User-level Grid Monitoring with Inca 2

Shava Smallen, Kate Ericson, Jim Hayes, Catherine Olschanowsky
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093-0505, USA
{ssmallen,kericson,jhayes,cmills}@sdsc.edu

Abstract

The primary goal in the creation of Grids is to provide unified and coherent access to distributed computing, data storage and analysis, instruments, and other resources to advance scientific exploration. Grids combine multiple complex and interdependent systems that span several administrative domains. This complexity poses challenges for both the administrators who build and maintain the Grid resources and the scientists who use them. While other Grid monitoring tools provide system-level information on the utilization of Grid resources, the Inca system provides user-level Grid monitoring with periodic, automated user-level testing of the software and services required to support Grid operation. Inca can be used by Grid operators, system administrators, and application users to identify, analyze, and troubleshoot user-level Grid failures, thereby improving Grid stability. In this paper, we describe the new features of our current Inca release, Inca 2. We then describe the architecture of the Inca 2 system, in addition to use cases that describe two Inca 2 deployments in production environments.

1. Introduction

Grid systems provide unified and coherent access to distributed computing, data storage and analysis, instruments, and other resources. These systems require the careful coordination of software packages, services, and configurations across multiple, potentially heterogeneous, resources. For example, the TeraGrid project [19] manages the coordination of software and services by deploying and monitoring a common user environment across distributed, heterogeneous resources. TeraGrid's software and services uniformity simplifies access to its resources, which consist of more than 102 teraflops of compute capability and more than 15 petabytes of online storage, all interconnected by a network that can transfer a terabyte of data in under 10 minutes.

Providing and maintaining a stable infrastructure for these complex Grid systems poses challenges for both administrators who build and maintain Grid resources and scientists who use them. System administration is distributed across multiple administrative domains requiring a significant amount of coordination. This is typically performed by a group of Grid administrators or operators who consider local site policies and resource heterogeneity and decide when software (including updates and patches) should be deployed to resources and how it should be configured. Each site's system administrators, who are not necessarily Grid experts, are then responsible for providing the required Grid software components and services for their resource and debugging any problems that arise. Well-defined requirements and good communication are required in this model; otherwise, inconsistencies arise between the resources that either inconvenience the user or prevent them from using a particular resource altogether. Periodic failures that occur in Grid services due to network or system failures, software misconfiguration, or software bugs also present a challenge. Grid monitoring can be used to detect such problems, leading to a more stable and dependable Grid infrastructure.

One approach to monitoring Grids, used by tools such as MonALISA [10] and GridICE [1], is to aggregate and display data from existing cluster or system administrator monitoring tools such as Ganglia [12], CluMon [3], or Nagios [16]. This provides a centralized, systems-level view of Grid resources where low-level host statistics and queue information can be examined. This type of monitoring information is useful for showing the utilization of Grid resources, but it does not provide the type of high-level monitoring needed to detect user problems, such as incompatible software versions, within the Grid infrastructure.

User-level Grid monitoring provides Grid infrastructure testing and performance measurement from a generic, impartial user's perspective. The goal of user-level monitoring is to detect and fix Grid infrastructure problems before users notice them – user complaints should not be the first indica-

tion of Grid failures. In our view, successful user-level Grid monitoring needs to include these features:

- Runs from a standard user account in order to reflect regular user experiences. It is important not to execute from a system administrator's account, which may have special privileges and use custom shell initialization files.
- Executes with a standard user GSI credential mapped to a standard user account when tests or performance measurements require authentication to Grid services.
- Emulates a regular user by using tests and performance measurements created and configured based on user documentation, rather than on system administrator knowledge (of hostnames, ports, pathnames, etc.). In cases where documentation and tests are developed simultaneously during pre-production, test development should be closely coordinated with the documentation as it is written.
- Centrally manages the configuration of user-level tests or performance measurements in order to ensure consistent testing across resources.
- Easily updates and maintains user-level tests and performance measurements. This is important because tests and measurements are often updated when Grid infrastructure changes. Also, multiple iterations of test development are often required to determine whether a detected test failure stems from a faulty test, incomplete user documentation, or a failed Grid resource.
- Provides a representative indication of Grid status by testing documented user commands and individual Grid software components.
- Automates the periodic execution of user-level tests or performance measurements to understand Grid behavior over time.
- Executes locally on Grid resources to verify user accessible Grid access points. Executes from each resource to every other resource (all-to-all) to detect site-to-site configuration errors such as authentication problems.

In 2003 SDSC, in partnership with TeraGrid, began developing Inca 1 [18] to implement a user-level Grid monitoring system that provided these features. At that time, the only available user-level Grid monitoring tools were the NCSA TestGrid script [15] and Grid Integration Test Script (GITS) [20], both of which ran a fixed number of Grid tests and formatted results in HTML. Although these tools were easy to install and produced useful information, they showed the view of the Grid from a single resource, lacked automation, and were not easily extensible. Inca 1 provided the means to verify that the common user environment was deployed consistently across all TeraGrid resources and to monitor its status.

