs better than the native NWS interface as the number of hosts grows. Again, this is because the NWS LDAP daemon proactively caches NWS registration information. In the NWS API case, every pair of queries (cpu/memory or bandwidth/latency) requires a query to the NWS Nameserver to determine where that data can be fetched. With the NWS LDAP daemon, most of that information is already cached. More importantly, the client (library) doesn't need to request the location explicitly, it need only ask for each datum and the location is resolved automatically.

Again, we examine the case where the clients are running at UCSB and the NWS nameserver is running at U. Tennessee. Figure 8 depicts these results. We note that the case in which the cache is loaded is not as profound an improvement as we expect. This under performance is because many of the $2n^2$ measurements described above do not exist. The NWSlapd does not implement negative caching at this time, so queries for non-existent data remain expensive.

There are many ways to improve the resource selection mechanism by offering additional functionality from the NWS. We briefly discuss this in Section 6.

6 Future Work

The NWSlapd can serve as a specialized Grid Resource Information Service (GRIS) within the Globus MDS-2 process hierarchy [3] or as a stand-alone caching mechanism for NWS clients. We are currently working with the Globus MDS team to make this more efficient by pursuing integration the NWSlapd functionality into the Globus MDS distribution as a GRIS "provider plugin". In this way, the protocol and object translation modules will be able to be dynamically loaded into an existing GRIS. Then the NWS will be able to take advantage of the information management facilities that are built into the MDS-2. Access control for NWS dynamic data, for example, will automatically be provided by the existing MDS mechanisms. At the same time, the GRIS includes a generalized caching facility that the NWS can leverage. We plan to investigate whether further performance benefits can be obtained by allowing the NWS to "tune" the caching procedures used by the GRIS for certain types of NWS data such as time series. Perhaps most importantly, however, Globus clients will be able to discover and characterize the NWS data that is available using the MDS-2 resource discovery mechanisms.

Independent of the discovery and delivery mechanisms, we plan to investigate the form in which the data should be delivered in order to maximize performance. We believe further performance benefits can be gained through a tighter integration of information subsystems, and by introducing an aggregation capability within the caching system itself. Our intention is to allow NWS clients to specify aggregate "virtual objects" that are effectively user-defined views, and that exist within the caching hierarchy. The caching system will be responsible for assembling these objects asyn-

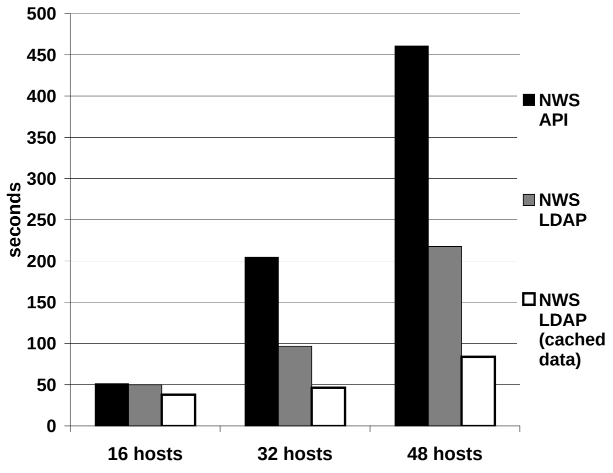

Figure 8. GrADS ScaLAPACK Resource Selector, remote infrastructure, local NWSlapd

chronously and then delivering them on-demand. For example, a client such as the ScaLAPACK scheduler could request the N^2 matrix of network connectivity once, in a single message thereby removing much of the messaging overhead that it currently incurs. We intend to work on general mechanisms for forming these “views” as part of the evolution of the object model.

7 Conclusion

We have presented a vision for dynamic information infrastructure in Grid environments. We have demonstrated its efficacy with an implementation that uses the object model we have defined and implements the late syntactic to semantic binding. This is an important first step in the evolution of Computational Grid infrastructure. We have demonstrated that careful attention to to both performance and flexibility can yield effective tools in this space.

References

[1] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, L. J. Dennis Gannon, K. Kennedy, C. Kesselman, D. Reed, L. Torczon, , and R. Wolski. The GrADS project: Software support for high-level grid application development. Technical Report Rice COMPTR00-355, Rice University, February 2000.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *10th International Symposium on High-Performance Distributed Computing*. IEEE, August 2001. `http://www.globus.org/research/papers.html#GlobusToolkit`.

[4] P. Dinda and B. Plale. A unified relational approach to grid information services. Grid Forum Draft GWD-GIS-012-1, March 2001.

[5] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, August 1997.

[6] S. Fitzgerald, M. Swany, and M. Helm. Using object identifiers in the grid forum standards process. Grid Forum Draft, GWD-GIS-015-4, July 2001.

[7] S. Fitzgerald, G. von Laszewski, and M. Swany. GOS: A Data Definition Language for Grid Information Services Version 2.0. Grid forum working group document, Grid Forum, Feburary 2001. http://www.gridforum.org.

[8] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[9] GrADS. `http://hipersoft.cs.rice.edu/grads`.

[10] I. P. O. S. I. International Organization for Standardization. Specification of abstract syntax notation one (asn.1). ISO/IEC 8824, 1990.

[11] I. P. S. O. S. I. International Organization for Standardization. The directory - overview of cencepts, models, and service. ISO/IEC 9594-1, 1988.

[12] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Structure of management information version 2 (SMIv2). Internet Engineering Task Force, RFC 2578, April 1999.

[13] OpenLDAP. `http://www.openldap.org/`.

[14] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical libraries and the grid. In *(to appear) Proc. of SC01*, November 2001.

[15] N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceedings of ACM International Conference on Supercomputing 1998*, July 1998.

[16] M. Swany. Information services architecture for dynamic information. Grid Information Services Workshop, July 2000.

[17] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany. A Grid Monitoring Architecture. Grid forum working group document, Grid Forum, Feburary 2001. http://www.gridforum.org.

[18] M. Wahl, A. Coulbeck, T. Howes, and S. Kille. Lightweight directory access protocol (v3): Attribute syntax definitions. Internet Engineering Task Force, RFC 2252, December 1997.

[19] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Writing programs that run everyware on the computational grid. *(to appear) IEEE Transactions on Parallel and Distributed Systems*, 2001.

[20] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 1999.