# The Inca Test Harness and Reporting Framework

Shava Smallen † Catherine Olschanowsky † Kate Ericson †
Pete Beckman * Jennifer M. Schopf *

† San Diego Supercomputer Center
{ssmallen, cmills, kericson}@sdsc.edu

* Argonne National Laboratory
{beckman, jms}@mcs.anl.gov

## ABSTRACT

Virtual organizations (VOs), communities that enable coordinated resource sharing among multiple sites, are becoming more prevalent in the high-performance computing community. In order to promote cross-site resource usability, most VOs prepare service agreements that include a minimum set of common resource functionality, starting with a common software stack and evolving into more complicated service and interoperability agreements. VO service agreements are often difficult to verify and maintain, however, because the sites are dynamic and autonomous. Automated verification of service agreements is critical: manual and user tests are not practical on a large scale.

The Inca test harness and reporting framework is a generic system for the automated testing, data collection, verification, and monitoring of service agreements. This paper describes Inca's architecture, system impact, and performance. Inca is being used by the TeraGrid project to verify software installations, monitor service availability, and collect performance data.

## 1. INTRODUCTION

Production Grids have become prevalent in the high-performance computing community as a platform for running large-scale compute-intensive and data-intensive applications [35, 7, 31, 19, 17]. From an administrative perspective, the degree of coordination at the site and resource level can vary from one Grid to another. Some Grids are uncoordinated collections of resources, while others are more tightly coordinated through operational service agreements. The latter Grids and their management structures are known as *virtual organizations* (VOs) [9].

*VO service agreements* are created to describe the requirements for resource sharing and operational policies across VO resources as quantifiable properties. They can describe system availability, performance of basic services, or software stack validation and verification. For example, a service agreement can require a resource to provide a high-performance file system with specific minimum capabilities, Globus Toolkit GridFTP and gatekeeper services [8], or a BLAS [20] math library installation. These service agreements are made available to users so they can understand how to best utilize VO resources and what to expect when they log into such resources. VO service agreements are less formal than an industry service level agreement (e.g., network or Web hosting), which are binding contracts between providers and consumers [22]. Moreover, VO service agreements generally do not require that users be compensated after an agreement violation (such as a malfunctioning Globus Toolkit gatekeeper). VO service agreements help VOs provide more consistent and interoperable environments to end users. Unfortunately, service agreements are often difficult to implement in a VO because of site autonomy and differing administrative policies.

In order to assess the degree to which VO service agreements are being implemented consistently across sites, verification is required. Verification is accomplished by gathering data from each VO resource, comparing that data to the service agreement, and measuring compliance.

In this paper, we present the *Inca test harness and reporting framework*, a flexible system for automated verification of VO service agreements. Originally developed for the TeraGrid project [35], Inca is a general framework that can be adapted and used by other VOs.

Inca provides components for executing each step required for VO service agreement verification and a specification for expressing various types of data collected from resources. Inca includes mechanisms for scheduling the execution of information gathering scripts, as well as collecting, archiving, and publishing data. We show in this paper that Inca accomplishes these tasks with an average low system impact on VO resources.

The rest of this paper is structured as follows. Section 2 outlines the motivation for creating Inca and describes related work. Section 3 details Inca's current design and implementation. Section 4 presents an existing Inca deployment, and Section 5 evaluates Inca's system impact and performance. We conclude with a discussion of future work.

## 2. PROJECT MOTIVATION AND RELATED WORK

VOs prepare service agreements to promote cross-site interoperability. In general, a VO service agreement can include agreements with respect to the software stack, the user environment, a set of required services, guaranteed response time from the hardware, or an


0-7695-2153-3/04 $20.00 (c)2004 IEEE .

expected transfer time for large data sets. VO service agreements are broader in scope than typical network service-level agreements which usually deal directly with network quality of service and are tailored specifically for that use.

Efforts in monitoring service level agreements include the work done within the networking community [13] and as well as a growing body of work in the web services community [29]. Inca is a generic framework that enables the collection of a wide variety of data from resources. Data can be archived for performance characterization as well as used during agreement verification. Defining the scope and depth of service level agreements which will benefit the Grid community is still a work in progress.

In this paper, we use the term VO service agreements to distinguish this work from the already well developed term, service level agreements, which are common in industry.

## 2.1 Use Cases

The basic data collection and analysis framework provided by Inca supports diverse use cases.

- **Service Reliability** Most VOs deploy a set of persistent services for their users that are expected to be available 24/7. These include Grid tools such as the Globus Toolkit GRAM gatekeeper [8] or an SRB server [25], as well as SSH servers, and monitoring frameworks, such as the Network Weather Service [43]. These services may be susceptible to temporal bugs and external factors (e.g., misconfiguration). Therefore, it is important to periodically run test suites in order to discover problems before the users detect an interruption in service.

  Several Grids create a suite of tests for verifying local service availability when setting up a Grid. Two examples of Grid test suites are the NCSA TestGrid Project [34] and the UK Grid Integration Test Scripts [11]. Both are scripts that run a predefined set of tests and display the results on a Web page. However, these approaches lack the ability to easily change the set of defined tests or to provide multiple views that display the results.

- **Monitoring** In addition to basic service reliability, most VOs deploy a wide range of monitoring software. Cluster monitoring tools include Ganglia [21], CluMon [5], and BigBrother [2]. Each of these tools collects a set of data about cluster resources and displays it using a variety of client APIs and Web pages. Inca's data collection framework can be used to complement such monitoring tools and act as a uniform interface to multiple data sources.

  Grid monitoring tools used by many VOs include the Globus Toolkit's Monitoring and Directory Service (MDS2) [45], Condor's Hawkeye [12], the European Data Grid's R-GMA [26](based on the Global Grid Forum's Grid Monitoring Architecture [36]), GridLab's Testbed Status Monitoring Tool [14], SCALEA-G [38], and MonALISA [23]. A survey of other monitoring tools can be found in [10]. Although Inca's architecture is similar to such Grid monitoring tools, its goal is to collect and format data so that it can be compared to a VO service agreement and the result archived.