The initial version of Inca was implemented as a client-server architecture. It provided data collection, data storage (accessible from a Web services interface and with limited archiving capabilities), and data display though Web status pages. Inca 1 was first deployed to TeraGrid in mid 2003 and released in late 2003. Running Inca 1 for a year and a half on TeraGrid taught valuable lessons that have been incorporated into the design of Inca 2, our current release. Inca 2 contains a number of substantial improvements over Inca 1 with respect to security, installation and maintenance, and storage capabilities. Inca 2 has been running on TeraGrid since November 2006, and a production version of the software was released in February 2007. Currently, seven other Grids in the U.S., Europe, and Australia use Inca 2.

In the next section, we describe the design goals and features of Inca 2. We then describe the Inca 2 architecture and how it can be used to provide user-level Grid monitoring. In Section 4, we describe two uses of Inca 2 for Grid software environment verification and benchmarking. Finally, we describe some future work and summarize the paper.

2. Inca 2 Features

Inca 2 is a system that provides user-level monitoring of Grid functionality and performance. It was designed to be general, flexible, scalable, and secure, in addition to being easy to deploy and maintain. Inca benefits Grid operators who oversee the day-to-day operation of a Grid, system administrators who provide and manage resources, and users who run applications on a Grid. Besides implementing the features described in Section 1, Inca 2 has additional features listed below. A '*' after a feature description indicates it is a new feature of Inca 2; other items note improvements to Inca 1 features. The Inca system:

1. Collects a wide variety of user-level monitoring results (e.g., simple test data to more complex performance benchmark output).
2. Captures the context of a test or benchmark as it executes (e.g., executable name, inputs, source host, etc.) so that system administrators have enough information to understand the result and can troubleshoot system problems without having to know the internals of Inca.
3. Eases the process of writing tests or benchmarks and deploying them into Inca installations.
4. Provides means for sharing tests and benchmarks between Inca users. *

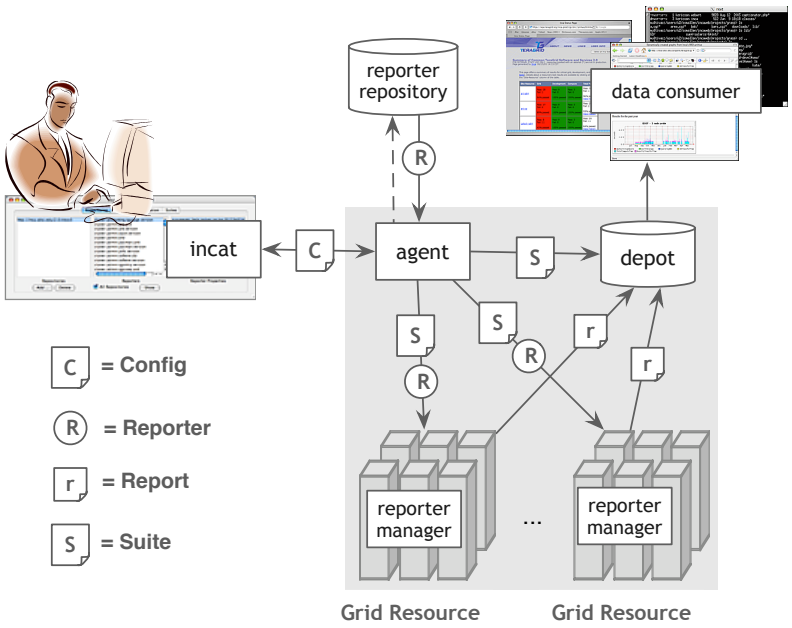


Figure 1. Inca architecture.

5. Easily adapts to new resources and monitoring requirements in order to facilitate maintenance of a running Inca deployment.
6. Stores and archives monitoring results (especially any error messages) in order to understand the behavior of a Grid over time. The results are available through a flexible querying interface.
7. Securely manages short-term proxies for testing of Grid services using MyProxy [13]. *
8. Measures the system impact of tests and benchmarks executing on the monitored resources in order to tune their execution frequency and reduce the impact on resources as needed. *

The following section describes how the Inca architecture implements these features.

3. Inca 2 Architecture

Figure 1 shows the architecture of Inca 2, which incorporates three core components (highlighted box) – the agent, depot, and reporter manager. The *agent* and *reporter managers* coordinate the execution of tests and performance measurements on the Grid resources and the *depot* stores and archives the results. The inputs to Inca 2 are one or more *reporter repositories* that contain user-level tests and benchmarks, called *reporters*, and a configuration file describing how to execute them on the Grid resources. This configuration is normally created using an administration GUI tool called *incat* (Inca Administration Tool). The output or results collected from the resources are queried by the *data consumer* and displayed to users. The following steps describe how an Inca administrator would deploy user-level tests and/or performance measurements to their resources.

