Daniel Templeton, Sun Microsystems, Inc.
Andreas Haas, Sun Microsystems GmbH
May 25, 2006
Updated: February 27, 2007

N1™ Grid Engine DRMAA 1.0 Implementation – Experience Report

Status of This Document

This document provides information to the Grid community about the adoption of the GGF specification GFD-R-P.022 in the N1 Grid Engine distributed resource management (DRM) system. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This document describes experiences in the implementation of the *Distributed Resource Management Application API* (DRMAA) specification for the N1 Grid Engine workload management system. The document reports about issues that where identified during implementation and test of the DRMAA C library for N1 Grid Engine, which was evaluated successfully with the DRMAA working group compliance test for C bindings.

Contents

1. Introduction

The Distributed Resource Management Application API specification (GFD-R.P.022) [1] specifies a generalized API to distributed resource management systems in order to facilitate integration of application programs. Soon after DRMAA reached "proposed recommendation" status, various DRM vendors and Grid community-oriented projects started implementing DRMAA language bindings. Today, there are several DRM system implementations (Sun Grid Engine, Condor, GridWay, Torque) in several different languages (C, Java, Perl, Python).

This document describes the N1 Grid Engine DRMAA C binding implementation and the experience gained from producing that implementation.

2. N1 Grid Engine

N1 Grid Engine is a workload manager intended to handle thousands of compute node and millions of simultaneous “jobs.” N1 Grid Engine allows users to submit jobs, single, parametric, or parallel, to the system using a variety of methods. Administrators can configure the grid using “queues,” “host groups,” “departments,” and a variety of other abstractions to maximize workload throughput. The N1 Grid Engine DRM system places submitted jobs into a waiting pool until they can be scheduled. Once the scheduler has decided the optimal placement for the job, based on the current state of the grid, the job's needs, and the other jobs waiting to be scheduled, the job is dispatched to an execution queue. When the job finishes execution, the results and usage statistics are collected and made available to the submitting user. Simultaneously, N1 Grid Engine monitors the state of all nodes and resources in the grid. Should a node become overloaded or a resource become scarce, the information is provided to the scheduler in real-time, so that scheduling decisions are always based on the current and complete state of the grid.

Beyond the basic functionality of a workload manager, N1 Grid Engine also provides a rich concept of policy. Administrators can configure N1 Grid Engine to manage workflow based on pre-defined policies. A policy may be as simple as always giving highest priory on a particular host to a particular type of job, or as complex as allowing users from a specific department working on a specific project to submit jobs to use a given resource on Tuesdays and Thursdays and only if the number of jobs currently submitted is less than some number.

Policy in N1 Grid Engine is supported by two important concepts: complexes and load sensors. A complex is a representation of a resource. It may represent something concrete and measurable, like CPU usage, or something more abstract, like licenses or execution rights. Some complexes are defined by default, but administrators have the ability to create new complexes at any time. A complex can be attached globally, or to a host, a group of hosts, a queue instance, or any combination thereof. Load sensors are the eyes and ears of N1 Grid Engine. A load sensor is a small program, often a shell script, which measures the current value of one or more complexes. The execution nodes measure the default complex values automatically, but administrators can add new load sensors to measure the values of custom defined complexes. Given that complexes can be arbitrarily defined, and that administrators can create new load sensors, N1 Grid Engine provides an almost infinite potential for specialization.

Adding to the flexibility of N1 Grid Engine is the Grid Engine open source project (see the Grid Engine open source project at http://gridengine.sunsource.net/). The Grid Engine open source project makes publicly available the core components of N1 Grid Engine, for research, extension, and reuse. The community around the Grid Engine open source project is lively and active. Community members assist each other with support questions and make source code contributions, which eventually become part of N1 Grid Engine. The source code available from the Grid Engine open source project is released under the Sun Industry Standards Source License (SISSL) Version 1.2 [2].

3. Implementation of the N1 Grid Engine DRMAA Library

A DRMAA C binding implementation was first released with the N1 Grid Engine 6.0 DRM system. The binding implementation was included as a shared library and was based on the DRMAA C binding specification[3], version 0.95. This version of the specification differed from 1.0 only in very minor details. Using the DRMAA binding implementation, several ISVs and Grid Engine open source community members began work on DRMAA-enabling their applications (see DRMAA Wiki, Information about DRMAA users at http://www.drmaa.org/wiki/index.php/DrmaaUsers).

