SDSC TR-2004-3

The Inca Test Harness and Reporting Framework

Shava Smallen
Catherine Olschanowsky
Kate Ericson

San Diego Supercomputer Center
University of California, San Diego

Pete Beckman
Jennifer Schopf

Argonne National Laboratory

San Diego Supercomputer Center
TECHNICAL REPORT

The Inca Test Harness and Reporting Framework

Shava Smallen [†] Catherine Olschanowsky [†] Kate Ericson [†]
Pete Beckman [*] Jennifer 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 a Service Level Agreement that includes a minimum set of common resource functionality, starting with a common software stack and evolving into more complicated service and interoperability agreements. VO Service Level Agreements are often difficult to verify and maintain, however, because the sites are dynamic and autonomous. Automated verification of service-level agreements is critical: manual and user tests simply are not practical on a large scale.

This paper presents the Inca test harness and reporting framework, a generic framework for the automated testing, verification, and monitoring of service-level agreements. Inca is currently being used by the TeraGrid project to verify software installations and to monitor service availability. This paper describes Inca's architecture, system impact, and preliminary scalability experiments.

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 [1, 2, 3, 4, 5]. 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 agreements. The latter Grids and their management structures are known as *virtual organizations* (VOs).

Service Level Agreements (SLAs) [6] express a VO's resource sharing and operational policies. A preliminary SLA is often a common software stack and can evolve into more complicated service and interoperability agreements. Agreement on an SLA can assist a VO in providing more consistent and interoperable environments to end users. Unfortunately, SLAs are often difficult to implement because of site autonomy and differing administrative policies.

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

In this paper, we present the *Inca test harness and reporting framework*, a flexible system to perform automated verification of SLAs. Originally developed for the TeraGrid project [1], Inca is a general framework that can be adapted and used by almost any multisite collaboration or Grid.

Inca provides components for executing each step required for SLA verification and a specification for expressing data that can be collected from resources. Components of Inca provide mechanisms to schedule the execution of scripts that query resources and collect, archive, and publish the resulting data. We show in this paper that Inca accomplishes these tasks with low system impact.

The rest of this paper is structured as follows. The next section outlines the motivation for creating Inca. Section 3 describes Inca's current design and implementation, and Section 4 describes an existing Inca deployment. Section 5 evaluates Inca's system impact and scalability, Section 6 offers additional uses for the Inca framework, and Section 7 compares Inca to related work. We conclude with a discussion of future work.

2 Project Motivation

VOs prepare *Service Level Agreements* (SLAs) to promote cross-site interoperability. The TeraGrid SLA, for example, is designed to facilitate the development and execution of scientific applications. According to the TeraGrid's SLA, participating sites are required to 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 target application development to the common environment rather than to each site or resource independently.

Although SLAs can greatly benefit users, because of site autonomy and different administrative policies they are difficult to implement in practice. Sites may interpret SLAs 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 [7] by measuring compliance to the SLA through a set of predefined metrics. In the case of the TeraGrid, differences in the interpretation of the SLA clearly indicated that a tool was needed to verify the software stack and environment.

A site's SLA compliance cannot be guaranteed throughout the life of a VO by one-time verification. Since Grid resources and SLAs change over time, ongoing validation is required to measure a site's continued SLA 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 and periodic 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.

2.1 Requirements

Inca was designed and developed to be a general framework for the automated verification of SLAs. In this section, we present the requirements that 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 needs 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. Maintaining an average of less than 1% of the CPU will not affect users.
- **Scalability**: As the number of resources and amount of data increase, low system impact should be maintained. 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 lightweight and installed on every VO resource. The server collects data from the distributed