1. The Inca administrator either writes reporters to monitor the user-level functionality and performance of their Grid or uses existing reporters in a published repository.
2. The Inca administrator creates a deployment configuration file that describes the user-level monitoring for their Grid using incat and submits it to the agent.
3. The agent fetches reporters from the reporter repository, creates a reporter manager on each resource, and sends the reporters and instructions for executing them to each reporter manager.
4. Each reporter manager executes reporters according to its schedule and sends data to the depot.
5. Data consumers display collected data by querying the depot.

The following subsections describe the Inca components in more detail, using the order of the steps above.

```
#!/usr/bin/env perl
use Inca::Reporter::SimpleUnit;
my $reporter = new Inca::Reporter::SimpleUnit(
 name => 'grid.globus.gramPing',
 version => 2,
 description =>
   'Checks gatekeeper is accessible from local machine',
 url => 'http://www.globus.org',
 unit_name => 'gramPing'
);
$reporter->addDependency('Inca::Reporter::GridProxy');
$reporter->addArg('host', 'gatekeeper host');
$reporter->processArgv(@ARGV);
my $host = $reporter->argValue('host');
my $out = $reporter->loggedCommand
  ("globusrun -a -r $host", 30);
if (!$out) {
 $reporter->unitFailure("globusrun failed: $!");
} elsif($out !~ /GRAM Authentication test successful/) {
 $reporter->unitFailure("globusrun failed: $out");
} else {
 $reporter->unitSuccess();
}
$reporter->print();
```

(a)

```
<?xml version='1.0'?>
<rep:report
  xmlns:rep='http://inca.sdsc.edu/dataModel/report_2.1'>
  <gmt>2007-02-22T18:05:37Z</gmt>
  <hostname>client64-51.sdsc.edu</hostname>
  <name>grid.globus.gramPing</name>
  <version>2</version>
  <workingDir>/Users/ssmallen</workingDir>
  <reporterPath>grid.globus.gramPing-2</reporterPath>
  <args>
    <arg>
      <name>help</name>
      <value>no</value>
    </arg>
    <arg>
      <name>host</name>
      <value>localhost</value>
    </arg>
    <arg>
      <name>log</name>
      <value>3</value>
    </arg>
    <arg>
      <name>verbose</name>
      <value>1</value>
    </arg>
    <arg>
      <name>version</name>
      <value>no</value>
    </arg>
  </args>
  <log>
    <system>
      <gmt>2007-02-22T18:05:36Z</gmt>
      <message>globusrun -a -r localhost</message>
    </system>
  </log>
  <body>
    <unitTest>
      <ID>gramPing</ID>
    </unitTest>
  </body>
  <exitStatus>
    <completed>true</completed>
  </exitStatus>
</rep:report>
```

(b)

Figure 2. (a) An Inca reporter that tests the user-level availability of a Globus GRAM server and (b) sample of its output.

3.1. Reporters

An Inca reporter is an executable program that tests or measures some aspect of a system or installed software. Reporter executables are designed to be easy to produce and can be run outside of the Inca system (e.g., by a system administrator). Existing reporters range from a simple Globus [6] GRAM gatekeeper ping test (see Figure 2) to complex Grid application benchmarks [2]. Reporters must support certain command line options and produce an XML document, or *report*, according to the Inca reporter schema. The goal of the reporter schema is to accommodate multiple types of data and to capture enough information about the reporter execution to diagnose any detected failures. The reporter schema consists of the following elements:

- a set of header tags describing the context of the reporter execution: GMT timestamp, hostname, reporter name, reporter version, working directory, reporter path, and arguments;
- an optional set of debug or information log tags similar to log4j [11] output;
- a body containing the results expressed as an XML sequence; and
- an exit status tag indicating whether the reporter was able to complete its test or measurement, and an optional error message.

Because the body of a report can be any XML sequence, it enables reporters to express a wide variety of information. Today, we have four standard body schemas to express software version information, software functionality or service tests results, usage information, and performance results.

An extensible set of Perl APIs is provided to handle much of the effort of writing reporters. Figure 2 shows (a) an Inca reporter that pings a pre-WS Globus GRAM server and (b) its corresponding output when executed on a local machine. The reporter uses two of the Perl APIs: Inca::Reporter::SimpleUnit (a subclass of Inca::Reporter) and Inca::Reporter::GridProxy. The Inca::Reporter API provides convenience functions for handling the required input arguments (lines 2(a) 12-14 produces lines 2(b) 10-31) and printing the Inca-compliant header and exit status XML tags (lines 2(a) 3-10, 24 produces 2(b) 1-9, 43-46). It also includes some informational and debug logging functions similar to log4j. The log output has been very useful to system administrators, who can view a summary of the test or benchmark that the reporter performed without reading the reporter source code. Line 2(a) 15 shows a special log function, loggedCommand, that executes a system command (using a 30 second timeout) after logging it to the 'system' level of the log XML (producing lines

2(b) 32-37). The subclass Inca::Reporter::SimpleUnit is the API responsible for printing the simplest standard reporter body schema that reports software functionality or service tests results; it handles the printing of the small body XML (lines 2(b) 39-41). Finally, the Inca::Reporter::GridProxy dependency (line 2(a) 11) adds a proxy dependency to the reporter, telling Inca to download a short-term proxy before running this reporter (as discussed in Section 3.8). Most current reporters use the Perl APIs and consist of fewer than 30 lines of code.