- **Benchmarking** Benchmarks help users understand how applications will perform on Grids. They also can be used to detect when there are hardware and software performance problems. For example, the TeraGrid deploys high-performance networking, data storage facilities, and clusters; data is needed to verify that the hardware and software is functioning at the expected capacity.

  Several projects including the GRASP project [4] and GridBench [39] have begun examining the need for common Grid benchmarks. As these benchmarks become more widely accepted, VOs can determine a service agreement that allows more in-depth verification of resource capabilities. Such benchmarks can easily be included as part of the set of tests run by Inca.

- **Site Interoperability Certification** As a Grid grows, it is likely to collaborate with similar projects. When two Grid projects wish to collaborate, they frequently have trouble verifying the compatibility of their project environments. That is, the two projects are likely to define a higher-level service agreement of common functionality between the projects but may have no way to verify it.

  Inca provides a flexible environment to certify compatibility between sites and projects. For example, a Grid can define a suite of tests for service agreement verification and run that suite on any other Grid where user-level access can be obtained. The test suite can test the requirements for a single application to run on a collaborating Grid or verify a larger set of requirements for inter-Grid compatibility and application porting.

- **Software Stack Validation** Software package release updates and patches are inevitable. Over the course of a project, many changes will be made to any initial software deployment. In many projects, software packages are deployed by local system administrators according to local constraints and requirements regarding update and installation procedures. Inca can be used to verify that the installation of new software and updates does not interfere with the existing environment. Specially designed test suites can be run directly after changes and before the system is made available to users to ensure a stable environment.

  The test suites are a combination of regression and integration testing commonly practiced in software projects. Several tools exist for automating this type of testing [6, 37] in a single software project. Inca is used to test interactions between several software packages installed on multiple production systems.

## 2.2 Service Agreement Verification Challenges

VO service agreements include many facets of the overall project and resources. The TeraGrid service agreement, for example, is designed to facilitate the development and execution of scientific applications. According to the current TeraGrid service agreement, participating sites must deploy a common user environment, the TeraGrid Hosting Environment, that includes a software stack and default user environment. The TeraGrid Hosting Environment allows users to develop applications for the common environment rather than for each site or resource independently.

Although service agreements can greatly benefit users, site autonomy and different administrative policies make such agreements difficult to implement in practice. Sites may interpret service agreements differently, and miscommunications may go undiscovered until users log into systems and find inconsistencies. It is therefore important to provide VO-level validation and verification [40] by measuring compliance to the service agreement through a set of predefined metrics. In the case of the TeraGrid, differences in the

interpretation of the service agreements clearly indicated that a tool was needed to verify the software stack and environment.

A site's service agreement compliance cannot be guaranteed throughout the life of a VO by one-time verification. Since Grid resources and service agreements change over time, ongoing validation is required to measure a site's continued compliance. Amplifying this requirement is the increased probability of failure because of the large number of complex components involved in Grid infrastructure and their interdependencies. Frequent verification provides quick notification of failures, enabling system administrators to respond immediately to problems as they are detected by the verification process, rather than reacting after users discover them.

## 2.3 Requirements

Inca was designed and developed to be a general framework for the automated verification of VO service agreements for the uses detailed above. The following requirements guided Inca's design.

- **Configurable Data Collection** The type and frequency of data collection may vary at the Grid or resource level. Hence, a fine granularity of control over content is required, implying that the process of adding and removing tests be simple and controlled on a per resource basis. Furthermore, the frequency of data collection must be configurable on a per test basis in order to accommodate the diverse nature of data collected.

- **Central Configuration** Changes to the data collected from a resource and its frequency of collection are inevitable. A central location for denoting these changes, as well as an automated mechanism for communicating them to participating resources, is needed.

- **Data Access** Data need to be accessible from a single access point in order to increase usability. In order to support a diverse set of data consumers, access should be made available through standard interfaces.

- **Persistent Data Storage** Archiving collected data provides a historical perspective on VO health and performance and aids in detecting performance problems.

- **Low System Impact** Data collection infrastructure requires a component to reside on the resource. The resource component cannot be invasive and must not impact a user's interaction with the system.

- **Scalability** Low system impact should be maintained as the number of resources and amount of data increase. Expecting data querying times to remain the same is unreasonable, but they should scale accordingly.

# 3. DESIGN AND IMPLEMENTATION

In order to encourage modularity and low system impact on resources, Inca is implemented by using a client-server architecture as depicted in Figure 1. The client components (distributed controllers and reporters) are designed to be lightweight and are installed on every VO resource. The server receives data from the distributed controllers and coordinates the scheduling and configuration of reporters; it is composed of the centralized controller, depot, and querying interface. Data consumers then access that data through the server, filter the data, compare it to an existing VO service agreement and visualize it in a meaningful way for specific user groups.

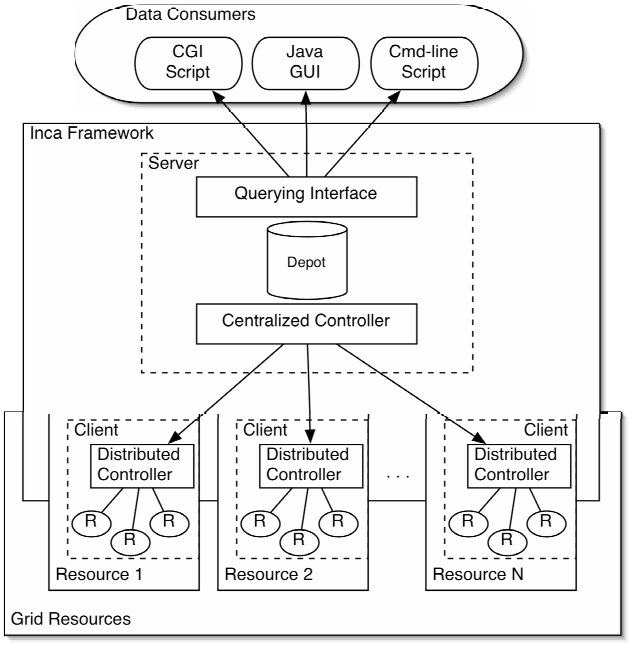



**Figure 1: Inca client/server architecture. The R in the figure represents reporter.**

## 3.1 Clients

The *clients* (reporters and distributed controller) interact directly with VO resources to gather data at a user level. In an effort to minimize system impact, the client functionality has been restricted to data collection. Resulting data is immediately forwarded to the server for processing.

