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

User-level Grid monitoring is valuable, how it relates to other Grid monitoring, and what are its implementation challenges. Inca 1 good but had limitations. In this paper, we introduce Inca 2 and describe its features and architecture. We then show a few use cases for Inca is being deployed on TeraGrid and XXX? System impact results and performance results. Finally, we discuss our future work.

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, heterogeneous resources. The TeraGrid project [16], for example, manages the coordination of software and services by deploying and monitoring a common user environment across distributed, heterogeneous resources. The organization of TeraGrid software and services is referred to as the Common TeraGrid Software and Services (CTSS). 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. First, 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 between the resources arise that either inconvenience the user or prevent them from using particular resources together. A second challenge is that failures will occur over time in the Grid services due to network or system failures, software misconfiguration, or software bugs. In order to address these challenges, 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 [9] and GridICE [1], is to aggregate and display data from existing cluster or system administrator monitoring tools such as Ganglia [11], CluMon [4], or Nagios [14]. 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 within the Grid infrastructure, such as incompatible software versions.

User-level Grid monitoring approaches testing and performance measurement of the Grid infrastructure from the user perspective. The goal is to act as an impartial test user of the Grid and detect problems with the infrastructure so they can be fixed before users notice them – user complaints should not be the first indication of Grid failures. In our view, the following guidelines define user-level Grid monitoring:

- In order to reflect regular user experiences, tests or performance measurements of the Grid infrastructure should be done from a standard user account. In fact, it is important to not execute under a system administrator's account because they may be privileged and often have custom shell initialization files.

- Also in order to reflect regular user experiences, tests should be written and configured using information directly from user documentation (e.g., hostnames, ports, pathnames, etc.). This may not always be possible when documentation and testing are done in parallel in a pre-production environment but they should be closely coordinated activities.
- Since Grids are dynamic, user-level tests or performance measurements should be automated and executed periodically.
- Tests or performance measurements that interact with Grid services require authentication and should be executed using a standard user GSI credential that is mapped to the standard user account.
- Tests or performance measurements should be executed locally on all Grid resources when appropriate so that all Grid access points available to users are verified. Similarly, it is important to execute some tests all-to-all in order to detect site-to-site configuration errors such as authentication problems.
- The configuration of tests or performance measurements should be managed centrally in order to ensure consistent testing of the resources, rather than by system administrators who may not use the same information that is available to users.

In our experience, writing and maintaining user-level tests and performance measurements is an iterative refinement process since Grids are dynamic and the software environment changes over time as packages are upgraded. It is also difficult to write and deploy a test perfectly the first time because of portability issues and sometimes incomplete user documentation. For example, it may take a few iterations to determine whether detected failures are the result of a faulty test, fault user documentation, or a faulty system. As such, this can be a labor-intensive activity and therefore difficult to achieve full test coverage. In practice, either testing all the commands that exist in the user documentation and/or having at least one basic test for each software component of the Grid infrastructure achieves a reasonable subset of testing.

In 2003, SDSC, in partnership with TeraGrid, began developing Inca 1 [15] to implement a user-level Grid monitoring system with the above features and started using it to validate and verify that CTSS was deployed consistently across all TeraGrid resources and to monitor its status. At that time, the only available user-level Grid monitoring tools were the NCSA TestGrid script [13] and GITS [6], scripts that 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. The initial version of Inca was implemented as a client-server architecture. It provided data collection, data storage (with limited archiving capabilities) that was accessible from a Web services interface, and data display though Web status pages. Inca 1 was first deployed to TeraGrid in mid 2003 and released in late 2003. After running Inca 1 for a year and a half on TeraGrid, we learned some valuable lessons and designed 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. A production release of Inca 2 was provided in February 2007 and has been running on TeraGrid since November 2006. Currently, seven other Grids in the U.S., Europe, and Australia use Inca.

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

The main goal of the Inca 2 design was to XXX (general, flexible, easy to install, deploy, and maintain, scalable, secure). Inca benefits ... The following lists the key features of Inca 2 noting which are new features to Inca 1 versus improved.

1. Collects a wide variety of user-level monitoring results (e.g., simple test data to more complex performance benchmark output). (improved)
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. (improved)
3. Eases the process of writing tests or benchmarks and deploying them into Inca installations. (improved)
4. Provides means for sharing tests or benchmarks with other Inca users in order to promote sharing of each other's expertise (new)
5. Easily adapts to new resources and monitoring requirements in order to facilitate maintenance of a running Inca deployment. (improved)
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. (improved)

7. Securely manages short-term proxies for testing of Grid services using MyProxy [12].

8. Measures the system impact of tests and benchmarks executing on the monitored resources in order to tune their execution frequency to reduce the impact on the resources.

