A Directory Service for Configuring High-Performance Distributed Computations

Steven Fitzgerald,[1] Ian Foster,[2] Carl Kesselman,[1] Gregor von Laszewski,[2] Warren Smith,[2] Steven Tuecke[2]

[1] Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292

[2] Mathematics and Computer Science
Argonne National Laboratory
Argonne, IL 60439

http://www.globus.org/

Abstract

High-performance execution in distributed computing environments often requires careful selection and configuration not only of computers, networks, and other resources but also of the protocols and algorithms used by applications. Selection and configuration in turn require access to accurate, up-to-date information on the structure and state of available resources. Unfortunately, no standard mechanism exists for organizing or accessing such information. Consequently, different tools and applications adopt ad hoc mechanisms, or they compromise their portability and performance by using default configurations. We propose a Metacomputing Directory Service *that provides efficient and scalable access to diverse, dynamic, and distributed information about resource structure and state. We define an extensible data model to represent required information and present a scalable, high-performance, distributed implementation. The data representation and application programming interface are adopted from the Lightweight Directory Access Protocol; the data model and implementation are new. We use the Globus distributed computing toolkit to illustrate how this directory service enables the development of more flexible and efficient distributed computing services and applications.*

1 Introduction

High-performance distributed computing often requires careful selection and configuration of computers, networks, application protocols, and algorithms. These requirements do not arise in traditional distributed computing, where configuration problems can typically be avoided by the use of standard default protocols, interfaces, and so on. The situation is also quite different in traditional high-performance computing, where systems are usually homogeneous and hence can be configured manually. But in high-performance distributed computing, neither defaults nor manual configuration is acceptable. Defaults often do not result in acceptable performance, and manual configuration requires low-level knowledge of remote systems that an average programmer does not possess. We need an *information-rich* approach to configuration in which decisions are made (whether at compile-time, link-time, or run-time [19]) based upon information about the structure and state of the system on which a program is to run.

An example from the I-WAY networking experiment illustrates some of the difficulties associated with the configuration of high-performance distributed systems. The I-WAY was composed of massively parallel computers, workstations, archival storage systems, and visualization devices [6]. These resources were interconnected by both the Internet and a dedicated 155 Mb/sec IP over ATM network. In this environment, applications might run on a single or multiple parallel computers, of the same or different types. An optimal communication configuration for a particular situation might use vendor-optimized communication protocols within a computer but TCP/IP between computers over an ATM network (if available). A significant amount of information must be available to select such configurations, for example:

- What are the network interfaces (i.e., IP addresses) for the ATM network and Internet?
- What is the raw bandwidth of the ATM network and the Internet, and which is higher?
- Is the ATM network currently available?
- Between which pairs of nodes can we use vendor protocols to access fast internal networks?
- Between which pairs of nodes must we use TCP/IP?

Additional information is required if we use a resource location service to select an "optimal" set of resources from among the machines available on the I-WAY at a given time.

In our experience, such configuration decisions are not difficult *if* the right information is available. Until now, however, this information has not been easily available, and this lack of access has hindered application optimization. Furthermore, making this information available in a useful fashion is a nontrivial problem: the information required to configure high-performance distributed systems is diverse in scope, dynamic in value, distributed across the network, and detailed in nature.

In this article, we propose an approach to the design of high-performance distributed systems that addresses this need for efficient and scalable access to diverse, dynamic, and distributed information about the structure and state of resources. The core of this approach is the definition and implementation of a Metacomputing Directory Service (MDS) that provides a uniform interface to diverse information sources. We show how a simple data representation and application programming interface (API) based on the Lightweight Directory Access Protocol (LDAP) meet requirements for uniformity, extensibility, and distributed maintenance. We introduce a data model suitable for distributed computing applications and show how this model is able to represent computers and networks of interest. We also present novel implementation techniques for this service that address the unique requirements of high-performance applications. Finally, we use examples from the Globus distributed computing toolkit [9] to show how MDS data can be used to guide configuration decisions with realistic settings. We expect these techniques to be equally useful in other systems that support computing in distributed environments, such as Legion [12], NEOS [5], NetSolve [4], Condor [16], Nimrod [1], PRM [18], AppLeS [2], and heterogeneous implementations of MPI [13].

The principal contributions of this article are

- a new architecture for high-performance distributed computing systems, based upon an information service called the Metacomputing Directory Service;
- a design for this directory service, addressing issues of data representation, data model, and implementation;
- a data model able to represent the network structures commonly used by distributed computing systems, including various types of supercomputers; and
- a demonstration of the use of the information provided by MDS to guide resource and communication configuration within a distributed computing toolkit.

The rest of this article is organized as follows. In Section 2, we explain the requirements that a distributed computing information infrastructure must satisfy, and we propose MDS in response to these requirements. We then describe the representation (Section 3), the data model (Section 4), and the implementation (Section 5) of MDS. In Section 6, we demonstrate how MDS information is used within Globus. We conclude in Section 7 with suggestions for future research efforts.