3.2. Reporter Repositories

Reporter repositories are collections of reporters, required packages and libraries (including the Perl APIs described in the previous section), and a catalog file. Repository contents are accessible using a URL and designed to be shared across multiple Inca deployments. The catalog file, patterned after APT's Packages.gz [4], allows the user to indicate reporter dependencies on CPAN modules, tar.gz packages, and other external sources. Inca automatically deploys dependencies into a subdirectory of its installation (see Section 3.4); this enables reporters to be deployed to resources with minimal delay. Each package specified in the catalog is versioned, allowing Inca to automatically distribute reporter code updates such as bug fixes. Because of this, Inca administrators who maintain repositories need to ensure that updates to reporters work properly before committing them to a repository.

The Inca team publishes a reporter repository that contains 143 reporters developed for TeraGrid. Most reporters are either version reporters that collect version information from a software package, unit test reporters that run a more involved functionality test, or performance benchmark reporters. The unit and version reporters cover software such as Grid middleware and tools, compilers, math libraries, data tools, and visualization tools. Current performance reporters measure data transfer and execute Grid benchmarks as described in Section 4.2.

3.3. Inca administration tool (incat)

The Inca administrative tool (incat) is a Java Swing GUI that Inca administrators use to configure user-level Grid monitoring on their resources. Incat allows an Inca administrator to choose which resources to monitor and which reporters to deploy to those resources. The configuration is stored in a XML file and sent to the agent, which handles its implementation (described in the next section). The Inca administrator can reload the configuration to make updates. Incat provides a number of conveniences that enable a large number of results to be managed and collected with a minimum effort.

To begin reporter configuration, the Inca administrator enters the location of one more reporter repositories, and incat loads and displays the reporters available from them (Figure 3). For each reporter, the Inca administrator can specify the resources to run on, input arguments, the runtime environment, the frequency of execution, resource limits, and email notifications. The sequence of reports that a reporter produces on a resource when executed with a specific set of input arguments and runtime environment is known as a *report series* (or *series* for short). Thus, the grid.globus.gramPing reporter in Figure 2 would produce one report series when executed with 'blue.ufo.edu' as the host argument and another report series with 'red.ufo.edu'. A set of related series can be grouped into *suites* and shared across other Inca deployments, e.g., a Globus suite or a data transfer performance suite. Suites can also be useful in determining interoperability among Grids or whether an application's requirements are being fulfilled on a Grid.

For each monitored resource, an Inca administrator selects an access method (SSH or Globus) to access the resource and start monitoring there (described in Sections 3.4 and 3.5). Resources can be aggregated into groups for reference in the series configuration, facilitating the process of executing the same series on multiple resources. Also, input and runtime environment attributes can be attached to resource groups for reference in series configuration. These resource *macros* can have multiple values, in which case the series will be executed on the resource once for each macro value. Resource groups and macros ease the process of defining and maintaining multiple similar series.

3.4. Agent

The Inca agent is a server that implements the configuration specified by the Inca administrator. When the agent receives a configuration from incat, it expands any resource group and macro references to determine the series to execute on each resource and stores this information in the depot. The agent stages and launches an Inca component, called a reporter manager (described in the following subsection), on each resource, using either SSH or Globus. After a reporter manager contacts its agent, the agent transmits the reporters to execute, any dependencies, and a suite that contains series configuration and an execution schedule. As an Inca administrator makes updates to the configuration, the agent will push out new reporters and/or suite changes to each of the affected resources.

The agent regularly pings its reporter managers in order to detect when one has failed (for example, when a system is taken down for maintenance) and so must be restarted. The agent also maintains a local cache of reporters and their dependencies; it periodically checks repositories and automatically distributes reporter updates to the reporter managers