3. Inca 2 Architecture

Figure 1 shows the architecture of Inca 2 designed to implement the features described in the previous section. Inca 2 is composed of 3 core components (in 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 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 test and/or performance measurements to their resources.

1. Using the guidelines described in the introduction, the Inca administrator either writes reporters to monitor 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

   (a) fetches reporters from the reporter repository

   (b) creates a reporter manager on each resource

   (c) sends the reporters and instructions for executing them to each reporter manager.

4. Each reporter manager executes reporters according to its schedule and sends results (reports) to the depot

5. Data consumers display collected data (reports) 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;
use Inca::Reporter::GridProxy;
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->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) an output sample.

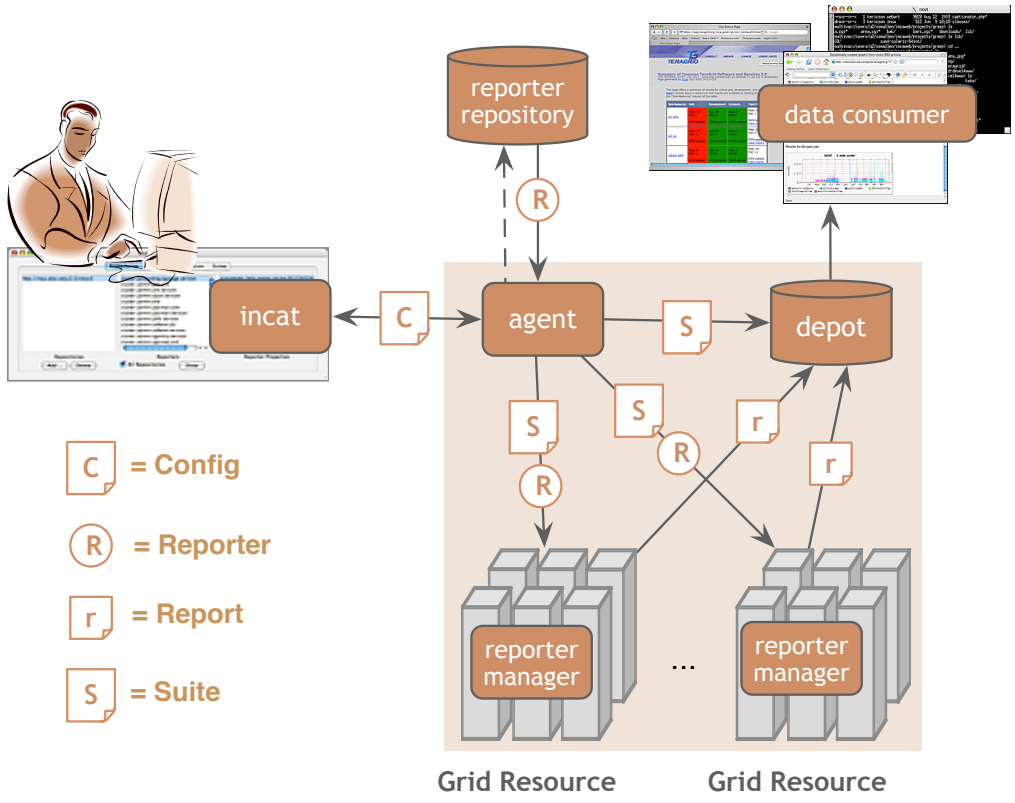


Figure 1. Inca architecture.

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). A reporter should be written using the user-level monitoring guidelines described in the introduction and could be a simple Globus [5] GRAM gatekeeper ping test (see Figure 2) or a more complex Grid application benchmark [3]. 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 [10] output;
- a body containing the results expressed as any 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 the 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 (which is 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 → lines