### *3.1.1 User Account and Privileges*

Inca is designed to verify operability from a user perspective, therefore, the clients can and should be run from a regular user account and have no special privileges to the system. Often when administrators customize their environment to get software to function correctly, they unintentionally create a disparity between their functioning account and environment and the average user's. Therefore, it is important that the Inca user account be setup in the same manner as a default account given to a new user (i.e., with no customizations) in order to appropriately detect user problems. Furthermore, information needed to configure the reporters (e.g., Globus Toolkit gatekeeper contact strings) should come directly from the VO user guides and tutorials. In order to execute Grid reporters, the user account will need to be setup with a valid GSI [42] credential and be able to generate proxies for testing according to their VO's security policy.

### *3.1.2 Reporters*

A *reporter* interacts directly with a resource to perform a test, benchmark, or query. For example, a reporter can publish the version of a software package or perform a unit test to evaluate software functionality. Reporters do not control their execution schedule. Scheduling is directly controlled by the distributed controllers.

A reporter can be written in any language, but its output should be formatted in XML and follow the Inca reporter specification [15]. The reporter specification is designed to allow for the expression of a wide variety of data while providing enough structure to enable generic data handling. This is achieved by separating

```
<metric>
  <ID>bandwidth</ID>
  <statistic>
    <ID>upperBound</ID>
    <value>998.67</value>
    <units>Mbps</units>
  </statistic>
  <statistic>
    <ID>lowerBound</ID>
    <value>984.99</value>
    <units>Mbps</units>
  </statistic>
</metric>
```

**Figure 2: XML snippet describing a lower and upper bound on a bandwidth measurement.**

the results of the report into three sections: header, footer, and body.

The format for the header and footer are uniform across all reporters. A *header* provides metadata about the reporter, including the machine it ran on, the time at which it ran, and the input arguments supplied at run time. The *footer* contains an exit status indicating success or failure; if a failure is reported, a brief error message is required.

The variable portion of the report is a *body* that expresses the information collected by the reporter. For example, the body can be the version of a software package or measured network throughput to a machine such as the SRB [41] server. The schema for the body is open; there is not a set XML schema. Restrictions on tag formatting are enforced to enable generic data handling by the Inca framework. The most important restriction is that each branch of the XML document (i.e., an element containing other elements) have a unique identifier. With these unique identifiers, a unique path can then be constructed to locate any piece of data. For example, to locate the lower-bound bandwidth value in the XML snippet shown in Figure 2, one can use the path `value,statistic=lowerBound,metric=bandwidth`.

Inca includes an extensible set of Perl and Python APIs for reporter development. The APIs help developers to comply with the Inca reporter specifications, cut development time, and reduce duplicate code. Reporters created with the APIs are therefore typically small in size (less than 100 lines of code). Table 1 shows the size of reporters currently deployed to TeraGrid.

### *3.1.3 Distributed Controllers*

The *distributed controllers* are responsible for managing the execution of reporters on a resource and forwarding data to the Inca server. Distributed controllers are designed to receive execution instructions in the form of a specification file from the Inca server. In the current implementation this process is done manually. The specification file describes execution details for each reporter including frequency, expected run time, and input arguments. In order to distribute the impact of the reporter execution on a VO resource, reporters are scheduled to run at random times during their period. For example, a reporter executed hourly can be randomly chosen to run at the 20th minute of each hour, while another chosen to run on the 31st minute of each hour. The frequency of execution for a reporter is then expressed in the format of a cron table entry and can be configured on a per reporter basis. Since reporters can have differing impact on the system (e.g., a BLAS unit test will have more impact on the system than will a query for the version of Condor-G), it is important to be able to schedule reporters on different frequencies.

**Table 1: Reporter sizes for TeraGrid deployment (in lines of code).**

| Lines of Code | Number of Reporters |
|---|---|
| 0-50 | 106 |
| 50-100 | 9 |
| 100-150 | 7 |
| 150-200 | 1 |
| 200-250 | 1 |
| 300-350 | 1 |
| 450-500 | 1 |
| 1250-1300 | 1 |
| 1350-1400 | 1 |
| 1500-1550 | 1 |
| 1600-1650 | 1 |
| Total | 130 |

Each reporter also has a field, called a *branch identifier*, that indicates to the Inca server where the data should be stored. A branch identifier is a comma delimited list of name/value pairs similar to LDAP distinguished names [44]. The following example indicates where network performance data collected by the tool pathload running from `siteA` to `siteB` in `samplegrid` should be placed.

```
dest=siteB,tool=pathload,
performance=network,site=siteA,vo=samplegrid
```

The distributed controller is implemented as a Perl daemon with built-in cron capability (using the Schedule::Cron Perl library). When a reporter is scheduled to run, the daemon wakes up and forks off a process to execute it. The daemon also monitors all forked processes and terminates them if they exceed expected run time. The distributed controller communicates a report to the Inca server along with its branch identifier using a TCP connection. If there is an error executing a reporter, a special report is sent to the central controller to indicate an error.

## 3.2 Server

The Inca *server* handles coordination of clients as well as data collection and management. In the current implementation, the Inca server is a centralized component consisting of the centralized controller, depot, and querying interface.

### *3.2.1 Centralized Controller*

The *centralized controller* manages the dissemination of execution instructions to the clients and receives data from the distributed controllers and forwards this data to the depot. The current centralized controller is implemented as a Perl daemon and listens on a TCP port for incoming reports from the distributed controllers. We recommend that the centralized controller be run under a nonprivileged account. The TCP port number is configurable. When the centralized controller receives an incoming connection from a distributed controller, it checks the host against a list of hostnames to see whether it should accept the connection. If the host is verified, the centralized controller receives the report and branch identifier

from the distributed controller. It then creates a XML *envelope*, where the content of the envelope is the report and the envelope address is the branch identifier. The envelope is forwarded to the depot through a Web services interface for storage.

### *3.2.2 Depot*

The *depot* is Inca's facility for data management, caching and archiving. The design of the depot was driven by the need to require very little administration.