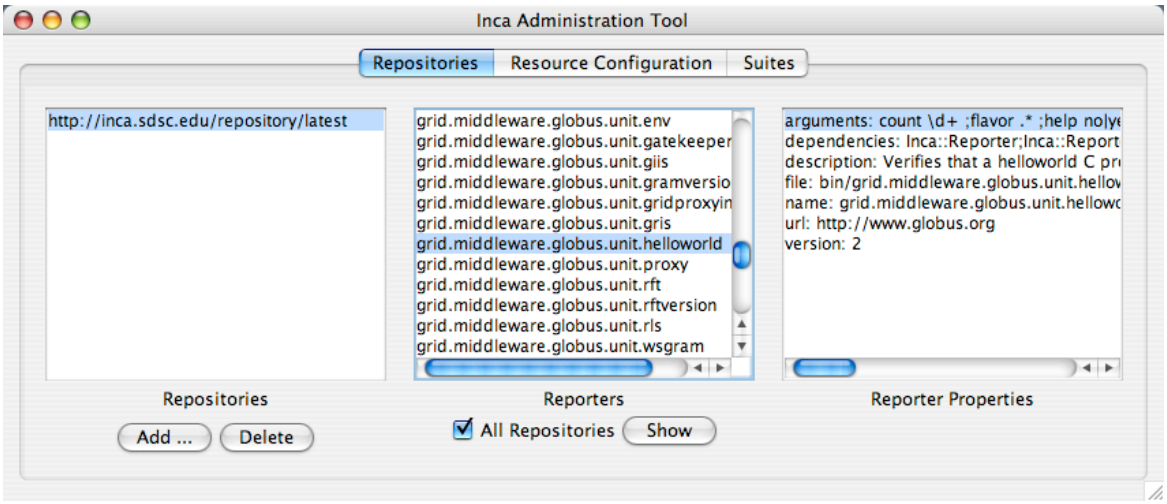


Figure 3. Screenshot of incat that shows the reporters and their attributes available from the reporter repository.

as needed. The agent is implemented using Java.

3.5. Reporter Manager

The Inca reporter manager is a lightweight process responsible for managing the schedule and execution of Inca reporters on a single resource. The reporter manager receives reporter updates and dependencies from the agent along with requests for reporter scheduling changes. Running under a regular user account, the reporter manager executes reporters on-demand or using an internal cron scheduler and sends reports to the depot for archiving. The reporter manager monitors reporter system usage and enforces limits (e.g., wall clock time, CPU time, memory). System usage information is sent to the depot with each report. The reporter manager is implemented using Perl to allow execution on a wide variety of UNIX systems.

3.6. Depot

The Inca depot server is responsible for storing configuration information and the data produced by reporters. The depot uses a relational database via Hibernate [7], enabling the use of a variety of database backends. The depot provides full archiving of reporter output, and the database schema is tuned to minimize redundant data. Inca administrators can specify pattern tests to run on reports as they are received, and the depot will send email when the results of these tests change. This provides the administrator with quick notification of problems as they appear and of the results of repairs.

Data can be retrieved using SQL queries. Predefined queries exist to return the latest report instances of a suite, a single report instance, or a report history. A Web services interface is also available to provide unauthenticated query access to data. The depot is implemented using Java.

3.7. Data Consumer

The Inca data consumer is a Web application that queries the Inca depot for data and displays it in a user-friendly format. The data consumer is packaged with Jetty [8] so that Web pages can be served out of the box. A set of extensible JSP tags and pages query the Inca depot for data (returned as XML) and a set of XSL stylesheets format the data as HTML. An Inca administrator can customize the data display, either by modifying the XSL stylesheets or by writing their own JSP tags and pages.

3.8. Security

To provide for security, all Inca components communicate with each other using SSL, and sensitive information is encrypted before being written to disk. To handle reporters that require a proxy credential to execute, the reporter manager fetches a short-term proxy from a MyProxy [13] server (as specified by an Inca administrator), then destroys the proxy once the reporter has completed. MyProxy authentication information is fetched from the agent each time a proxy credential is needed and then cleared from memory. Generating short-term proxies as needed is more secure than maintaining long-term credentials and is easier to manage.

3.9. Coordinating Inca Deployments

A planned extension to Inca 2 supports the distribution of control and storage workload among multiple Inca components. Figure 4 shows a scenario where two Inca deployments coordinate with each other. This could occur in application communities like GEON [17], which maintains its own Grid but also has access to TeraGrid resources. Running two Inca deployments independently could result in duplicate testing. Instead, we plan to eliminate redundancy by allowing multiple deployments to coordinate suite execution. Figure 4 shows this type of coordinated deployment, where Grid B has its own resources but also has access to resources in Grid A. When an Inca administrator submits a suite to Grid B, the Inca agent running on Grid B will determine which reporters run on Grid A resources and forward them as a suite request to Grid A's agent. If Grid B has permission to submit suites, Grid A's agent will execute the suite and configure its depot to forwards results back to Grid B's depot. This extension is also expected to provide the mechanisms for scalability and fault tolerance in the Inca system.

4. Use Cases

4.1. TeraGrid

Requirements for user-level Grid monitoring on TeraGrid initially included the validation and verification of Coordinated TeraGrid Software and Services (CTSS) on each TeraGrid resource. As described in the introduction, CTSS was designed to allow users to run Grid jobs more easily across its distributed, heterogeneous resources. CTSS version 3 (CTSSv3) is available on all TeraGrid compute resources and contains approximately 30 software packages that provide Grid tools and services (e.g., Globus, GSI-SSH, MyProxy), data management tools (e.g., SRB, HDF5), and applications tools (e.g., MPICH, GCC).