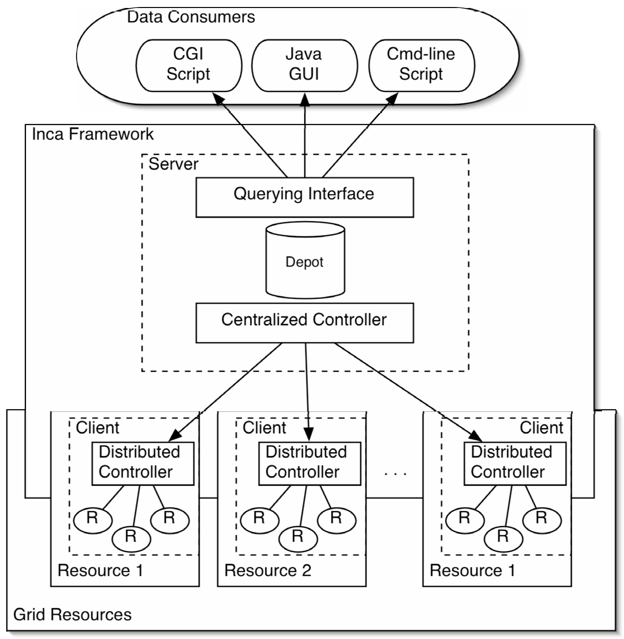


Figure 1. Inca is implemented as a client/server architecture. The R in the figure represents reporter.

controllers and coordinates the configuration of reporters; it is composed of the centralized controller, depot, and querying interface. Data consumers can then access that data through the server, filter the data, and visualize it in a meaningful way for specific user groups.

3.1 Clients

The *clients* (reporters and distributed controller) interact directly with VO resources to gather data. 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 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. 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 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 [8] 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.

To facilitate reporter development, Inca includes an extensible set of Perl and Python helper APIs that generate compliant XML.

3.1.2 Distributed Controllers

The *distributed controllers* are responsible for managing the execution of reporters 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; automation is the subject of future work. The specification file describes execution details including frequency, expected run time, and input arguments.

The distributed controller is implemented as a Perl daemon with built-in cron capability. 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. Motivated by system impact results described in Section 5, we plan to move the distributed controller to a multithreaded model.

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. In order to satisfy scalability issues, server components will need to be distributed. Redesigning and implementing components of the server to improve scalability are the subject of future work.

3.2.1 Centralized Controller

The goal of the *centralized controller* is to manage the dissemination of execution instructions to the clients and to receive data from the distributed controllers and forward this data to the depot. A partial implementation of the centralized controller that handles forwarding data to the depot has been completed. This centralized controller is implemented as a Perl daemon and listens on a well-known port for incoming reports from the distributed controllers. After processing a report, it uses a Web services interface to forward the information to the depot. Work continues on automating the communication of execution instructions to clients.

3.2.2 Depot

The *depot* is Inca's facility for data management. Currently, it supports caching and archiving of numerical data. The reporter structure allows new reports and report types to be ingested without additional configuration. This lowers the effort required to add new data, including the addition of new resources and reporters. In the current implementation, the depot is implemented as a Java Web service. Caching is accomplished by storing data in a XML document. Insertion and replacement of reports are handled through a SAX parser [9]. Archiving of numerical data is done by RRDTool [10]. Although RRDTool is a scalable solution, Inca requires a more general solution. Generic data archiving at this scale is challenging, and the investigation of appropriate solutions is ongoing.

3.2.3 Querying Interface

A *querying interface* is planned to satisfy the requirement from Section 2.1 that data be available from a single access point. The interface will be designed to support 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 will support the temporal nature of archived data queries.

In the current implementation of Inca, the data consumer filters cached data after it receives a dump of the entire cache directly from the depot. Archived data is retrieved through a Web service, which wraps the interface provided by RRDTool. Because of the wide acceptance of Grid services [11, 12], we also plan to expose the functionality of the querying interface through a Grid service.

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 SLA 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. The next section describes a data consumer in use today by TeraGrid.

4 Deployment

This section describes the Inca deployment on TeraGrid. At the time this paper was written, TeraGrid's production participants included Argonne National Laboratory (ANL), California Institute of Technology (Caltech), the National Center for Supercomputing Applications (NSCA), 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.

As described in Section 2, every site of TeraGrid 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 [13]. The goal is to facilitate application development by providing a consistent environment at all of the TeraGrid sites. In order to verify the TeraGrid's SLA, customization was needed at the reporter level to collect data from TeraGrid resources and at the data consumer level to visualize SLA compliance.