With the N1 Grid Engine 6.0 update 2 DRM system, a Java™ language binding implementation was also included. This implementation was built on top of the C binding implementation, and was based on the DRMAA Java language binding specification, version 0.5. The Java language binding implementation was also warmly received by the ISV and open source community, spawning several notable projects, some of which have been released (see DRMAA Wiki, Information about DRMAA users at http://www.drmaa.org/wiki/index.php/DrmaaUsers).

A DRMAA C binding implementation supporting the DRMAA C binding specification, version 1.0 is included in the N1 Grid Engine 6.0 update 9 DRM system. While the differences from the previous binding implementation were very minor, this release was significant in that it contained a binding implementation compliant with the final version of the GFD-R.P.022 language independent DRMAA specification.

N1 Grid Engine's DRMAA C binding implementation was developed in parallel with the DRMAA specification itself. In some cases the binding implementation served as a test ground for concepts in the specification. Because of this close relationship, the development of the binding implementation was an “insider” process and was largely unaffected by semantics of the DRMAA specification. The intent of the specification was understood, and the intent is what was implemented.

The DRMAA specification provides a concise and targeted application programmer interface (API) for the management of workloads. It lays out in detail the elements which are important for cross-platform compatibility while allowing enough freedom to enable cross-language implementations. The binding specifications then constrain those concepts in a manner appropriate for a particular language or platform. The purpose of the binding specifications is to repackage the DRMAA specification in a manner which is suitable for implementation.

3.1 Integration Overview

In order to implement a job submission, monitoring, and control API for N1 Grid Engine, the functionality of DRMAA was abstracted into an internal API called JAPI, or Job API. JAPI implements the core DRMAA functionality, with the DRMAA implementation being a relatively thin software layer built on top of JAPI. This separation of interface from implementation allowed the main command-line job submission utility, *qsub*, to be re-implemented on top of JAPI to share common functionality with the DRMAA implementation.

The DRMAA C binding implementation in N1 Grid Engine communicates directly with the grid via the internal Grid Engine Database Interface (GDI) and Event Client Interface (EVC). The binding implementation, and hence DRMAA-enabled applications running on top of the binding implementation, is a first class member of the grid, just like any other grid utility. As previously mentioned, the binding implementation actually shares much of its source with the main command-line job submission utility.

3.2 Wait Semantics

Since the release of N1 Grid Engine C and Java language binding implementations, several issues have been raised by ISVs and community members around the details of job synchronization. There were several cases where behavior was unclear, misdocumented, or non-intuitive. These

issues have all been addressed by the DRMAA working group.

3.3 Session Boundaries

A very common request from users of the binding implementations is for the ability to access jobs submitted outside of the current DRMAA session. The DRMAA specification allows for the *drmaa_synchronize()*, *drmaa_wait()*, *drmaa_job_ps()*, and *drmaa_control()* functions to access jobs submitted outside of the current DRMAA session. The fact that N1 Grid Engine DRMAA C binding implementation does not support this functionality is a short-coming of the implementation which will most probably be addressed in a future release.

3.4 Multi-threading and Reentrancy

The N1 Grid Engine DRM system DRMAA C binding implementation is multi-threaded, which was a requirement to implement the DRMAA job synchronization semantics without the use of polling, which can produce an excessive burden on the DRM. Additionally, DRMAA binding implementations are encouraged to implement the DRMAA routines as reentrant. Producing a reentrant, multi-threaded DRMAA C binding implementation which is free of deadlocks and does not impose excessive performance penalties was a challenging task. Several iterations were required to to produce a quality multi-threaded C binding implementation.

3.5 Performance

The performance of N1 Grid Engine DRMAA C binding implementation is a topic of great interest to many users. Unofficial testing has revealed that with the N1 Grid Engine 6.0 update 6 DRM system, the DRMAA C binding implementation is approximately ten times faster than using the *qsub* command-line utility with which the binding implementation shares much of its source. The main reason for the performance increase is that a DRMAA-enabled application only need go through the initialization process once. After that, it can submit as many jobs as desired. A traditional “fork and exec” approach results in the initialization procedure being carried out for every job submission.