2 Designing a Metacomputing Directory Service

The problem of organizing and providing access to information is a familiar one in computer science, and there are many potential approaches to the problem, ranging from database systems to the Simple Network Management Protocol (SNMP). The appropriate solution depends on the ways in which the information is produced, maintained, accessed, and used.

2.1 Requirements

Following are the requirements that shaped our design of an information infrastructure for distributed computing applications. Some of these requirements can be expressed in quantitative terms (e.g., scalability, performance); others are more subjective (e.g., expressiveness, deployability).

Performance. The applications of interest to us frequently operate on a large scale (e.g., hundreds of processors) and have demanding performance requirements. Hence, an information infrastructure

must permit rapid access to frequently used configuration information. It is not acceptable to contact a server for every item: caching is required.

Scalability and cost. The infrastructure must scale to large numbers of components and permit concurrent access by many entities. At the same time, its organization must permit easy discovery of information. The human and resource costs (CPU cycles, disk space, network bandwidth) of creating and maintaining information must also be low, both at individual sites and in total.

Uniformity. Our goal is to simplify the development of tools and applications that use data to guide configuration decisions. We require a uniform data model as well as an application programming interface (API) for common operations on the data represented via that model. One aspect of this uniformity is a standard representation for data about common resources, such as processors and networks.

Expressiveness. We require a data model rich enough to represent relevant structure within distributed computing systems. A particular challenge is representing characteristics that span organizations, for example network bandwidth between sites.

Extensibility. Any data model that we define will be incomplete. Hence, the ability to incorporate additional information is important. For example, an application can use this facility to record specific information about its behavior (observed bandwidth, memory requirements) for use in subsequent runs.

Multiple information sources. The information that we require may be generated by many different sources. Consequently, an information infrastructure must integrate information from multiple sources.

Dynamic data. Some of the data required by applications is highly dynamic: for example, network availability or load. An information infrastructure must be able to make this data available in a timely fashion.

Flexible access. We require the ability to both read and update data contained within the information infrastructure. Some form of search capability is also required, to assist in locating stored data.

Security. It is important to control who is allowed to update configuration data. Some sites will also want to control access.

Deployability. An information infrastructure is useful only if is broadly deployed. In the current case, we require techniques that can be installed and maintained easily at many sites.

Decentralized maintenance. It must be possible to delegate the task of creating and maintaining information about resources to the sites at which resources are located. This delegation is important for both scalability and security reasons.

2.2 Approaches

It is instructive to review, with respect to these requirements, the various (incomplete) approaches to information infrastructure that have been used by distributed computing systems.

Operating system commands such as `uname` and `sysinfo` can provide important information about a particular machine but do not support remote access. SNMP [21] and the Network Information Service (NIS) both permit remote access but are defined within the context of the IP protocol suite, which can add significant overhead to a high-performance computing environment. Furthermore, SNMP does not define an API, thus preventing its use as a component within other software architectures.

High-performance computing systems such as PVM [11], p4 [3], and MPICH [13] provide rapid access to configuration data by placing this data (e.g., machine names, network interfaces) into files maintained by the programmer, called "hostfiles." However, lack of support for remote access means that hostfiles must be replicated at each host, complicating maintenance and dynamic update.

The Domain Name Service (DNS) provides a highly distributed, scalable service for resolving Internet addresses to values (e.g., IP addresses) but is not, in general, extensible. Furthermore, its update strategies are designed to support values that change relatively rarely.

The X.500 standard [14, 20] defines a directory service that can be used to provide extensible distributed directory services within a wide area environment. A directory service is a service that provides read-optimized access to general data about entities, such as people, corporations, and computers. X.500 provides a framework that could, in principle, be used to organize the information that is of interest to us. However, it is complex and requires ISO protocols and the

heavyweight ASN.1 encodings of data. For these and other reasons, it is not widely used.

The Lightweight Directory Access Protocol [24] is a streamlined version of the X.500 directory service. It removes the requirement for an ISO protocol stack, defining a standard wire protocol based on the IP protocol suite. It also simplifies the data encoding and command set of X.500 and defines a standard API for directory access [15]. LDAP is seeing wide-scale deployment as the directory service of choice for the World Wide Web. Disadvantages include its only moderate performance (see Section 5), limited access to external data sources, and rigid approach to distributing data across servers.

Reviewing these various systems, we see that each is in some way incomplete, failing to address the types of information needed to build high-performance distributed computing systems, being too slow, or not defining an API to enable uniform access to the service. For these reasons, we have defined our own metacomputing information infrastructure that integrates existing systems while providing a uniform and extensible data model, support for multiple information service providers, and a uniform API.

2.3 A Metacomputing Directory Service

Our analysis of requirements and existing systems leads us to define what we call the Metacomputing Directory Service (MDS). This system consists of three distinct components:

1. **Representation and data access**: The directory structure, data representation, and API defined by LDAP.

2. **Data model**: A data model that is able to encode the types of resources found in high-performance distributed computing systems.