Inca's TeraGrid deployment is illustrated in Figure 2. The Inca server components, the centralized controller and depot (within a Tomcat [14] server), were hosted at SDSC on `griddle.sdsc.edu` and `grid-devel.rocksclusters.org` respectively. Inca's client components, reporters and the distributed controllers, ran on eight resources at ANL, Caltech, NCSA, PSC, 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 of installed packages and test package functionality. A reporter was also written to

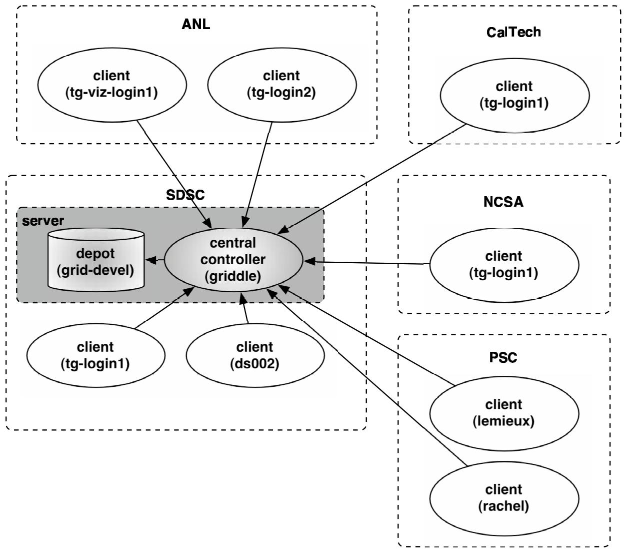
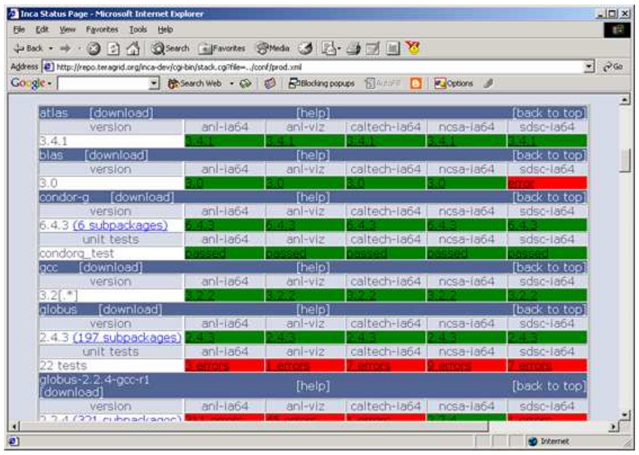


Figure 2. Inca deployment to TeraGrid.

collect the set of environment variables in the default user environment and a resource's SoftEnv database. A summary of the distributed controller's configuration, illustrating the number of reporters executed on each resource and their frequency of execution, is provided in Table 1. In order to distribute reporter execution, each reporter was scheduled to run at a random time during its cycle. For example, one hourly reporter was randomly chosen to run at the 20th minute of each hour, while another was randomly chosen to run on the 31st minute of each hour. In the future, we plan to implement integrated reporters to test interpackage functionality, eventually wrapping representative applications. We also plan benchmark reporters to detect performance problems.

In order to visualize resource compliance to the TeraGrid Hosting Environment, a machine-readable version of the SLA was formatted in XML. CGI scripts were written to compare the data collected from the resources to the SLA and display the results in red/green status pages. For example, Figure 3 shows the software stack status page, detailing the specific list of packages in the software stack: green indicates 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. In this setup, over 900 pieces of data are being compared and verified.


Inca Status Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://repo.teragrid.org/inca-dev/cgi-bin/stack.cgi?file=../conf/prod.xml
Go
Google Search Web Blocking popups Options
Internet

atlas [download]	[help]				[back to top]
version	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
3.4.1	3.4.1	3.4.1	3.4.1	3.4.1	3.4.1
blas [download]	[help]				[back to top]
version	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
3.0	3.0	3.0	3.0	3.0	error
condor-g [download]	[help]				[back to top]
version	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
6.4.3 (6 subpackages)	6.4.3	6.4.3	6.4.3	6.4.3	6.4.3
unit tests	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
condorg_test	passed	passed	passed	passed	passed
gcc [download]	[help]				[back to top]
version	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
3.2[.*]	3.2.2	3.2.2	3.2.2	3.2.2	3.2.2
globus [download]	[help]				[back to top]
version	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
2.4.3 (197 subpackages)	2.4.3	2.4.3	2.4.3	2.4.3	2.4.3
unit tests	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
22 tests	9 errors	1 errors	7 errors	9 errors	7 errors
globus-2.2.4-gcc-r1 [download]	[help]				[back to top]
version	anl-ia64	anl-viz	caltech-ia64	ncsa-ia64	sdsc-ia64
2.2.4 (321 subpackages)	[illegible]	[illegible]	[illegible]	2.2.4	[illegible]

Figure 3. The TeraGrid hosting environment status monitor.