4. DRMAA Compliance Test

The DRMAA working group published their first version of a DRMAA C compliance test suite in September 2005. The test suite is based on the N1 Grid Engine 6.0 DRM system DRMAA C binding implementation test code, and was extended and tuned during testing with the Condor DRMAA C binding implementation. The current version of the test suite reflects all functional properties of the specification to be submitted for GFD recommendation status (see material for the DRMAA C binding tests at http://sourceforge.net/projects/condor-ext).

See Appendix A for a description of the tests included in the test suite.

4.1 Test Report

N1 Grid Engine DRMAA C binding implementation was tested with the DRMAA test suite version 1.4.2.

Startup command	`./test_drmaa ALL_AUTOMATED $SGE_ROOT/examples/jobs/sleeper.sh test_exit_helper test_kill_helper root 2>&1 |tee log.txt`

Test platform:

Operating system	SunOS 5.9 Generic_112233-08 sun4u sparc SUNW,Sun-Blade-1000

N1 Grid Engine DRM system version	N1 Grid Engine 6.0 update 9
Testsuite compiled with	`cc -g -g -DDRMAA_10 -D__EXTENSIONS__ -xarch=v9 -I ../sge60u9/include -DHAVE_CONFIG_H -I./ -o test_drmaa test_drmaa.c -L../sge60u9/lib/sol-sparc64/ -ldrmaa -lpthread`
GCC version	`Configured with: ../src/configure -v --enable-languages=c,c++,java,f77,pascal,objc,ada,treelang --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-gxx-include-dir=/usr/include/c++/3.3 --enable-shared --enable-__cxa_atexit --with-system-zlib --enable-nls --without-included-gettext --enable-clocale=gnu --enable-debug --enable-java-gc=boehm --enable-java-awt=xlib --enable-objc-gc i486-linux CPPFLAGS="-DDRMAA_10 -D__EXTENSIONS__ -xarch=v9 -I ../sge60u9/include" Thread model: posix gcc version 3.3.5 (Debian 1:3.3.5-12)`

On the test platform the tests completed without error.

5. N1 Grid Engine End User Reports

Since release of DRMAA with N1 Grid Engine 6.0, a number of users and developers have reported success in developing applications with the API. A few of the more notable examples follow.

5.1 DRMAA Java language binding being used behind Web UI to run optimization applications

Miles Krivoshia from CombineNet Inc. reports: "We are responsible for CombineNet's new ASP offerings, including exposing our optimization engine via web services. That's one piece that uses Grid Engine. We also use it to handle optimization jobs from our own ASP web applications. CombineNet has a product that solves very difficult combinatorial problems, which tend to be quite compute intensive. The problems deal primarily with the procurement space today. Clients can partner with us to use our solver, and send us problems to optimize via a web service interface. Our own web applications, that implement reverse auction style events, also use the grid to run optimizations. Those optimization jobs are managed via N1 Grid Engine across a pool of cycle servers. We were very interested in using the DRMAA interface (as it's vendor neutral), and especially the java language bindings. We're using WebLogic on the front end, and have a middle layer on top of N1 Grid Engine, submitting jobs and waiting for results via Java DRMAA."

5.2 DRMAA C binding being used to provide integrated data grid solution

Benoit Marchand from eXludus Technologies Inc. reported about their RepliCator data transfer management software, which accelerates data movement for clusters and edge grids and thus optimizes their throughput. The RepliCator software binds to DRM systems such as N1 Grid Engine via DRMAA in order to implement RepliCator's novel data-activated processing mechanism.

5.3 DRMAA C binding connects a zOS mainframe system

Johann Geyer from WAVE Solutions Information Technology GmbH reports that the DRMAA C binding is being used to connect a zOS mainframe system with an N1 Grid Engine cluster by an IT finance service provider in Austria.

5.4 Requirement of a Perl API for DRM systems leads to the DRMAA CPAN module for Perl

Tim Harsch from Lawrence Livermore National Laboratory Bioinformatics group reports: "First, I love the DRMAA-C thing. My project, demanded a Perl API for conversing with a job scheduler, and I found DRMAA, and here I am."