3. **Implementation**: A set of implementation strategies designed to meet requirements for performance, multiple data sources, and scalability.

We provide more details on each of these components in the following sections.

Figure 1 illustrates the structure of MDS and its role in a high-performance distributed computing system. An application running in a distributed computing environment can access information about system structure and state through a uniform API. This information is obtained through the MDS client library, which may access a variety of services and data sources when servicing a query.

3 Representation

The MDS design adopts the data representations and API defined by the LDAP directory service. This choice is driven by several considerations. Not only is the LDAP data representation extensible and flexible, but LDAP is beginning to play a significant role in Web-based systems. Hence, we can expect wide deployment of LDAP information services, familiarity with LDAP data formats and programming, and the existence of LDAP directories with useful information. Note that the use of LDAP representations and API does not constrain us to use standard LDAP implementations. As we explain in Section 5, the requirements of high-performance distributed computing applications require alternative implementation techniques. However, LDAP provides an attractive interface on which we can base our implementation. LDAP also provides a mechanism to restrict the types of operations that can be performed on data, which helps to address our security requirements.

In the rest of this section, we talk about the "MDS representation," although this representation comes directly from LDAP (which in turn "borrows" its representation from X.500). In this representation, related information is organized into well-defined collections, called entries. MDS contains many entries, each representing an instance of some type of object, such as an organization, person, network, or computer. Information about an entry is represented by one or more attributes, each consisting of a name and a corresponding value. The attributes that are associated with a particular entry are determined by the type of object the entry represents. This type information, which is encoded within the MDS data model, is encoded in MDS by associating an *object class* with each entry. We now describe how entries are named and then, how attributes are associated with objects.

3.1 Naming MDS Entries

Each MDS entry is identified by a unique name, called its *distinguished name*. To simplify the process of locating an MDS entry, entries are organized to form a hierarchical, tree-structured name space called a *directory information tree* (DIT). The distinguished name for an entry is constructed by specifying the entries on the path from the DIT root to the entry being named.

Each component of the path that forms the distinguished name must identify a specific DIT entry. To enable this, we require that, for any DIT entry, the children of that entry must have at least one attribute, specified a priori, whose value distinguishes it from

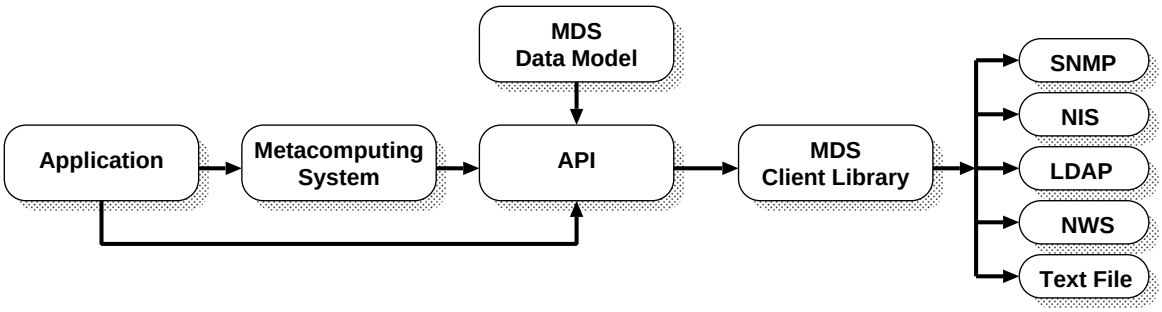

Figure 1. Overview of the architecture of the Metacomputing Directory Service

its siblings. (The X.500 representation actually allows more than one attribute to be used to disambiguate names.) Any entry can then be uniquely named by the list of attribute names and values that identify its ancestors up to the root of the DIT. For example, consider the following MDS distinguished name:

```
< hn = dark.mcs.anl.gov,
  ou = MCS,
  o  = Argonne National Laboratory,
  o  = Globus,
  c  = US >
```

The components of the distinguished name are listed in *little endian* order, with the component corresponding to the root of the DIT listed last. Within a distinguished name, abbreviated attribute names are typically used. Thus, in this example, the names of the distinguishing attributes are: host name (HN), organizational unit (OU), organization (O), and country (C). Thus, a country entry is at the root of the DIT, while host entries are located beneath the organizational unit level of the DIT (see Figure 2). In addition to the conventional set of country and organizational entries (US, ANL, USC, etc.), we incorporate an entry for a pseudo-organization named "Globus," so that the distinguished names that we define do not clash with those defined for other purposes.

3.2 Object Classes

Each DIT entry has a user-defined type, called its *object class*. (LDAP defines a set of standard object class definitions, which can be extended for a particular site.) The object class of an entry defines which attributes are associated with that entry and what type of values those attributes may contain. For example, Figure 3 shows the definition of the object classes `GlobusHost` and `GlobusResource`, and Figure 4 shows the values associated with a particular host. The object class definition consists of three parts: a parent class, a list of required attributes, and a list of optional attributes.