2(b) 10-31) and printing the Inca-compliant header and exit status XML tags (lines2(a) 4-11, 24 → 2(b) 1-9, 43-46). It also includes some informational and debug logging functions again 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) and logs to the 'system' level of the log XML output (lines 2(b) 32-36). The subclass Inca::Reporter::SimpleUnit is the API responsible for printing the simplest standard reporter body schema – reporting software functionality or service tests results; it handles the printing of the small body XML (lines 2(b) 39-41) using the functions unitSuccesss (line 2(a) 22) or unitFailure (lines 2(a) 18, 20) depending on the output of system command. Finally, the Inca::Reporter::GridProxy API (line 2(a) 3) adds a proxy dependency to the reporter telling the Inca system to download a short-term proxy before running this reporter (this is discussed further 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 that consist 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 follows APT's Packages.gz [2] format and thus tar.gz dependencies such as non-standard Perl APIs or source code can be added to reporter catalog entries. Inca will then automatically deploy tar.gz dependencies into a subdirectory of its installation (more in Section 3.4). This enables reporters to be deployed to resources with a minimum of effort. Furthermore, each package specified in the catalog is versioned. This allows Inca to automatically distribute reporter code updates such as bug fixes. As such, Inca administrators maintaining repositories need to be careful to ensure their updates are tested and work.

The Inca team publishes a reporter repository that contains 143 reporters developed and tested for TeraGrid. Most reporters are either version reporters that collect version information from a software package and serve as a most basic unit test, unit test reporters that run a more involved unit test, or performance benchmark reporters. The unit and version reporters cover software such as Grid middleware, compilers, math libraries, data tools, and visualization tools. The performance reporters available currently 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 GUI implemented in Java Swing that an Inca administrator uses 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 the implementation (as described in the next section). The Inca administrator can reload the configuration at any point and 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 will load and display the reporters available from them. 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 reports that a reporter produces on a resource when executed with specific set of input arguments and runtime environment are 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 host 'tg-login.sdsc.edu' and another report series when executed with 'dslogin.sdsc.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 to determine interoperability among Grids or to determine 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 also be grouped together and referenced in the series configuration. This facilitates the process of executing the same series on multiple resources. Furthermore, attributes called *macros* can be attached to resources and resource groups and similarly be referenced in the series configuration (i.e., inputs and runtime environment). For example, it is useful to have a macro called 'jobmanager' since the batch system is likely to differ from one machine to the next and multiple tests will require its value. A resource macro can also have multiple values in which case the series will be executed on the resource once for each macro value. The benefit of resource groups and macros is that when a reporter is executed on a resource multiple times with different input arguments, it can be represented succinctly. Similarly, when a resource attribute is changed, it only has to be changed in one place and then every series that references it will be updated as well. This makes the initial configuration and maintenance easier on the Inca administrator and less error prone.

3.4. Agent

The Inca agent is a server that implements the configuration specified by the Inca administrator. When it receives a configuration file from the Inca administrator, it will determine which reporters should be executed on each resource by expanding the resource macros and groups; it then stores the expanded configuration 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. Once a reporter manager contacts its agent, the agent transmits the reporters to execute along with their dependencies and a suite that contains their series configuration and schedule of execution. 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.

To further manage the Inca deployment, the agent regularly pings the reporter managers on each resource in order to detect when a reporter manager has failed so it can be restarted. For example, if a resource goes down for a planned maintenance, the agent will attempt to restart the reporter manager (by default every hour) until it succeeds. The agent also maintains a local cache of reporters and their dependencies; it periodically checks the repositories for updates every 4 hours by default and automatically distributes them to the reporter managers (if the latest version is desired). The agent is implemented in 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 using the system command `ps` 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 in Perl since Perl is available by default on UNIX machines. The cron scheduling is implemented using the CPAN Perl module Schedule::Cron.

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] so that it can use a variety of databases. The depot provides full archiving of reporter output and structures its schema around series and the minimization of redundant data, i.e., only stores changes from one report to another in a report series. The depot also supports comparisons that are executed on a newly received report; currently regular expression matching and version comparisons are supported. Optionally, an Inca administrator can select to receive email notification when the result of the comparison changes.

Data can be queried 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 in 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 by either modifying the XSL stylesheets or writing their own JSP tags and pages.

3.8. Security

To reduce the security risk of Inca, all Inca components communicate with each other using SSL by default. Also, sensitive information such as passwords is encrypted before being stored to disk on the agent. To handle reporters that require a valid proxy credential to execute, the reporter manager will fetch a short-term proxy from a MyProxy [12] server (as specified by an Inca administrator) and then destroys the proxy once the reporter has completed. The goal is to only have short-term proxies on the machine for the times at which they are needed; this is more secure than maintaining a long-term credential on the resource and easier to manage. Furthermore, the MyProxy authentication information is fetched from the agent each time a proxy credential is needed and then cleared from memory.