6. Conclusion

The experience gathered in the successful implementation of the Distributed Resource Management Application API (DRMAA) specification GFD-R-P.022 for N1 Grid Engine is presented in this document. The DRMAA specifications have shown themselves to be both flexible and useful as demonstrated by the plethora of implementations and the variety of applications. Feedback from N1 Grid Engine customers has shown that DRMAA is a useful tool for developing both ISV and end-user software applications. It meets the demand for a standardized job submission interface for a wide variety of use cases and is being adopted by the ISV community. More over, the above feedback shows that there is a need for an interface which standardizes the feature set available with modern DRM systems, and DRMAA is being accepted as that interface, both in academic environments and in the IT industry.

7. Security Considerations

Security issues are not discussed in this document. For security considerations of the DRMAA specification, please refer to the GFD-R-P.022 document.

8. Contributors

Contact information for authors. You can also use this section to recognize contributions by people who are not listed on the title page, but made a useful contribution (it's OK to mention the nature of the contribution).

The actual Authors (or Editors) listed on the title page are those committed to taking permanent stewardship for this document – receiving communication in the future and otherwise being responsive to its content. The GFSG recommends at most three Author/Editors be listed on the title page, unless there are compelling reasons to list more.

9. Glossary

Recommended but not required.

10. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

11. Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

12. Full Copyright Notice

13. References

[1] Hrabri Rajic, Roger Brobst, Waiman Chan, Fritz Ferstl, Jeff Gardiner, Andreas Haas, Bill Nitzberg, and John Tollefsrud. Distributed Resource Management Application API Specification 1.0. http://forge.ggf.org/projects/drmaa-wg/, 2004.
[2] Sun Industry Standards Source License Version 1.2. Available at http://gridengine.sunsource.net/Gridengine_SISSL_license.html
[3] Andreas Haas, Roger Brobst, Nicholas Geib, Hrabri Rajic, Daniel Templeton, John Tollefsrud, Peter Tröger. Distributed Resource Management Application API C Bindings v1.0, presented at GGF13, February 2005

14. Appendix A: DRMAA C Compliance Test

The DRMAA C compliance test is an application which tests for binary compatibility of a DRMAA C binding implementation with the DRMAA C binding specification. The test application includes over 40 tests which test various aspects of compliance of the binding implementation. The test application also includes an automated test, which runs all of the other tests in series and reports comprehensively on compliance. This automated test is the test used to verify the compliance of the N1 Grid Engine 6.0 update 9 DRM system DRMAA C binding implementation.

14.1 ST_SUBMIT_WAIT

This test initializes the session and starts one thread that submits several single jobs and then waits individually for each job to end before exiting the session.

14.2 MT_SUBMIT_WAIT

This test initializes the session and starts several threads that simultaneously submit jobs. The main thread then waits individually for each job to end before exiting the session.

14.3 MT_SUBMIT_BEFORE_INIT_WAIT

This test starts several threads that simultaneously submit jobs, without first initializing the session.

After waiting a few seconds, the main thread initializes the session, waits for the submission threads to finish, and then waits individually for each job to end before exiting the session. All job submissions attempted before the session is initialized are expected to fail.

14.4 ST_MULT_INIT

This test attempts to initialize the session multiple times, with the expectation that the first attempt will succeed and all subsequent attempts will fail. It then exits the session.

14.5 ST_MULT_EXIT

This test initializes the session and then attempts to exit the session multiple times, with the expectation that the first attempt will succeed and all subsequent attempts will fail.

14.6 MT_EXIT_DURING_SUBMIT

This test initializes the session and then starts several threads that simultaneously submit jobs. After a short pause, the main thread exits the session, while the submission threads are still running, with the expectation that all submissions which happen after the session has been exited will fail.

14.7 MT_SUBMIT_MT_WAIT

This test initializes the session and starts several threads that simultaneously submit jobs and then wait for those jobs to end. The main thread then waits for the submission threads to finish before exiting the session.

14.8 MT_EXIT_DURING_SUBMIT_OR_WAIT

This test initializes the session and starts several threads that simultaneously submit jobs and then wait for those jobs to end. The main thread then pauses long enough for the threads to begin waiting for their jobs to end, and then exits the session, while the submission threads are still waiting, with the expectation that all waits which are in progress when the session is exited or which happen after the session has been exited will fail.

14.9 ST_BULK_SUBMIT_WAIT

This test initializes the session and starts one thread that submits several parametric jobs and then waits individually for each job task to end before exiting the session.