The `SUBCLASS` section of the object class definition enables a simple inheritance mechanism, allowing an object class to be defined in terms of an extension of an existing object class. The `MUST CONTAIN` and `MAY CONTAIN` sections specify the required and optional attributes found in an entry of this object class. Following each attribute name is the type of the attribute value. While the set of attribute types is extensible, a core set has been defined, including case-insensitive strings (`cis`) and distinguished names (`dn`).

In Figure 3, `GlobusHost` inherits from the object class `GlobusResource`. This means that a `GlobusHost` entry (i.e., an entry of type `GlobusHost`) contains all of the attributes required by the `GlobusResource` class, as well as the attributes defined within its own `MUST CONTAIN` section. In Figure 4, the administrator attribute is inherited from `GlobusResource`. A `GlobusHost` entry may also optionally contain the attributes from both its parent's and its own `MAY CONTAIN` section.

Notice that the administrator attribute in Figure 4 contains a distinguished name. This distinguished name acts as a pointer, linking the host entry to the person entry representing the administrator. One must be careful not to confuse this link, which is part of an entry, with the relationships represented by the DIT, which are not entry attributes. The DIT should be thought of as a separate structure used to organize an arbitrary collection of entries and, in particular, to enable the distribution of these entries over multiple physical sites. Using distinguished names as attribute values enables one to construct more complex relation-

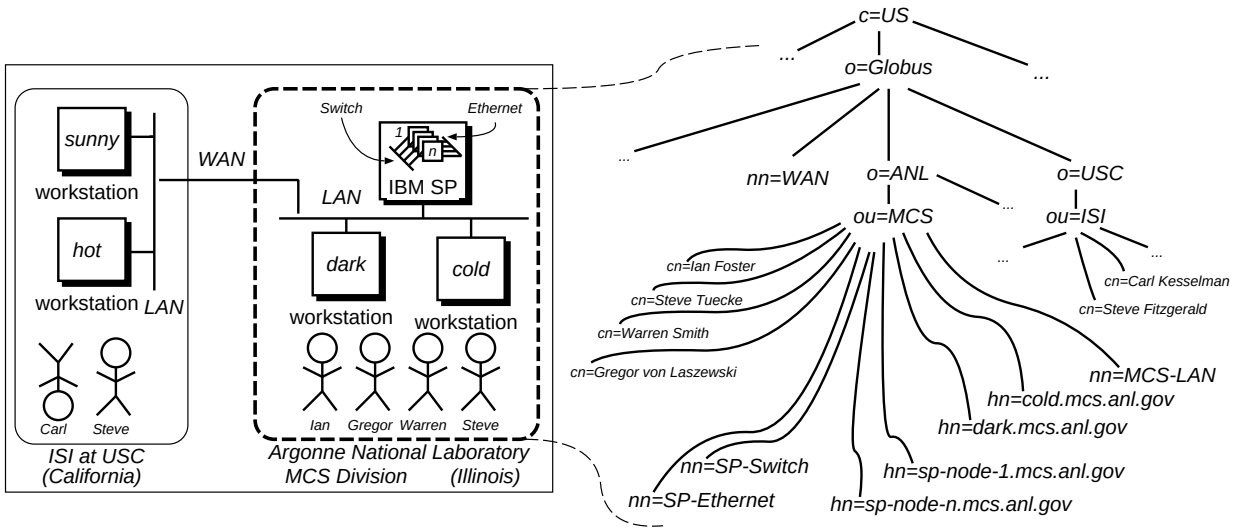

Figure 2. A subset of the DIT defined by MDS, showing the organizational nodes for Globus, ANL, and USC; the organizational units ISI and MCS; and a number of people, hosts, and networks.

ships than the trees found in the DIT. The ability to define more complex structures is essential for our purposes, since many distributed computing structures are most naturally represented as graphs.

4 Data Model

To use the MDS representation for a particular purpose, we must define a data model in which information of interest can be maintained. This data model must specify both a DIT hierarchy and the object classes used to define each type of entry.

In its upper levels, the DIT used by MDS (see Figure 2) is typical for LDAP directory structures, looking similar to the organization used for multinational corporations. The root node is of object class *country*, under which we place first the *organization* entry representing Globus and then the *organization* and *organizational unit* (i.e., division or department) entries. Entries representing people and computers are placed under the appropriate organizational units.

The representation of computers and networks is central to the effective use of MDS, and so we focus on this issue in this section.

4.1 Representing Networks and Computers

We adopt the framework for representing networks introduced in RFC 1609 [17] as the starting point for the representation used in MDS. However, the RFC 1609 framework provides a network-centric view in which computers are accessible only via the networks to which they are connected. We require a representation of networks and computers that allows us to answer questions such as

- Are computers A and B on the same network?
- What is the latency between computers C and D?
- What protocols are available between computers E and F?

In answering these questions, we often require access to information about networks, but questions are posed most often from the perspective of the computational resource. That is, they are computer-centric questions. Our data model reflects this perspective.