3.9. Scalability

4. Use Cases

4.1. TeraGrid

Requirements for user-level Grid monitoring on TeraGrid initially included the validation and verification of CTSS on each resource. As described in the introduction, CTSS was designed to allow users to more easily run Grid jobs across its distributed, heterogeneous resources. CTSS version 3 (CTSSv3) is available from 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). Inca was originally deployed on TeraGrid in 2003 using Inca 1 and was transitioned to Inca 2 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.

To provide user-level monitoring of CTSSv3, 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 1928 series executing on TeraGrid resources. Figure 3 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 also hosted at SDSC on inca.sdsc.edu.

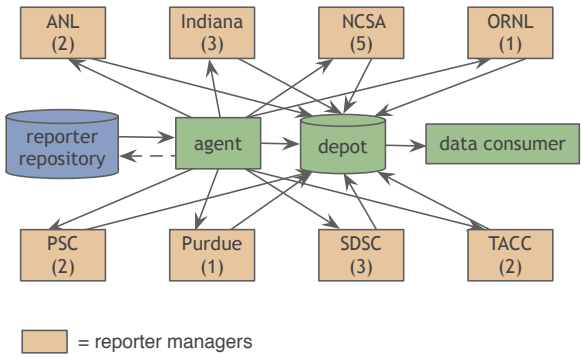
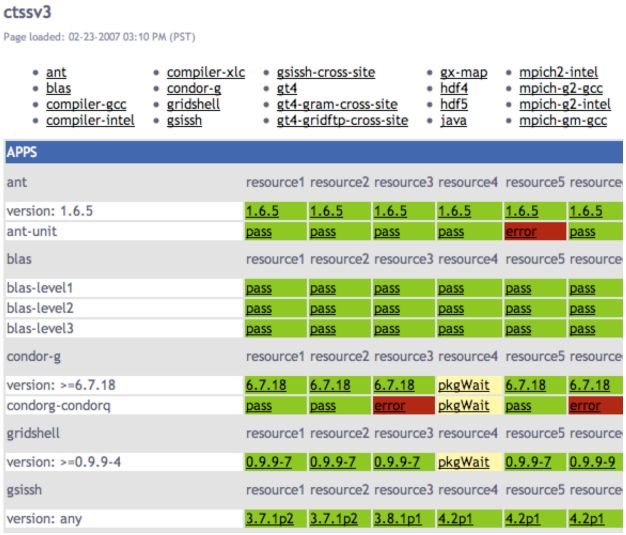


Figure 3. TeraGrid's Inca 2 Deployment.

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 4 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.


ctssv3

Page loaded: 02-23-2007 03:10 PM (PST)

- ant
- blas
- compiler-gcc
- compiler-intel
- compiler-xlc
- condor-g
- gridshell
- gsissh
- gsissh-cross-site
- gt4
- gt4-gram-cross-site
- gt4-gridftp-cross-site
- gx-map
- hdf4
- hdf5
- java
- mpich2-intel
- mpich-g2-gcc
- mpich-g2-intel
- mpich-gm-gcc

APPS						
ant	resource1	resource2	resource3	resource4	resource5	resource
version: 1.6.5	1.6.5	1.6.5	1.6.5	1.6.5	1.6.5	1.6.5
ant-unit	pass	pass	pass	pass	error	pass
blas	resource1	resource2	resource3	resource4	resource5	resource
blas-level1	pass	pass	pass	pass	pass	pass
blas-level2	pass	pass	pass	pass	pass	pass
blas-level3	pass	pass	pass	pass	pass	pass
condor-g	resource1	resource2	resource3	resource4	resource5	resource
version: >=6.7.18	6.7.18	6.7.18	6.7.18	pkgWait	6.7.18	6.7.18
condorg-condorq	pass	pass	error	pkgWait	pass	error
gridshell	resource1	resource2	resource3	resource4	resource5	resource
version: >=0.9.9-4	0.9.9-7	0.9.9-7	0.9.9-7	pkgWait	0.9.9-7	0.9.9-9
gsissh	resource1	resource2	resource3	resource4	resource5	resource
version: any	3.7.1p2	3.7.1p2	3.8.1p1	4.2p1	4.2p1	4.2p1

Figure 4. Portion of the TeraGrid CTSSv3 detailed status page with the resources anonymized.

4.2. GrASP

4.2.1 Description and Requirements

4.2.2 Usage of Inca

4.2.3 Results

6. Summary

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] Debian Web page. `http://www.debian.org`, 2007.

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

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

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

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

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

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

[9] 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.

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

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

[12] MyProxy Web page. `http://myproxy.ncsa.uiuc.edu`, 2007.

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

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

[15] 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.

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