5 Scalability and Impact Analysis

Inca's success greatly depends on its scalability. For example, reporters must be small and require low system utilization in order for a large number of them to run simultaneously. The centralized and distributed controllers have similar system utilization requirements. Scalability for the depot implies low response time. In this section we detail our experiments that examine Inca client overhead and depot scalability.

5.1 System Impact

One of Inca's requirements (described in Section 2.1) is for its client components to have low system impact on its monitored resources. In this section, we study the client's system impact within the context of the Inca TeraGrid deployment described in the preceding section. We also analyze Inca's server components.

5.1.1 Experimental Setup

To study the impact of Inca's client and server components, we monitored its deployment on TeraGrid during the week of March 8–15, 2004, logging the CPU and memory usage for the central controller, depot, and a single distributed controller at Caltech every 10 seconds using the Unix system command `top`. System impact results are summarized in Table 2, and the machines on which they ran are described in Table 3 [*].

[*]The number of measurements used to calculate the system usage of the client components may not be precisely a week's worth of data. Our logs showed evidence of clock skew on the

Machine	Reporter Frequency			
	1 hour	4 hours	12 hours	total
tg-login2.uc.teragrid.org	5	105	50	160
tg-viz-login1.uc.teragrid.org	5	112	50	167
tg-login1.caltech.teragrid.org	5	105	50	160
tg-login1.ncsa.teragrid.org	5	105	50	160
rachel.psc.edu	4	63	36	103
lemieux.psc.edu	4	63	36	103
ds002.sdsc.edu	4	63	36	103
tg-login1.sdsc.teragrid.org	5	105	50	160
Total	37	721	358	1116

Table 1. The current number of Inca reporters on TeraGrid systems by frequency of execution.

Usage	Inca Components			
	distributed controller (tg-login1)	reporters (tg-login1)	centralized controller (griddle)	depot (grid-devel)
% CPU				
mean	0.02	1.59	0.35	9.22
std	0.62	10.68	1.31	23.60
min	0.00	0.00	0.00	0.00
max	99.90	200.00	35.90	201.50
median	0.00	0.00	0.00	0.00
Memory (MB)				
mean	38.86	37.23	13.67	144.01
std	38.57	40.71	0.58	6.69
min	6.05	0.00	13.00	132.00
max	852.53	628.82	15.00	158.00
median	18.14	18.14	14.00	148.00

Table 2. TeraGrid CPU and memory use by Inca's controllers, reporters, and depot during the week of March 8-15, 2004.

5.1.2 Results

Recall from Section 3.1.2 that the distributed controller maintains and runs a schedule for reporter execution on a resource, forking a process to run a single reporter. Table 1 shows that Caltech's distributed controller executed 5 reporters every hour, 105 reporters every 4 hours, and 50 reporters every 12 hours. As shown in Table 2, the distributed controller's CPU usage was low, averaging 0.02%, fulfilling our low system impact requirement described in Section 2.1. Memory usage averaged 39 MB (0.64% of physical memory). Reporters executed by the distributed controller were more resource-intensive, consuming an average of 1.59% of the CPU and 38 MB (0.62% of physical memory). When heavyweight reporters were executed, CPU utilization was high.

order of 2–5 minutes, but we were unable to access the machine's logs to accurately adjust the client log timestamps. These results will be re-executed in the final version of the paper.

Recall that the system impact of reporters is configurable by varying the frequency of execution. In cases where heavyweight reporters are executed, it is important to execute them less frequently to amortize their impact on the system. One area of future work is to provide a tool that will estimate the system impact of a reporter execution schedule on a resource, so that the system administrator can trade-off freshness of data for system impact.