A high-level view of the DIT structure used in MDS is shown in Figure 2. As indicated in this figure, both people and hosts are immediate children of the organizations in which they are located. For example, the distinguished name

```
GlobusHost OBJECT CLASS
  SUBCLASS OF GlobusResource
  MUST CONTAIN {
    hostName         :: cis,
    type             :: cis,
    vendor           :: cis,
    model            :: cis,
    OStype           :: cis,
    OSversion        :: cis
  }
  MAY CONTAIN {
    networkNode       :: dn,
    totalMemory       :: cis,
    totalSwap         :: cis,
    dataCache         :: cis,
    instructionCache  :: cis
  }
```

```
GlobusResource OBJECT CLASS
  SUBCLASS OF top
  MUST CONTAIN {
    administrator :: dn
  }
  MAY CONTAIN {
    manager       :: dn,
    provider      :: dn,
    technician    :: dn,
    description   :: cis,
    documentation :: cis
  }
```

Figure 3. Simplified versions of the MDS object classes `GlobusHost` and `GlobusResource`

```
dn: <hn=dark.mcs.anl.gov, ou=MCS,
     o=Argonne National Laboratory, o=Globus, c=US>
objectclass:   GlobusHost
objectclass:   GlobusResource
administrator: <cn=John Smith, ou=MCS,
                o=Argonne National Laboratory, o=Globus, c=US>
hostName:      dark.mcs.anl.gov
type:          sparc
vendor:        Sun
model:         SPARCstation-10
OStype:        SunOS
OSversion:     5.5.1
```

Figure 4. Sample data representation for an MDS computer

```
< hn=dark.mcs.anl.gov,
  ou=MCS, o=Argonne National Laboratory,
  o=Globus, c=US >
```

identifies a computer administered by the Mathematics and Computer Science (MCS) Division at Argonne National Laboratory.

Communication networks are also explicitly represented in the DIT as children of an organization. For example, the distinguished name

```
< nn=mcs-lan,
  ou=MCS, o=Argonne National Laboratory,
  o=Globus, c=US >
```

represents the local area network managed by MCS. This distinguished name identifies an instance of a `GlobusNetwork` object. The attribute values of a `GlobusNetwork` object provides information about the *physical* network link, such as the link protocol (e.g., ATM or Ethernet), network topology (e.g., bus or ring type), and physical media (e.g., copper or fiber). As we shall soon see, logical information, such as the network protocol being used, is *not* specified in the `GlobusNetwork` object but is associated with a `GlobusNetworkImage` object. Networks that span organizations can be represented by placing the `GlobusNetwork` object higher in the DIT.

Networks and hosts are related to one another via `GlobusNetworkInterface` objects: hosts contain net-

work interfaces, and network interfaces are attached to networks. A network interface object represents the physical characteristics of a network interface (such as interface speed) and the hardware network address (e.g. the 48-bit Ethernet address in the case of Ethernet). Network interfaces appear under hosts in the DIT, while a network interface is associated with a network via an attribute whose value is a distinguished name pointing to a `GlobusNetwork` object. A reverse link exists from the `GlobusNetwork` object back to the interface.

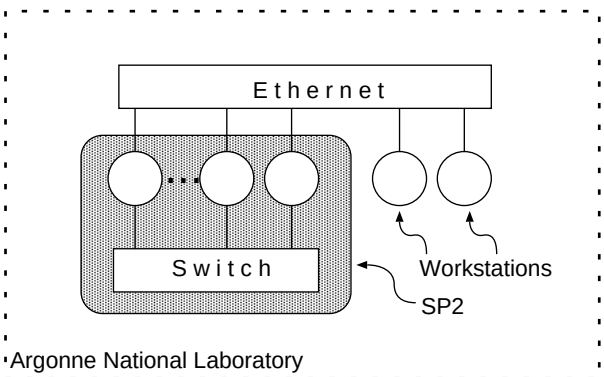

Figure 5. A configuration comprising two networks and $N+2$ computers

To illustrate the relationship between `GlobusHost`, `GlobusNetwork`, and `GlobusNetworkInterface` objects, we consider the configuration shown in Figure 5. This configuration consists of an IBM SP parallel computer and two workstations, all associated with MCS. The SP has two networks: an internal high-speed switch and an Ethernet; the workstations are connected only to an Ethernet. Although the SP Ethernet and the workstation Ethernet are connected via a router, we choose to represent them as a single network. An alternative, higher-fidelity MDS representation would capture the fact that there are two interconnected Ethernet networks.

The MDS representation for Figure 5 is shown in Figure 6. Each host and network in the configuration appear in the DIT directly under the entry representing MCS at Argonne National Laboratory. Note that individual SP nodes are children of MCS. This somewhat unexpected representation is a consequence of the SP architecture: each node is a fully featured workstation, potentially allowing login. Thus, the MDS representation captures the dual nature of the SP as a parallel computer (via the switch network object) and as a collection of workstations.