The most important feature of the depot is that new data with unknown schemas can be added to the cache with no configuration. Reports are received by the depot wrapped in an envelope. The depot uses the branch identifier from the envelope to identify a unique location for that report to be saved. Further updates of the report will result in the replacement of the previous copy. Since no additional configuration is needed, the effort to add new data, including new resources and reporters, is lowered.

The cache is implemented by using a SAX parser [3] and a single XML file. The SAX parser is used for both updates and queries to the cache. The initial design included the use of DOM parsing on the cache, but it was quickly discovered that the memory requirements of the DOM parser grew too rapidly with the size of the data, slowing the update and query times.

Archiving of numerical data is done by RRDTool [28]. In order to indicate that a piece of data is to be archived, an archival policy for that data must be uploaded to the depot. The archival policy describes the granularity of archiving (e.g., every fifth measurement) and the length of history to keep. This configuration has to be done only once and one can assign several pieces of data the same policy at the same time or can assign policies on a reporter-by-reporter basis. RRDTool is a scalable solution for archiving numerical data and supports a querying interface that is both fast and flexible.

### *3.2.3 Querying Interface*

The *querying interface* supports queries on both cached and archived data and will be optimized for common queries. In addition to filtering cached data to satisfy common user queries (e.g., by site, resource, software), the querying interface supports the temporal nature of archived data queries.

Querying the depot is currently split into two separate interfaces. One is for the retrieval of the most current data, which is held in the cache; the second is for graphing historical data from the archive.

Current data is requested through a web service call to the depot. If a branch identifier is supplied only the portion of the cache matching that identifier will be returned; this can either be a single report, a set of related reports, or a specific portion of a report. In the case that no branch identifier is supplied, the entire contents of the cache is returned. Depending on the cache size, the latter option tasks the data consumer with a large amount of XML processing, a process that can be time consuming.

Archived data is also retrieved through a Web service call, which wraps the interface provided by RRDTool.

## 3.3 Data Consumers

A *data consumer* queries the Inca server for data. Often, data consumers display the comparison of data stored at the Inca server to a machine-readable description of the service agreements and apply predefined metrics to express the degree of resource compliance. For example, a metric for measuring Grid service availability on a resource can be defined as follows: (1) at least one site can access the resource's Grid service, and (2) the resource can access at least one other site's Grid service.

Data consumers can be implemented as CGI scripts that visualize results through Web pages, an interactive Java GUI, or command-line scripts. An example of a data consumer in common use by the TeraGrid project is detailed in Section 4.

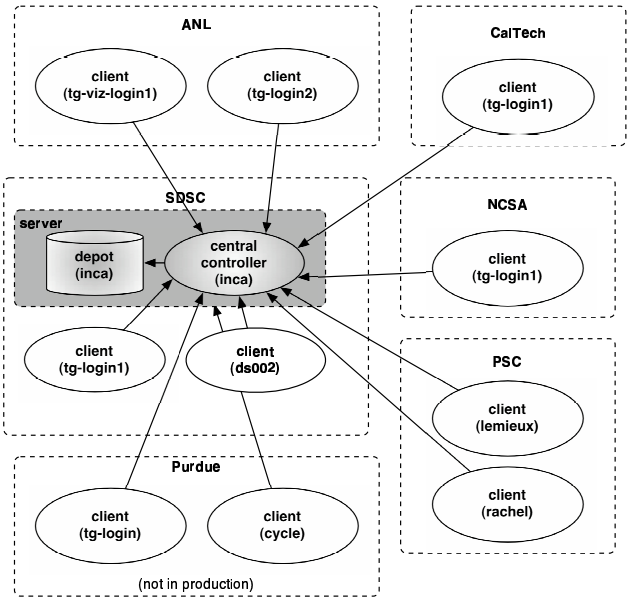



**Figure 3: Inca deployment to TeraGrid.**

# 4. DEPLOYMENT

This section describes the Inca deployment on TeraGrid, which has been in use for over a year. At the time this paper was written, the TeraGrid's production participants were Argonne National Laboratory (ANL), California Institute of Technology (Caltech), the National Center for Supercomputing Applications (NCSA), Pittsburgh Supercomputing Center (PSC), and San Diego Supercomputer Center (SDSC). These sites collectively provided 20 TF of compute power and 1 PB of data storage interconnected through a 40 Gb/s backbone. Section 4.1 describes how Inca is being used to verify the software and environment setup of the TeraGrid, and Section 4.2 describes performance data being collected through Inca.

## 4.1 TeraGrid Hosting Environment Verification and Validation

Every TeraGrid site is required to provide the TeraGrid Hosting Environment, a software stack, default user environment, and common methods for manipulating their environment through a tool called SoftEnv [30] to facilitate application development by providing a consistent environment at all of the TeraGrid sites. In order to verify the TeraGrid's service agreement, customized data consumers and reporters were developed.

Inca's TeraGrid deployment is illustrated in Figure 3. The Inca server components, the centralized controller, and the depot (within a Tomcat [33] server) were hosted at SDSC on `inca.sdsc.edu`. Inca's client components, reporters, and distributed controllers ran on ten resources at ANL, Caltech, NCSA, PSC, Purdue*, and SDSC. In order to detect user level problems, all client components ran under a default user account called inca.

In order to query the resources for compliance to the TeraGrid Hosting Environment, reporters were written to collect versions

*Purdue University is one of the recently added TeraGrid sites but is not yet in production.

**Table 2: Current number of Inca reporters executing per hour on TeraGrid systems.**

| Site | Machine | Number of Reporters |
|---|---|---|
| ANL | tg-viz-login1.uc.teragrid.org | 136 |
| | tg-login2.uc.teragrid.org | 128 |
| Caltech | tg-login1.caltech.teragrid.org | 128 |
| NCSA | tg-login1.ncsa.teragrid.org | 128 |
| PSC | rachel.psc.edu | 71 |
| | lemieux.psc.edu | 71 |
| Purdue | cycle.cc.purdue.edu | 128 |
| | tg-login.rcs.purdue.edu | 71 |
| SDSC | tg-login1.sdsc.teragrid.org | 128 |
| | dslogin.sdsc.edu | 71 |
| | Total | 1060 |

of installed packages and test package functionality. A reporter was also written to collect the set of environment variables in the default user environment and a resource's SoftEnv database. In addition, we deployed a set of cross-site tests to check for basic service availability including Globus Toolkit GRAM gatekeepers, GridFTP, OpenSSH, and SRB. The distributed controller's configuration, illustrating the number of reporters executed on each resource and their frequency of execution, is summarized in Table 2.