To provide user-level monitoring of CTSS, Inca 1 was deployed on TeraGrid in 2003; TeraGrid's transition to Inca 2 completed in November 2006. In addition to CTSSv3 monitoring, Inca is now also being used to collect job usage information from Globus 2 GRAM logs and to detect expired host and CA certificates and CA CRLs.

A number of version and unit test reporters were developed to test the software packages in CTSS. Currently, there are 56 such reporters executing an average of 109 series on each of TeraGrid's 18 resources. They include 48 all-to-all tests (GSI-SSH, GRAM, and GridFTP), 20 Grid-related tests, 28 data-related tests, 30 application tool-related tests, and 2 security-related tests.

The TeraGrid Inca configuration makes extensive use of macros and resource groups and currently uses 80 series and 82 resource macros to manage the total of 1,928 series executing on TeraGrid resources. Figure 5(a) illustrates TeraGrid's Inca 2 deployment configuration. The agent, depot and consumer components are hosted at SDSC on sapa.sdsc.edu. A reporter repository for TeraGrid is hosted at SDSC on inca.sdsc.edu.

TeraGrid currently has six status pages to display the Inca monitoring information: a summary of CTSSv3 test failures, a detailed grid of CTSSv3 test results, a page showing grid job usage, results for security tests, results for secure MDS tests, and a list of all running reporters.

Figure 5(b) shows a portion of the TeraGrid detailed grid of CTSSv3 test results page. CTSSv3 software packages are listed along the column and TeraGrid resources (anonymized) in the header rows. Test results for each package and resource are shown in the table body rows.

4.2. GrASP

The Grid Assessment Probes (GrASP) [2] are designed to serve as simple grid application kernel exemplars as well as a set of diagnostic tools. They test and measure the performance of basic Grid functions including file transfers, remote execution, and information services queries. There are two basic probes: circle and gather. The circle probe transfers a 100 MB file around a ring of Grid nodes and measures the transfer time at each step. The gather probe transfers a 100 MB file from some number of source nodes to a compute node, runs a computation through the job queue on the compute resource, and then transfers a 100 MB file to a single destination node. The transfer of the first set of files is executed in parallel. Transfer, compute, and queue times are recorded.

The GrASP probes yield data about Grid infrastructure performance and reliability. In order to be most useful, probe data must be gathered repeatedly over time and archived. In addition to storing the performance measurements, any error messages encountered must be stored as well. In order to gather historical data, the GrASP probes were modified to follow the Inca reporter specifications and deployed using Inca. Porting the original code to follow the reporter specifications proved to be trivial. A number of scripts were then written to graph and analyze the GrASP data and related error messages stored in the Inca depot.

Using Inca, GrASP was deployed on TeraGrid and GEON for a period of over 6 months [9]. During this deployment, the execution time of Preco, a Grid application run on the TeraGrid, was predicted within 12% (excluding setup and cleanup time). A recurring Globus 2 bug was also identified as shown in Figure 6. The figure shows the frequency and types (shortened) of system errors that were detected while executing GrASP on TeraGrid in 2005. The highest occurring error was number 9, Assertion GLOBUS