As discussed above, the `GlobusNetworkInterface` objects are located in the DIT under the `GlobusHost` objects. Note that a `GlobusHost` can have more than one network interface entry below it. Each entry corresponds to a different physical network connection. In the case of an SP, each node has at least two network interfaces: one to the high-speed switch and one to an Ethernet. Finally, we see that distinguished names are used to complete the representation, linking the network interface and network object together.

4.2 Logical Views and Images

At this point, we have described the representation of a physical network: essentially link-level aspects of the network and characteristics of network interface cards and the hosts they plug into. However, a physical network may support several "logical" views, and we may need to associate additional information with these logical views. For example, a single network might be accessible via several different protocol stacks: IP, Novell IPX, or vendor-provided libraries such as MPI. Associated with each of these protocols can be distinct network interface and performance information. Additionally, a "partition" might be created containing a subset of available computers; scheduling information can be associated with this object.

The RFC 1609 framework introduces the valuable concept of *images* as a mechanism for representing multiple logical views of the same physical network. We apply the same concept in our data model. Where physical networks are represented by `GlobusHost`, `GlobusNetwork`, and `GlobusNetworkInterface` object classes, network images are represented by `GlobusHostImage`, `GlobusNetworkImage`, and `GlobusNetworkInterfaceImage` object classes. Each image object class contains new information associated with the logical view, as well as a distinguished name pointing to its relevant physical object. In addition, a physical object has distinguished name pointers to all of the images that refer to it. For example, one may use both IP and IPX protocols over a single Ethernet interface card. We would represent this in MDS by creating two `GlobusNetworkInterfaceImage` objects. One image object would represent the IP network and contain the IP address of the interface, as well as a pointer back to the object class representing the Ethernet card. The second image object would contain the IPX address, as well as a distinguished name pointing back to the same entry for the Ethernet card. The `GlobusNetworkInterface` object would in-

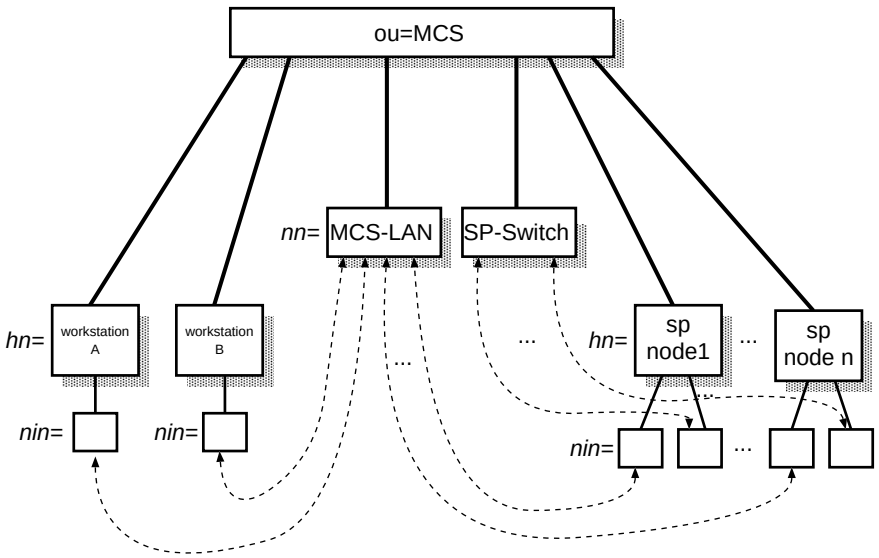

Figure 6. The MDS representation of the configuration depicted in Figure 5, showing host (HN), network (NN), and network interface (NIN) objects. The dashed lines correspond to "pointers" represented by distinguished name attributes

clude the distinguished names of both interface images.

The structure of network images parallels that of the corresponding physical networks, with the exception that not all network interfaces attached to a host need appear in an image. To see why, consider the case of the IBM SP. One might construct a network image to represent the "parallel computer" view of the machine in which IBM's proprietary message-passing library is used for communication. Since this protocol cannot be used over the Ethernet, this image of the network will not contain images representing the Ethernet card. Note that we can also produce a network image of the SP representing the use of IP protocols. This view may include images of both the switch and Ethernet network interfaces.

4.3 Questions Revisited

At this stage we have gone quite deeply into the representation of computers and networks but have strayed rather far from the issue that motivated the MDS design, namely, the configuration of high-performance distributed computations. To see how MDS information can be used, let us revisit the questions posed in Section 1 with respect to the use of multiple computers on the I-WAY:

- *What are the network interfaces (i.e., IP addresses) for the ATM network and Internet?* A host's IP address on the ATM network can be found by looking for a `GlobusNetworkInterface` that is pointing to a `GlobusNetwork` with a link protocol attribute value of ATM. From the interface, we find the `GlobusNetworkInterfaceImage` representing an IP network, and the IP address will be stored as an attribute in this object.