14.10 ST_BULK_SINGLESUBMIT_WAIT_INDIVIDUAL

This test initializes the session and starts one thread that submits several parametric jobs and several single jobs and then waits individually for each job task to end before exiting the session.

14.11 ST_SUBMITMIXTURE_SYNC_ALL_DISPOSE

This test initializes the session and starts one thread that submits several parametric jobs and several single jobs and then synchronizes against all jobs, disposing of the job exit information, before exiting the session.

14.12 ST_SUBMITMIXTURE_SYNC_ALL_NODISPOSE

This test initializes the session and starts one thread that submits several parametric jobs and several single jobs and then synchronizes against all jobs. After all jobs have ended, the main thread waits individually for each job before exiting the session.

14.13 ST_SUBMITMIXTURE_SYNC_ALLIDS_DISPOSE

This test initializes the session and starts one thread that submits several parametric jobs and several single jobs and then synchronizes against a list of all submitted jobs, disposing of the job exit information, before exiting the session.

14.14 ST_SUBMITMIXTURE_SYNC_ALLIDS_NODISPOSE

This test initializes the session and starts one thread that submits several parametric jobs and several single jobs and then synchronizes against a list of all submitted jobs. After all jobs have ended, the main thread waits individually for each job before exiting the session.

14.15 ST_INPUT_FILE_FAILURE

This test initializes the session and submits a single job with a bad input path. If the job submission succeeds, the test synchronizes against the job and then gets the job's current state. The test then waits for the job, check whether the job aborted, and exits the session. The expectation is that either 1) the job submission will fail, or 2) the job submission will succeed, the job's current state will be *FAILED*, and the job will have aborted.

14.16 ST_OUTPUT_FILE_FAILURE

This test initializes the session and submits a single job with a bad output path. If the job submission succeeds, the test synchronizes against the job and then gets the job's current state. The test then waits for the job, check whether the job aborted, and exits the session. The expectation is that either 1) the job submission will fail, or 2) the job submission will succeed, the job's current state will be *FAILED*, and the job will have aborted.

14.17 ST_ERROR_FILE_FAILURE

This test initializes the session and submits a single job with a bad error path. If the job submission succeeds, the test synchronizes against the job and then gets the job's current state. The test then waits for the job, check whether the job aborted, and exits the session. The expectation is that either 1) the job submission will fail, or 2) the job submission will succeed, the job's current state will be *FAILED*, and the job will have aborted.

14.18 ST_SUBMIT_IN_HOLD_RELEASE

This test initializes the session and submits a single job in a hold state. It then verifies that the job's state is *USER_HOLD* and releases the job. The test then synchronizes with the job, verifies that the job's state is *FINISHED*, and waits for the job before exiting the session.

14.19 ST_SUBMIT_IN_HOLD_DELETE

This test initializes the session and submits a single job in a hold state. It then verifies that the job's state is *USER_HOLD* and deletes the job. The test then synchronizes with the job, verifies that the job's state is *FAILED*, and waits for the job before exiting the session.

14.20 ST_BULK_SUBMIT_IN_HOLD_SINGLE_RELEASE

This test initializes the session and submits a parametric job in a hold state. Then for each job task, the test verifies that the job task's state is *USER_HOLD* and releases the job task. The test then synchronizes with the job, verifies that each job tasks's state is *FINISHED*, and waits for each job task to end before exiting the session.

14.21 ST_BULK_SUBMIT_IN_HOLD_SESSION_RELEASE

This test initializes the session and submits a parametric job in a hold state. Then for each job

task, the test verifies that the job task's state is *USER_HOLD* and releases all jobs in the session. The test then synchronizes with the job, verifies that each job tasks's state is *FINISHED*, and waits for each job task to end before exiting the session.

14.22 ST_BULK_SUBMIT_IN_HOLD_SESSION_DELETE

This test initializes the session and submits a parametric job in a hold state. Then for each job task, the test verifies that the job task's state is *USER_HOLD* and deletes all jobs in the session. The test then synchronizes with the job, verifies that each job tasks's state is *FAILED*, and waits for each job task to end before exiting the session.