In order to visualize resource compliance to the TeraGrid Hosting Environment, a machine-readable version of the service agreement was formatted in XML. A resource's status is divided into three categories: Grid, Development, and Cluster. The Grid category comprises tests that verify the status of Grid packages such as the Globus Toolkit, Condor-G, GridFTP, and SRB; the Development category comprises tests that verify the status of libraries such as MPICH, ATLAS, HDF4, and HDF5; and the Cluster category comprises tests that verify the status of cluster-level packages such as the batch scheduler.

CGI scripts were written to compare the data collected from the resources to the service agreement and display the results in red/green status pages. For example, the data consumer in Figure 4 displays each resource's status in the form of percentages for each category described above and for all three categories combined. If there are errors, the test that has failed is listed and a URL is given to display the error message and the metadata for that test for debugging. Other status page formats are also provided, but we have found summary percentages to be the most effective in illustrating when an error is occurred. These summary percentages are archived and can be useful in illustrating the stability of resources. Figure 5 shows the Grid availability over a week's period for one of the TeraGrid's resources calculated every ten minutes. Mondays are preventative-maintenance days, so some drop in availability is expected but the other times indicate a system failure.

Another status page shows a detailed view of the software stack, listing the packages and status for each resource. Green indicates that an acceptable version of a software package is located on a resource and the unit tests pass; red indicates otherwise. Similar status pages are shown to display the default user environment and SoftEnv status. The cross-site test data is included as part of the unit tests defined for each software package. Currently, over 900 pieces of data are compared and verified.

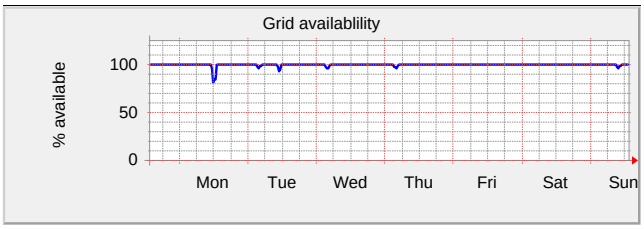



**Figure 5: Example of Grid availability on a TeraGrid resource.**

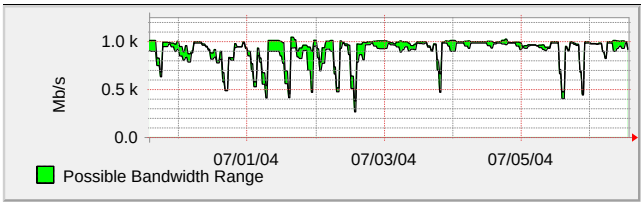



**Figure 6: Bandwidth data measured from Pathload running from SDSC to Caltech.**

## 4.2 TeraGrid Performance

In order to better understand how applications will perform on the TeraGrid, it is essential to run benchmarks on a periodic schedule to gather performance characteristics of TeraGrid. Furthermore, system updates such as recompiling the kernel or updating an Ethernet driver are subject to misconfiguration. Periodic benchmarks can be used to detect and diagnose performance problems when they occur. A reporter which executes the GRASP [4] benchmarks has been implemented and is currently collecting data. We have also implemented a number of network reporters that execute nonintrusive network monitoring tools such as Pathload [18], Pathchirp [27], and Spruce [32]. Figure 6 shows bandwidth measurements collected from the Pathload tool every hour from SDSC to Caltech.

# 5. IMPACT AND PERFORMANCE

In this section, we discuss Inca's impact on the monitored VO resources and analyze the Inca server performance. Table 3 displays the characteristics of the machines described in the following subsections. Section 5.1 discusses the system impact of the distributed controller, the client component of Inca installed on each VO resource. Section 5.2 discusses the performance of the depot, Inca's storage facility.

## 5.1 Distributed Controller System Impact

Recall from Section 3.1.3 that the distributed controller maintains and runs a schedule for reporter execution on a resource, forking a process to run a single reporter. In order to assess the impact of the distributed controller on VO resources, we monitored the TeraGrid deployment during the week of June 29–July 6. We logged the CPU and memory usage for the distributed controller running at Caltech every 10–11 seconds using the Unix system command top resulting in 57,149 measurements. Caltech's distributed controller executed 128 reporters every hour (from Table 2).

On average, the distributed controller's CPU usage was 0.02% per CPU and its memory usage was 35 MB of physical memory. Figures 7(a) and 7(b) show a more detailed view of the collected data distribution. Since the built-in cron capability of the reporter uses fork to run a process from its cron table, the memory usage