- *What is the raw bandwidth of the ATM network and the Internet, and which is higher? Is the ATM network currently available?* The raw bandwidth of the ATM network will be stored in the I-WAY `GlobusNetwork` object. Information about the availability of the ATM network can also be maintained in this object.

- *Between which pairs of nodes can we use vendor protocols to access fast internal networks? Between which pairs of nodes must we use TCP/IP?* Two nodes can communicate using a vendor protocol if they both point to `GlobusHostImage` objects that belong to the same `GlobusNetworkImage` object.

Note that the definition of the MDS representation, API, and data model means that this information can

be obtained via a single mechanism, regardless of the computers on which an application actually runs.

5 Implementation

We have discussed how information is represented in MDS, and we have shown how this information can be used to answer questions about system configuration. We now turn our attention to the MDS implementation. Since our data model has been defined completely within the LDAP framework, we could in principle adopt the standard LDAP implementation. This implementation uses a TCP-based wire protocol and a distributed collection of servers, where each server is responsible for all the entries located within a complete subtree of the DIT. While this approach is suitable for a loosely coupled, distributed environment, it has three significant drawbacks in a high-performance environment:

- **Single information provider**. The LDAP implementation assumes that all information within a DIT subtree is provided by a single information provider. (While some LDAP servers allow alternative "backend" mechanisms for storing entries, the same backend must be used for all entries in the DIT subtree.) However, restricting all attributes to the same information provider complicates the design of the MDS data-model. For example, the IP address associated with a network interface image can be provided by a system call, while the network bandwidth available through that interface is provided by a service such as the Network Weather Service (NWS) [23].

- **Client/server architecture**. The LDAP implementation requires at least one round-trip network communication for each LDAP access. Frequent MDS accesses thus becomes prohibitively expensive. We need a mechanism by which MDS data can be cached locally for a timely response.

- **Scope of Data**. The LDAP implementation assumes that any piece of information may be used from any point in the network (within the constraints of access control). However, a more efficient implementation of attribute update can be obtained if one can limit the locations from which attribute values can be accessed. The introduction of scope helps to determine which information must be propagated to which information providers, and when information can be safely cached.

Note that these drawbacks all relate to the LDAP implementation, not its API. Indeed, we can adopt the LDAP API for MDS without modification. Furthermore, for those DIT subtrees that contain information that is not adversely affected by the above limitations, we can pass the API calls straight through to an existing LDAP implementation. In general, however, MDS needs a specialized implementation of the LDAP API to meet the requirements for high performance and multiple information providers.

The most basic difference between our MDS implementation and standard LDAP implementations is that we allow information providers to be specified on a *per attribute* basis. Referring to the above example, we can provide the IP address of an interface via SNMP, the current available bandwidth via NWS, and the name of the machine into which the interface card is connected. Additionally, these providers can store information into MDS on a periodic basis, thus allowing refreshing of dynamic information. The specification of which protocol to use for each entry attribute is stored in an *object class metadata entry*. Metadata entries are stored in MDS and accessed via the LDAP protocol.

In addition to specifying the access protocol for an attribute, the MDS object class metadata also contains a time-to-live (TTL) for attribute values and the update scope of the attribute. The TTL data is used to enable caching; a TTL of 0 indicates that the attribute value cannot be cached, while a TTL of -1 indicates that the data is constant. Positive TTL values determine the amount of time that the attribute value is allowed to be provided out of the cache before refreshing.

The update scope of an attribute limits the readers of an updated attribute value. Our initial implementation considers three update scopes: process, computation, and global. Process scope attributes are accessible only within the same process as the writer, whereas computation scope attributes can be accessed by any process within a single computation, and global scope attributes can be accessed from any node or process on a network.

6 MDS Applications in Globus

We review briefly some of the ways in which MDS information can be used in high-performance distributed computing. We focus on applications within Globus, an infrastructure toolkit providing a suite of low-level mechanisms designed to be used to implement a range of higher-level services [9]. These mechanisms include communication, authentication, resource location, resource allocation, process management, and (in the

form of MDS) information infrastructure.

The Globus toolkit is designed with the configuration problem in mind. It attempts to provide, for each of its components, interfaces that allow higher-level services to manage how low-level mechanisms are applied. As an example, we consider the problem referred to earlier of selecting network interfaces and communication protocols when executing communication code within a heterogeneous network. The Globus communication module (a library called Nexus [10]) allows a user to specify an application's communication operations by using a single notation, regardless of the target platform: either the Nexus API or some library or language layered on top of that API. At run-time, the Nexus implementation configures a communication structure for the application, selecting for each communication link (a Nexus construct) the communication method that is to be used for communications over that link [7]. In making this selection for a particular pair of processors, Nexus first uses MDS information to determine which low-level mechanisms are available between the processors. Then, it selects from among these mechanisms, currently on the basis of built-in rules (e.g., "ATM is better than Internet"); rules based on dynamic information ("use ATM if current load is low"), or programmer-specified preferences ("always use Internet because I believe it is more reliable") can also be supported in principle. The result is that application source code can run unchanged in many different environments, selecting appropriate mechanisms in each case.