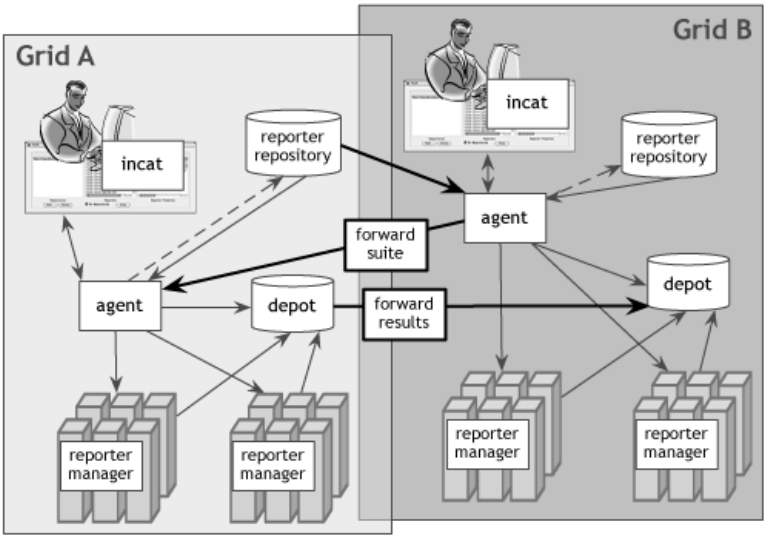
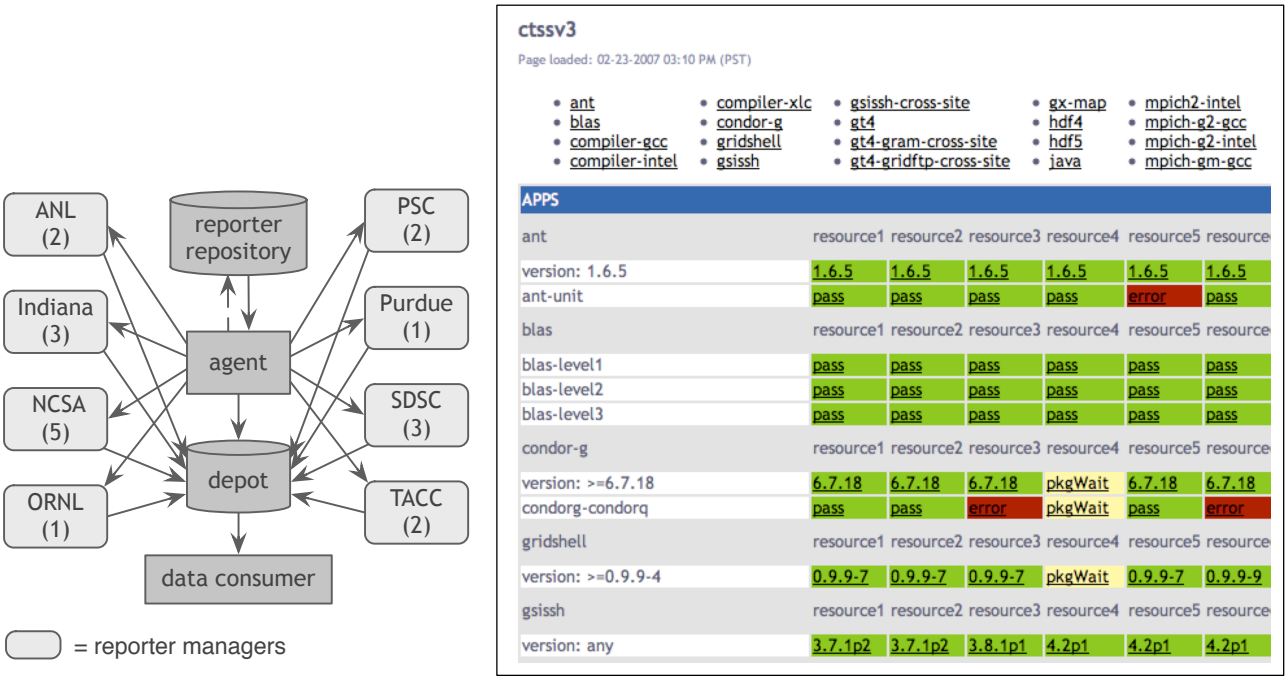


Figure 4. Coordinated Inca deployments.



(a) (b)

Figure 5. (a) TeraGrid's Inca 2 deployment and (b) portion of the TeraGrid CTSSv3 detailed status page with the resources anonymized.

FALSE, which was the result of a bug in the Globus 2 GRAM server.

4.3. Other Uses

Other Inca users include the European DEISA Grid [5] and the UK National Grid Service (NGS) [14]. DEISA and NGS began using Inca in late 2004 and early 2005, respec-

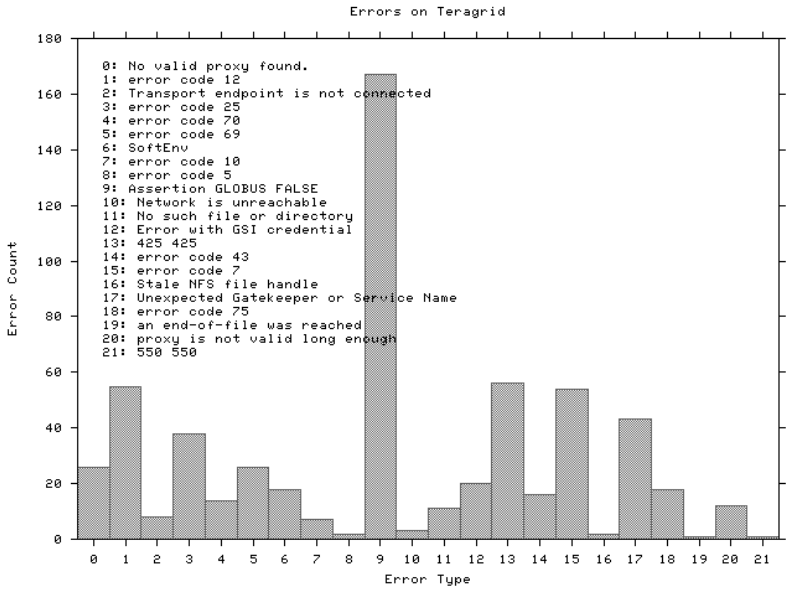


Figure 6. Number and types of errors collected by GrASP benchmark execution over a six-month period on TeraGrid.

tively. Similar to TeraGrid, DEISA uses Inca to verify their Common Production Environment (DCPE) across 11 heterogeneous resources. They currently have 40 version reporters to verify each of the packages included in DCPE. In addition, they have unit reporters for the DEISA Applications Test Suite (DATS) to check basic operation of the C/C++/Fortran compilers, the MPI library, some numerical libraries, OpenMP jobs, CORBA middleware, and a third party application. DEISA is currently using Inca 1 and will transition to Inca 2 in the next few months.