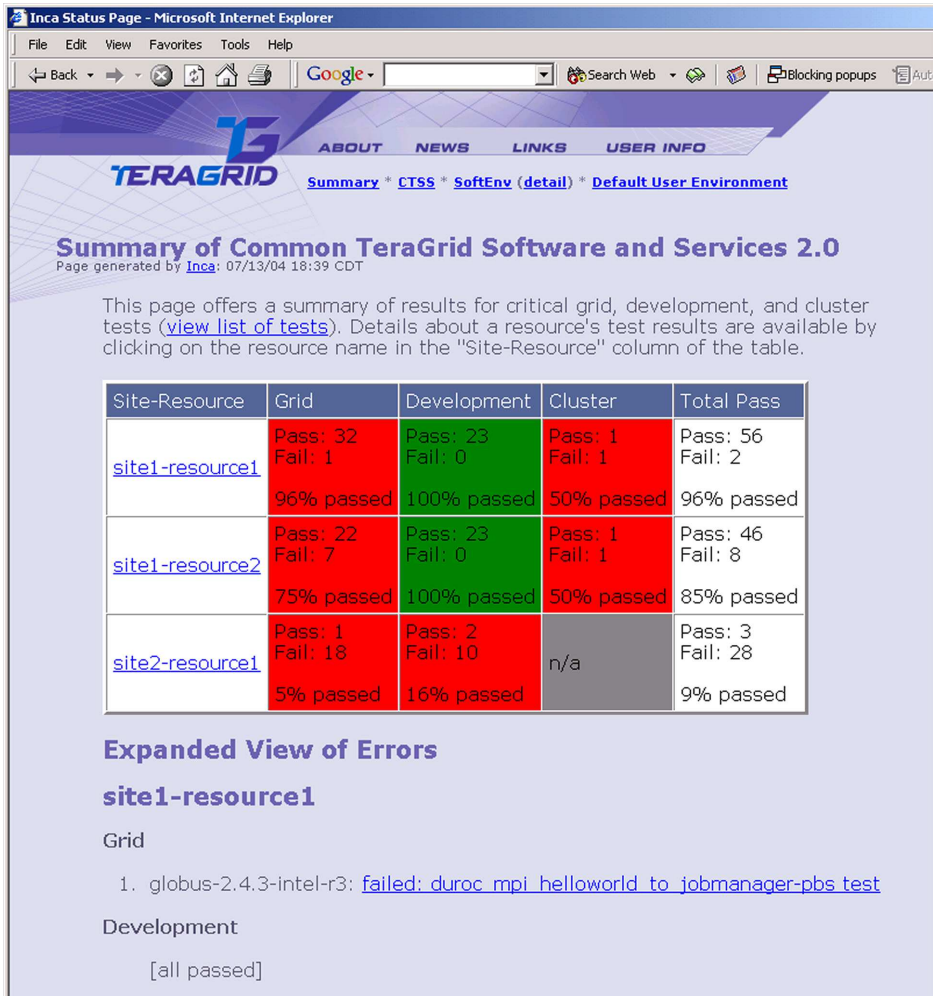



**Figure 4: The TeraGrid hosting environment status summary page.**

of the distributed controller was higher than expected. The average memory usage (35 MB) corresponds to the execution of the main controller process ( 18 MB) and one forked process (forked by Schedule::Cron). An unknown bug caused the memory usage to jump to 1 GB at one point because of a large number of forks in the controller. Memory usage of the distributed controller can be improved by enhancing the Schedule::Cron library to use a threaded implementation.

## 5.2 Depot Performance

The depot caches and optionally archives each report it receives. Recall that the depot receives envelopes from the centralized controller. The time that the centralized controller must wait while the depot receives and processes the envelope is referred to as the response time. The depot must be able to perform at a rate equal to or faster than the rate that reports are generated by the clients. Section 5.2.1 analyzes the depot response times in the current Inca TeraGrid deployment. In order to investigate the effects of larger deployments, we created synthetic deployments described in Section 5.2.2 that allowed us to analyze cache sizes greater than the existing TeraGrid deployment. We found that the response time depends on two factors: cache size and report size.

### *5.2.1 TeraGrid Results*

The TeraGrid depot was monitored for a week from July 7-July 14, 2004. The cache size remained steady at 1.5 MB. During the week, the depot received 151,955 reports from the centralized controller, at a mean rate of 15.07 reports per minute. As shown in Figure 8, 97.64% of the reports received were small, less than 10 KB. The amount of data received was 259.36 MB, at an average rate of 26.35 KB/min. Table 4 shows statistics on the response times for the TeraGrid deployment. On average, the response was quick enough to satisfy the needs of this deployment, but at times it jumped to very high numbers. The TeraGrid depot does not run on a dedicated machine and therefore contends with other processes for system resources.

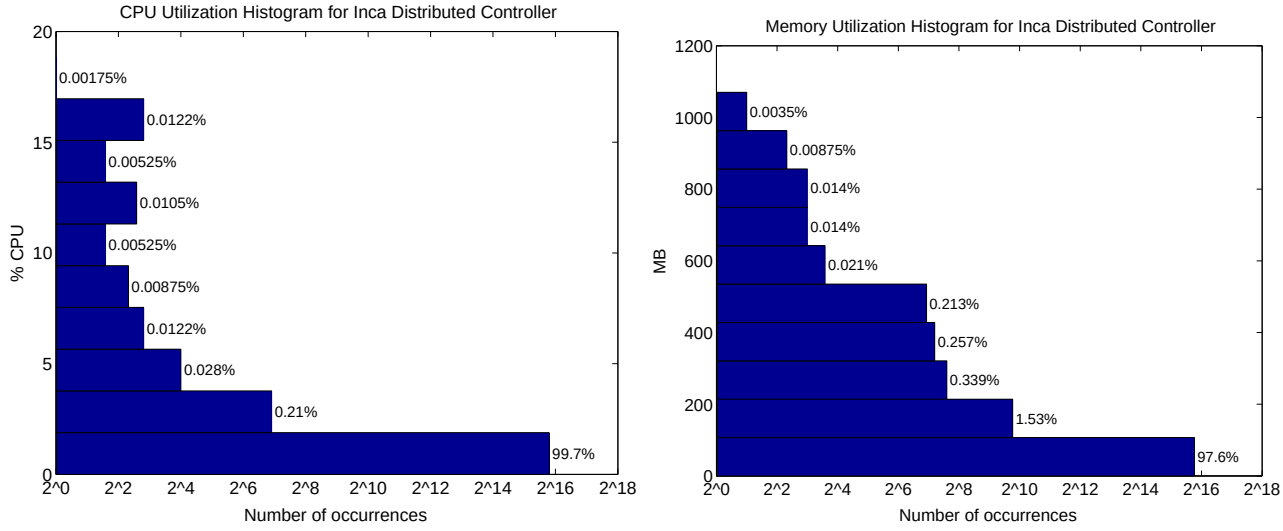
### *5.2.2 Synthetic Results*

To evaluate how well the depot handles cache sizes larger than the TeraGrid's, we ran a synthetic workload on a separate deployment. The deployment was similar to the TeraGrid's; the main difference was that only a single client was running. Because all requests are serialized through the centralized controller, multiple clients were emulated by a single client running at a higher frequency; this approach made it easier to control the frequency of updates. All components were run on `inca.sdsc.edu`.