14.23 ST_BULK_SUBMIT_IN_HOLD_SINGLE_DELETE

This test initializes the session and submits a parametric job in a hold state. Then for each job task, the test verifies that the job task's state is *USER_HOLD* and deletes the job task. The test then synchronizes with the job, verifies that each job tasks's state is *FAILED*, and waits for each job task to end before exiting the session.

14.24 ST_EXIT_STATUS

This test initializes the session and submits 255 single “exit” jobs, each with a different exit status. (An “exit job” is a job which can be configured to exit with a specific status code.) The test then waits individually for each job to end and checks that each job exited with the correct exit status before exiting the session.

14.25 ST_SUPPORTED_ATTR

This test initializes the session and gets the list of supported non-vector attributes before exiting the session.

14.26 ST_SUPPORTED_VATTR

This test initializes the session and gets the list of supported vector attributes before exiting the session.

14.27 ST_VERSION

This test initializes the session and gets the DRMAA C binding specification version number before exiting the session.

14.28 ST_CONTACT

This test gets the default contact information, initializes the session, gets the session contact information, and exits the session.

14.29 ST_DRM_SYSTEM

This test gets the default DRM system information, initializes the session, gets the session DRM system information, and exits the session.

14.30 ST_DRMAA_IMPL

This test gets the default DRMAA implementation information, initializes the session, gets the session DRMAA implementation information, and exits the session.

14.31 ST_EMPTY_SESSION_WAIT

This test initializes the session, waits for any job to finish, and then exits the session. The

expectation is that the wait will fail.

14.32 ST_EMPTY_SESSION_SYNCHRONIZE_DISPOSE

This test initializes the session, synchronizes against all jobs in the session, disposing of the job exit information, and then exits the session.

14.33 ST_EMPTY_SESSION_SYNCHRONIZE_NODISPOSE

This test initializes the session, synchronizes against all jobs in the session, and then exits the session.

14.34 ST_EMPTY_SESSION_CONTROL

This test initializes the session, attempts to suspend, resume, hold, release, and terminate all jobs in the session, and then exits the session.

14.35 ST_SUBMIT_SUSPEND_RESUME_WAIT

This test initializes the session and submits a single job. The state of the job is then polled until it becomes *RUNNING*. The test then suspends the job and checks that the job's state is *USER_SUSPENDED*. The test then resumes the job and checks that the job's state is *RUNNING*. The test then waits for the job to finish before exiting the session.

14.36 ST_SUBMIT_POLLING_WAIT_TIMEOUT

This test initializes the session and submits a single job. The test then waits for the job to finish by waiting repeatedly with a short time-out before exiting the session.

14.37 ST_SUBMIT_POLLING_WAIT_ZEROTIMEOUT

This test initializes the session and submits a single job. The test then waits for the job to finish by waiting repeatedly with an immediate time-out before exiting the session.

14.38 ST_SUBMIT_POLLING_SYNCHRONIZE_TIMEOUT

This test initializes the session and submits a single job. The test then synchronizes with the job to finish by synchronizing repeatedly with a short time-out before exiting the session.

14.39 ST_SUBMIT_POLLING_SYNCHRONIZE_ZEROTIMEOUT

This test initializes the session and submits a single job. The test then synchronizes with the job to finish by synchronizing repeatedly with an immediate time-out before exiting the session.

14.40 ST_ATTRIBUTE_CHANGE

This test initializes the session and creates a job template. Every attribute in the job template is then set to an initial value and then set to a second value. The test then checks that the values stored in the job template match the second values set. The test then exits the session.

14.41 ST_USAGE_CHECK

This test initializes the session, submits a single job, waits for the job to end, checks that the resource usage information was returned, and exits the session.

14.42 ST_UNSUPPORTED_ATTR

This test initializes the session, creates a job template, and attempts to set an unsupported non-

vector attribute value in the job template. The test then exits the session. The expectation is that the set will fail.

14.43 ST_UNSUPPORTED_VATTR

This test initializes the session, creates a job template, and attempts to set an unsupported vector attribute value in the job template. The test then exits the session. The expectation is that the set will fail.

14.44 ST_SUBMIT_KILL_SIG

This test initializes the session. For each of several POSIX signals, the test then submits a “kill job,” waits for the job to end, and tests that the job terminated on the correct signal. (A “kill job” is a job which will kill itself with a specified signal.) The test then exits the session.