To better illustrate the load on the centralized components of Inca, we logged a report's size and the time it was received by the centralized controller. During the week, the centralized controller received 25,002 re-

Hostname	Num. CPUs	Processor Type	CPU Speed (MHz)	Memory (GB)
griddle.sdsc.edu	1	Intel Pentium 4	1816	0.5
tg-login1.caltech.teragrid.org	2	Intel Itanium 2	1296	6.0
grid-devel.rocksclusters.org	4	Intel XEON	2193	1.0
remington.rocksclusters.org	2	Pentium III (Coppermine)	731	0.5
compute-0-0	1	Pentium III (Coppermine)	731	0.5

Table 3. This table details characteristics of the system nodes used in our scalability and impact experiments. Impact tests were conducted on `griddle`, a single node system hosting a centralized controller and depot, and Caltech's login node, `tg-login1`, with a distributed controller. The scalability experiments were performed on a single node with another Inca depot, `grid-devel`, and two `remington` nodes, one with a centralized controller and the other with a distributed controller.

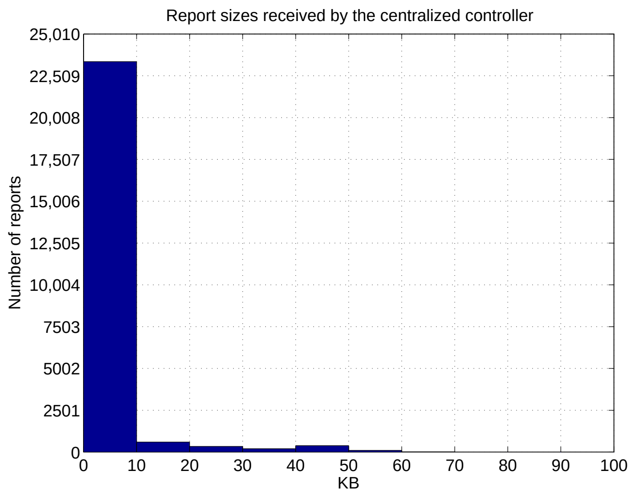


Figure 4. Histogram showing the report sizes received by the centralized controller in the TeraGrid deployment. Results show that 94.2% of the reports were less than 5 KB.

ports from all of the distributed controllers, at a mean rate of 2.48 reports per minute. As shown in Figure 4, 94.2% of the reports received were small, less than 5 KB. The amount of data received was 75 MB, at a mean rate of 7.66 KB/min. Recall from Section 3.2.1 that the centralized controller is responsible for collecting reports from the distributed controller and forwarding them to the depot. As shown in Table 2, the centralized controller is a lightweight process that used an average of 0.35% CPU and an average of 14 MB (2.59% of physical memory). In contrast, the depot is a more heavyweight process. The depot used an average of 9.22% of the CPU (standard deviation of 23.60%) and 144 MB (14% of physical memory). The depot requires more CPU and memory because it is currently implemented as a Java Web service, performs a large amount of XML processing, and has the overhead associated with running inside a Tomcat container.

5.2 Depot Scalability

To be considered scalable, the depot must be able to process both updates and queries in a timely manner. Update time depends on three factors: cache size, report size, and the location of the data in the cache. We designed experiments to investigate the effects of cache size and reporter size on depot scalability. The depot was tested using Inca's actual TeraGrid load, as well as a synthetic load used to explore the effects of cache sizes greater than those in the existing deployment.

5.2.1 TeraGrid Results

Figure 5 shows the update times for the TeraGrid deployment. Reporter size ranged from 753 bytes to 61 KB as illustrated in Figure 4, and the cache size remained steady around 1.5 MB. The minimum update time varied with reporter size. The large variation in update time is due to the report location in the cache. The final version of the paper will include additional graphs illustrating this phenomenon. The appropriate location for placing the report is found by processing the cache from the beginning until that location is found. This causes the update time to be largely dependent on that location, rather than on report size or cache size.