The synthetic workload was created by using a simple reporter that read one of four premade reports and printed its contents to

**Table 3: Characteristics of the machines used in our impact and performance experiments.**

| Hostname | Num. CPUs | Processor Type | CPU Speed (MHz) | Memory (GB) |
|---|---|---|---|---|
| inca.sdsc.edu | 4 | Intel Xeon | 2457 | 2.0 |
| tg-login1.caltech.teragrid.org | 2 | Intel Itanium 2 | 1296 | 6.0 |




(a) (b)

**Figure 7: Horizontal histograms showing the CPU and memory utilization for the Inca distributed controller running at Caltech. Measurements were taken every 10–11 seconds over a week period resulting in 57,149 data points. Figure 7(a) shows that 99.7% of the time CPU utilization was less than 2% per CPU and Figure 7(b) shows that 97.6% of the time memory utilization was less than 107 MB.**

standard out. The four synthetic report sizes were 851, 9,257, 23,168, and 45,527 bytes. These file sizes are a sample of actual TeraGrid reporter sizes. A specification file controlled how often the reporter was run and which file it printed. This made it possible to control the size of the cache.

To examine the effects of both report size and cache size on response time, we varied the specification file to hold the cache size steady at 0.928 MB, 1.8 MB, 2.7 MB, 3.6 MB, 4.4 MB, and 5.4 MB. Each specification was run for a minimum of two hours.

Response time can be broken into two parts, (1) receiving the report and unpacking the SOAP envelope, which is done using Axis [1], and (2), processing the cache to find the appropriate location for the report. Figure 9 shows the time taken for the total response time and the time spent just on processing the cache and inserting the report. The total response time is the upper line for each cache size and the insertion time is the lower. As reports grow larger the amount of time spent unpacking the SOAP envelope (the area between the lines) increases significantly. Regardless of the size of the cache, it takes almost 3 seconds to unpack the SOAP envelope and get the largest report ready for addition to the cache.

The measurements taken on the depot indicate that some changes need to be made in order for Inca to scale to larger VOs. For example, the cache will be split into multiple smaller files to minimize XML parsing time and the reports will be sent as SOAP attachment rather than in the body of the SOAP envelope in order to speed up the unpacking process. These changes will improve the the depot response time. Improving response time will not significantly increase the depot's ability to service a large VO consisting of hundreds of resources. In order to address larger scalability concerns work has begun on distributing the depot functionality.

## 6. CONCLUSIONS AND FUTURE WORK

The verification of VO service agreements promotes consistency and stability across Grid resources. The Inca test harness presented in this paper is a flexible framework for performing service agreement verification. Inca has a low average impact on VO resources and offers customizable data collection scheduling and representation. In order to encourage site interoperability, Inca has been successfully deployed on the TeraGrid to verify its common software environment agreement, and over 900 pieces of information are already being verified. Inca is currently being distributed in NMI [24] releases and can be downloaded from our Web site [16]. We are also pursuing collaborations with other Grids in addition to our work with the TeraGrid project.

We plan to enable more advanced test scheduling, specifically allowing for dependencies. Other future work includes improved

**Table 4: Data gathered on the TeraGrid installation of the depot over a one-week period in July 2004.**

| | Inca Report Sizes | | | | | |
|---|---|---|---|---|---|---|
| Response Time Stats(secs) | 0-4 KB | 4-10 KB | 10-20 KB | 20-30 KB | 30-40 KB | 40-50 KB |
| mean | 1.46 | 1.43 | 1.79 | 1.98 | 2.14 | 2.91 |
| std | 1.96 | 1.78 | 2.29 | 2.27 | 1.98 | 2.36 |
| min | 0.01 | 0.66 | 0.01 | 0.87 | 0.87 | 1.41 |
| max | 11.18 | 10.00 | 10.02 | 10.01 | 10.00 | 10.01 |
| median | 0.82 | 0.87 | 0.95 | 1.14 | 1.48 | 1.96 |
| number of updates | 147861 | 512 | 1473 | 1234 | 132 | 383 |

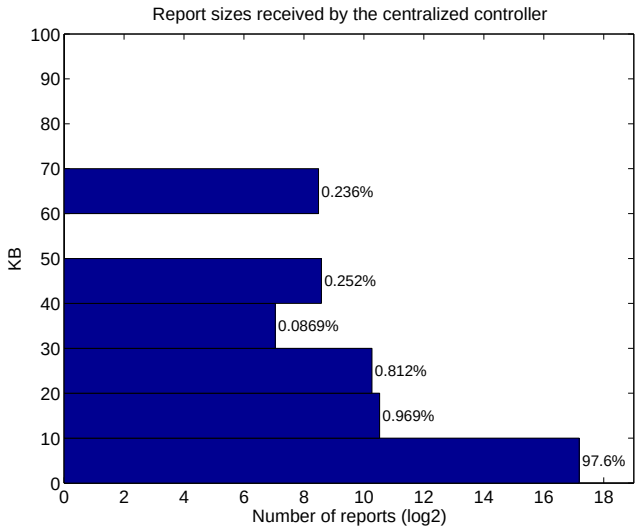



**Figure 8: Horizontal histogram showing the report sizes received by the centralized controller in the TeraGrid deployment.**

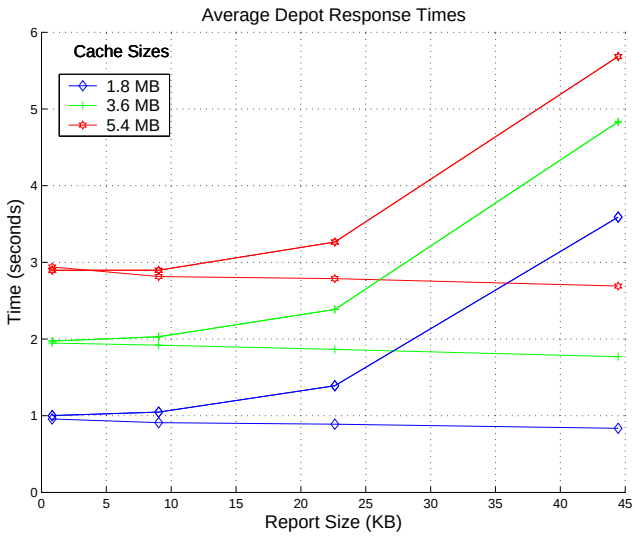



**Figure 9: Response times and XML processing time for the depot using a synthetic workload that varied the cache size and the reporter size. For each cache size the lower line is the amount of time spent on XML processing and the upper line is the total response time.**

security, additional user interfaces, automated reporter deployment, and improved data archival methods.