These method-selection mechanisms were used in the I-WAY testbed to permit applications to run on diverse heterogeneous virtual machines. For example, on a virtual machine connecting IBM SP and SGI Challenge computers with both ATM and Internet networks, Nexus used three different protocols (IBM proprietary MPL on the SP, shared-memory on the Challenge, and TCP/IP or AAL5 between computers) and selected either ATM or Internet network interfaces, depending on network status [8].

Another application for MDS information that we are investigating is resource location [22]. A "resource broker" is basically a process that supports specialized searches against MDS information. Rather than incorporate these search capabilities in MDS servers, we plan to construct resource brokers that construct and maintain the necessary indexes, querying MDS periodically to obtain up-to-date information.

7 Summary

We have argued that the complex, heterogeneous, and dynamic nature of high-performance distributed computing systems requires an *information-rich* approach to system configuration. In this approach, tools and applications do not rely on defaults or programmer-supplied knowledge to make configuration choices. Instead, they base choices on information obtained from external sources.

With the goal of enabling information-rich configuration, we have designed and implemented a *Metacomputing Directory Service*. MDS is designed to provide uniform, efficient, and scalable access to dynamic, distributed, and diverse information about the structure and state of resources. MDS defines a representation (based on that of LDAP), a data model (capable of representing various parallel computers and networks), and an implementation (which uses caching and other strategies to meet performance requirements). Experiments conducted with the Globus toolkit (particularly in the context of the I-WAY) show that MDS information can be used to good effect in practical situations.

We are currently deploying MDS in our GUSTO distributed computing testbed and are extending additional Globus components to use MDS information for configuration purposes. Other directions for immediate investigation include expanding the set of information sources supported, evaluating performance issues in applications, and developing optimized implementations for common operations. In the longer term, we are interested in more sophisticated applications (e.g., source routing, resource scheduling) and in the recording and use of application-generated performance metrics.

Acknowledgments

We gratefully acknowledge the contributions made by Craig Lee, Steve Schwab, and Paul Stelling to the design and implementation of Globus components. This work was supported by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523 and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc.*

4th IEEE Symp. on High Performance Distributed Computing. IEEE Computer Society Press, 1995.

[2] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96*. ACM Press, 1996.

[3] R. Butler and E. Lusk. Monitors, message, and clusters: The p4 parallel programming system. *Parallel Computing*, 20:547–564, April 1994.

[4] Henri Casanova and Jack Dongarra. Netsolve: A network server for solving computational science problems. Technical Report CS-95-313, University of Tennessee, November 1995.

[5] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The Network-Enabled Optimization System (NEOS) Server. Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.

[6] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide area visual supercomputing. *International Journal of Supercomputer Applications*, 10(2):123–130, 1996.

[7] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke. Managing multiple communication methods in high-performance networked computing systems. *Journal of Parallel and Distributed Computing*, 40:35–48, 1997.

[8] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY high-performance distributed computing experiment. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, pages 562–571. IEEE Computer Society Press, 1996.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997. To appear.

[10] I. Foster, C. Kesselman, and S. Tuecke. The Nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.

[12] A. Grimshaw, J. Weissman, E. West, and E. Lyot, Jr. Metasystems: An approach combining parallel processing and heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 21(3):257–270, 1994.

[13] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22:789–828, 1996.

[14] S. Heker, J. Reynolds, and C. Weider. Technical overview of directory services using the x.500 protocol. RFC 1309, FY14, 03/12 92.

[15] T. Howes and M. Smith. The ldap application program interface. RFC 1823, 08/09 95.

[16] M. Litzkow, M. Livney, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. 8th Intl Conf. on Distributed Computing Systems*, pages 104–111, 1988.

[17] G. Mansfield, T. Johannsen, and M. Knopper. Charting networks in the x.500 directory. RFC 1609, 03/25 94. (Experimental).

[18] B. Clifford Neumann and Santosh Rao. The Prospero resource manager: A scalable framework for processor allocation in distributed systems. *Concurrency: Practice & Experience*, 6(4):339–355, 1994.

[19] D. Reed, C. Elford, T. Madhyastha, E. Smirni, and S. Lamm. The Next Frontier: Interactive and Closed Loop Performance Steering. In *Proceedings of the 1996 ICPP Workshop on Challenges for Parallel Processing*, pages 20–31, August 1996.

[20] J. Reynolds and C. Weider. Executive introduction to directory services using the x.500 protocol. RFC 1308, FYI 13, 03/12 92.

[21] M. Rose. *The Simple Book*. Prentice Hall, 1994.

[22] Gregor von Laszewski. *A Parallel Data Assimilation System and Its Implications on a Metacomputing Environment*. PhD thesis, Syracuse University, December 1996.

[23] Richard Wolski. Dynamically forecasting network performance using the network weather service. Technical Report TR-CS96-494, U.C. San Diego, October 1996.

[24] W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol. RFC 1777, 03/28 95. Draft Standard.