5.2.2 Synthetic Results

To evaluate how well the depot handles cache sizes larger than the TeraGrid's, we ran a synthetic workload

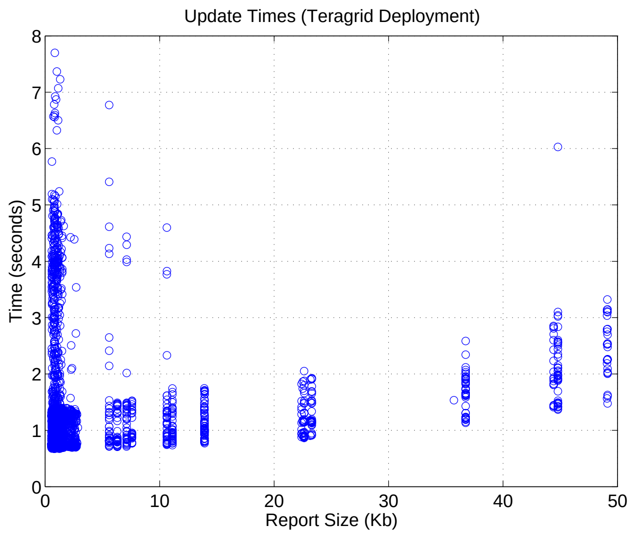


Figure 5. Update times for the TeraGrid deployment of Inca.

on a separate deployment. The deployment was similar to the TeraGrid's; the main difference was that only a single client running. Because all update requests were 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 of the resources were similar to those used by the TeraGrid. The centralized controller was run on one node of `remington.rocksclusters.org`, the distributed controller was run on the `compute0-0` node of the same cluster, and the depot was run on a faster machine (`grid-devel.rocksclusters.org`). Specifications for `remington` and `grid-devel` nodes are outlined in Table 3.

The synthetic workload was created by using a simple reporter that read one of four premade reports and printed its contents to standard out. The four synthetic report sizes were 851, 9257, 23168, and 45527 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 update times, we varied the specification file to hold the cache size steady at 0.9 MB, 1.8 MB, and 2.7 MB. Each specification was run for a minimum of two hours. For the final paper, these experiments will run for longer time periods.

Results for the synthetic tests were nearly identical to the TeraGrid test results. Update times also showed a dependence on the size of the report. Variations of total cache size did not have a significant effect on update times. The location of the report within the cache again caused large variations in the update times for same-size reporters.

Update times for both deployments were longer than expected. To determine which part of the update took the most time, we added extra timings to the depot code to evaluate the processing overhead (the time used for XML processing and file I/O). We found that as report size increased, processing time was dominated by the DOM [15] parser as it verified reports before adding them to the cache. The parsing of the cache document, which is much larger than the incoming report, took only a fraction of the time used for verification. The verification process can be changed to use a SAX parser, which will greatly reduce the processing time.

In order for the depot to scale to larger amounts of data, it is necessary to remove the dependency between cache location and update time and to minimize the relationship between report size and update time. One way to achieve these goals is to process updates asynchronously. This will also allow us to batch updates and reduce depot overhead.

6 Additional Usage Scenarios

Although the TeraGrid project uses Inca primarily for software and service verification, it is a general framework capable of supporting a variety of usage scenarios. Because Inca's client/server implementation is modular, it can be used to collect different types of data and flexibly schedule data collection. Inca can be employed to gather a wide variety of system data, from software and services data to benchmarks and performance data. For example, Inca can periodically schedule the execution of benchmarks, and its performance data can be archived and published. Inca is capable of executing cluster-level benchmarks, Grid-level benchmarks such as GRASP [16], or network benchmarks.

Inca's framework can also be adapted to collect data according to different scheduling mechanisms. Specifically, we anticipate that self-scheduling data collection mechanisms could communicate directly with the Inca server. For example, event-driven (e.g., triggered on machine reboot) data collection may be more appropriate for some types of data.

Inca has driven the development of a large set of valuable unit tests. These tests have a variety of applications.

- After upgrades and additions to the software stack, the unit test suite can be used to perform regression testing.

- A comprehensive unit test suite provides new resources with a well-defined specification and a mechanism for measuring their SLA compliance as they move toward production.

- Through an appropriate user interface, users and support personnel can employ the unit test suites as a debugging tool. A prototype interface has been completed, but it is not yet publicly available.

7 Related Work

Inca is unique in that it uses the data it gathers to measure whether an SLA is being satisfied. The following systems overlap with Inca's goal or infrastructure.