## 7. ACKNOWLEDGMENTS

The authors would like to thank Karan Bhatia and our reviewers for their insightful comments. Thanks also to Margaret Murray and Omid Okhalili for their assistance with the networking benchmark reporters and graph. This work was supported by NSF grant ACI-0122272 and in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38.

## 8. REFERENCES

[1] Apache axis project page. `http://ws.apache.org/axis`.

[2] Big Brother. `http://www.bb4.com`.

[3] D. Brownell. *SAX2* . O'Reilly & Associates, Inc., 2002.

[4] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark Probes for Grid Assessment. Technical Report CS2003-0760, University of California at San Diego, July 2003.

[5] Clumon Cluster Monitoring Web page. `http://clumon.ncsa.uiuc.edu`.

[6] Dart. `http://public.kitware.com/Dart/HTML/Index.shtml`.

[7] M. Ellisman and S. Peltier. Medical Data Federation: The Biomedical Informatics Research Network. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure, Second Edition*. Morgan Kaufmann, 2004.

[8] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[9] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[10] M. Gerndt, R. Wismller, Z. Balaton, G. Gombs, P. Kacsuk, Z. Neth, N. Podhorszki, H.-L. Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef. Performance Tools for the Grid: State of the Art and Future. Technical report.

[11] The UK Grid Integration Test Script - GITS. `http://www.soton.ac.uk/~djb1/gits.html`.

[12] Hawkeye. `http://www.cs.wisc.edu/condor/hawkeye`.

[13] U. Hofmann, I. Miloucheva, and T. Pfeiffenberger. Intermon: Complex QoS/SLA Analysis in Large Scale Internet Environment, 2004.

[14] P. Holub, M. Kuba, L. Matyska, and M. Ruda. Grid Infrastructure Monitoring as Reliable Information Service. In *2nd European Across Grids Conference*, 2004.

[15] The Inca Reporter Guide. `http://tech.teragrid.org/inca/www/documentation.html`.

[16] The Inca Home Page. `http://tech.teragrid.org/inca/`.

[17] International Virtual Data Grid Laboratory Web page. `http://www.ivdgl.org`.

[18] M. Jain and C. Dovrolis. Pathload: A Measurement Tool for End-to-End Available Bandwidth. *Proceedings of the 3rd Passive and Active Measurements Workshop*, March 2002.

[19] C. Kesselman, T. Prudhomme, and I. Foster. Distributed Telepresence: The NEESgrid Earthquake Engineering Collaboratory. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure, Second Edition*. Morgan Kaufmann, 2004.

[20] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Algorithm 539: Basic Linear Algebra Subprograms for Fortran Usage [F1]. *ACM Transactions on Mathematical Software*, 5(3):324–325, Sept. 1979.

[21] M. Massie, B. Chun, and D. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, April 2004.

[22] H. L. McKeefry. Service Level Agreements: Get 'Em in Writing. `http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2806173,00.html`, August 2001.

[23] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. MonALISA: A Distributed Monitoring Service Architecture . In *Proceedings of the Conference on Computing and High Energy Physics (CHEP)*, 2003.

[24] NSF Middleware Initiative Release 5. `http://www.nsf-middleware.org/NMIR5`.

[25] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky. Storage Resource Broker - Managing Distributed Data in a Grid. *Computer Society of India Journal, Special Issue on SAN*, 33(4):42–54, October 2003.

[26] R-GMA: Relational Grid Monitoring Architecture. `http://www.r-gma.org`.

[27] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. PathChirp: Efficient Available Bandwidth Estimation for Network Paths. *Passive and Active Measurement Workshop*, 2003.

[28] The Round Robin Database Tool Web page. `http://www.rrdtool.com`.

[29] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F. Casati. Automated SLA Monitoring for Web Services. In *IEEE/IFIP DSOM*, 2002.

[30] MCS Systems Administration Toolkit Web page. `http://www-unix.mcs.anl.gov/systems/software/msys`.

[31] A. K. Sinha, B. Ludaescher, B. Brodaric, C. Baru, D. Seber, A. Snoke, and C. Barnes. GEON: Developing the Cyberinfrastructure for the Earth Sciences - A Workshop Report on Intrusive Igneous Rocks, Wilson Cycle and Concept Spaces. `http://www.geongrid.org/workshops/conceptspace/igneous_rocks/workshop_report_intrusive_igneous_rocks.pdf`, 2004.

[32] Spruce home page. `http://project-iris.net/spruce`.

[33] Apache Tomcat Web page. `http://jakarta.apache.org/tomcat`.

[34] NCSA TestGrid Project. `http://grid.ncsa.uiuc.edu/test`.

[35] The TeraGrid Project Web page. `http://www.teragrid.org`.

[36] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. `http://www.ggf.org/documents/final.htm`.

[37] Tinderbox. `http://www.mozilla.org/tinderbox.html`.

[38] H. Truong and T. Fahringer. SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. In *2nd European Across Grids Conference*, 2004.

[39] G. Tsouloupas and M. Dikaiakos. GridBench: A Tool for Benchmarking Grids. In *Proceedings of the 4th International Workshop on Grid Computing*, pages 60–67, 2003.

[40] The Software Productivity Consortium's Verification and Validation Website. `http://www.software.org/pub/v&v`.

[41] M. Wan, A. Rajasekar, R. Moore, and P. Andrews. A Simple Mass Storage System for the SRB Data Grid. In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems & Technologies*, 2003.

[42] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *Proceedings of the 3rd Annual PKI R&D Workshop*, 2004.

[43] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999.

[44] W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol. RFC 1777, The Internet Engineering Task Force, 2000.

[45] X. Zhang, J. Freschl, and J. Schopf. A Performance Study of Monitoring and Information Services for Distributed Systems. In *Proceedings of HPDC-12*, 2003.