NGS currently has Inca 2 deployed on four resources to verify their Grid services (on eight sites) and compilers. They have wrapped the GITS [20] tests using a new API developed by the GITS developer. The GITS reporters include 17 tests, such as small job submissions and data transfers, that verify Globus and GSI-SSH functionality. NGS also has 12 tests for their compilers and utilize 2 security reporters from the Inca repository.

5. Future Work

In order to improve problem detection and diagnosis for system administrators, we plan to add graph rendering and statistical analysis capabilities to the Inca data consumer. This will improve troubleshooting capabilities by allowing errors to be understood in a historical context. It will also enable summary displays analogous to those generated by Web page statistics packages like Webalizer [21] that provide a number of useful statistics. We expect to follow recommendations from TeraGrid and other Inca users on the types of statistics and graphs that will be most useful. Our plan is to offer a default set of statistics and graphs that can be used directly and to provide the capability to add others.

Also to aid system administrator troubleshooting, we plan to enhance Inca with a knowledge base that can be used to discuss problems and suggest solutions. The knowledge base will be populated by solutions provided by system administrators and self-moderated to improve quality. This knowledge base could provide a vital resource for system administrators and offer a forum for sharing valuable experience and skills.

Finally, we plan to improve the fault tolerance of Inca by enhancing the reporter manager so that it caches reports locally if it is unable to contact the depot or sends reports to a backup depot if one is available. The agent will similarly be enhanced to cache configuration changes locally or send configuration to a backup depot if one is available in case of a depot failure.

6. Summary

The goal of user-level Grid monitoring is to test and measure Grid infrastructure from an impartial user perspective and detect problems before users notice them. Inca 2 implements such a user-level Grid monitoring system and provides a number of new and improved features over our first version, Inca 1. The architecture and features of Inca 2 enable an Inca administrator to easily write and deploy tests

and performance measurements to Grid resources and maintain them in a changing Grid environment. The storage, archiving, and querying capabilities also enable in-depth analysis of Grid errors and performance in a historical context, which benefit Grid operators and system administrators. Inca 2 also provides a number of security features such as short-term proxy management. A production version of Inca 2 was released in February 2007 and has been used in production environments to verify the common user environment of TeraGrid and to execute and collect data for a set of Grid benchmarks on both TeraGrid and GEON. Future enhancements of Inca 2 will improve its fault tolerance and enable more in-depth analysis of the overall Grid health and behavior. They will also make it easier to detect and resolve errors and understand them in a historical context. The features that Inca provides and its proposed enhancements improve the stability of the Grid infrastructure it monitors, ultimately improving the productivity of scientists who conduct research on a Grid.

7. Acknowledgements

The authors would like to thank Denis Girou and David Spence for providing details for the DEISA and NGS Inca deployments, respectively. This work was supported by NSF grants SCI-0503944, SCI-0438741, and SCI-0503697.

References

[1] S. Andreozzi, N. D. Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, and M. C. Vistoli. GridICE: a monitoring service for Grid systems. *Future Generation Computer Systems*, 21(4):559–571, 2005.

[2] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark Probes for Grid Assessment. April 2004.

[3] Clumon Web page. `http://clumon.ncsa.uiuc.edu`, 2007.

[4] Debian – The Universal Operating System. `http://www.debian.org`, 2007.

[5] Deisa - Distributed European Infrastructure for Supercomputing Appications. `http://www.deisa.org/`, 2007.

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

[7] Hibernate Web page. `http://www.hibernate.org`, 2007.

[8] Jetty Web page. `http://jetty.mortbay.org`, 2007.

[9] O. Khalili, J. He, C. Olschanowsky, A. Snavely, and H. Casanova. Measuring the Performance and Reliability of Production Computational Grids. September 2006.

[10] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, and C. Dobre. MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications. In *CHEP 2004*, Interlaken, Switzerland, September 2004.

[11] Log4J Project Web page. `http://logging.apache.org/log4j`, 2007.

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

[13] MyProxy Credential Management Service. `http://myproxy.ncsa.uiuc.edu`, 2007.

[14] NGS Web page. `http://www.grid-support.ac.uk`, 2007.

[15] NCSA TestGrid Project. `http://grid.ncsa.uiuc.edu/test`, 2007.

[16] Nagios Web page. `http://www.nagios.org`, 2007.

[17] 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. *Submitted to GSA Today*.

[18] S. Smallen, C. Olschanowsky, K. Ericson, P. Beckman, and J. M. Schopf. The Inca Test Harness and Reporting Framework. In *Proceedings of Supercomputing 04*, November 2004.

[19] The TeraGrid Project Web page. `http://www.teragrid.org`, 2007.

[20] The UK Grid Integration Test Script - GITS. `http://www.soton.ac.uk/~djb1/gits.html`, 2007.

[21] Webalizer Web page. `http://www.mrunix.net/webalizer/`, 2007.