One use of Inca is to test Grid service availability. Both the NCSA TestGrid Project [17] and GITS [18] share this goal. Implemented as script, they run a predefined set of tests (code within the script) and display the results on a Web page. The set of tests run within these scripts are valuable. However, a system such as Inca is better suited to handle changes in the SLA, development of multiple interfaces to collected data, and the addition of new resources.

A part of Inca's architecture, the server, can be considered an information service. This functionality overlaps with the Monitoring and Directory Service (MDS2) [19]. MDS2 collects information about Grid resource characteristics and availability for system administrators and users. MDS2 is currently implemented by using a pull model when the data is requested. In contrast, Inca requires a push model in order to ensure that reporters are run on a regular interval. This is important both for detecting errors and for maintaining a coherent archive.

MonALISA [20] provides Grid-level monitoring by aggregating data from system usage and cluster monitoring systems. MonALISA is useful for gathering resource usage statistics at the Grid-level from self-scheduling providers. However, it does not have the capability to schedule data collection, one of Inca's key requirements.

At a high level it may seem that there is overlap between cluster monitoring systems [21, 22, 23] and Inca because they both collect information from resources. For example, Ganglia [21] provides comprehensive, low-level monitoring information on all nodes in a cluster. But Inca's goal is not to gather this granularity of data. From the viewpoint of the goals and intended use, Inca and cluster monitoring systems operate at different levels and are in fact complementary.

8 Future Work

Through the use of Inca in the TeraGrid project, we have identified areas in need of further development and improvement. These areas include additional user interfaces, ordered execution of reporters, automated configuration, and improved data archival methods. Each of these is discussed further below.

Most of this paper has focused on the detection of system-level problems. Another use of the reporters developed for the TeraGrid is to troubleshoot user-level problems (e.g., to detect account setup problems). A Web-based prototype interface for enabling users to execute reporters under their own account has been completed, and further work will be done to make this publicly available. Reporters also serve as example code and be a learning resource for users.

Grid middleware packages often have several software dependencies. When when one service goes down, the functionality of other software may also be affected. One way to better detect the source of system failures is to enable ordered scheduling of reporters, such that the execution of one reporter could depend on the successful completion of another. Particularly, if a core Grid service fails, executing reporters which test software that has a dependency on the core Grid service is redundant. Handling reporter dependencies would enable more efficient scheduling and improved GUI interfaces to visualize the failures.

As a VO expands to include more resources, there needs to be a central location to trigger configuration changes when the SLA is changed. Automating execution instructions to be disseminated as described in Section 3.2.1 will make the management of an Inca installation more scalable.

The current solution for archiving data is targeted to numerical data. While this solution stores useful historical information, logging of error messages and other textual information would also be useful. Enabling richer archiving capabilities in a scalable fashion and providing flexible querying interfaces are challenging tasks, and we will continue work in this area.

9 Conclusion

The verification of VO Service Level Agreements promotes consistency and stability across Grid resources. The Inca test harness presented in this paper is a flexible framework for performing SLA verification. Because of its modular design, Inca has a low impact on resources and offers customizable data collection scheduling and representation. In order to encourage site interoperability, Inca has been successfully deployed to TeraGrid to verify its common software environment SLA, and over 900 pieces of information have already been verified. We are now pursuing collaborations with other Grids in addition to our work with the TeraGrid project.

Acknowledgements

This work was supported by the Distributed Terascale Facility project 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.

References

[1] The TeraGrid Project web page. `http://www.teragrid.org`.

[2] 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* . Morgan Kaufmann, second edition, 2004.

[3] 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_r%eport_intrusive_igneous_rocks.pdf`, 2004.

[4] 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* . Morgan Kaufmann, second edition, 2004.

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

[6] K. Czajkowski, I. Foster, C. Kesselman., V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems . In *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 153–183. Springer-Verlag, 2002.

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

[8] 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.

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

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

[11] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration . Open grid service infrastructure wg, Global Grid Forum, 2002.

[12] The WS-Resource Framework. White Paper, March 2004.

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

[14] Apache Tomcat web page. `http://jakarta.apache.org/tomcat`.

[15] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model (DOM) Level 2 Core Specification. W3c recommendation, W3C, 2000.

[16] 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.

[17] NCSA TestGrid Project.

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

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

[20] 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.

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

[22] Clumon Cluster Monitoring web page . `http://clumon.ncsa.uiuc.edu`.

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