GWD-E

Distributed Resource Management
Application API (DRMAA) Working Group

Peter Tröger, Hasso-Plattner-Institute
Becky Gietzel, University of Wisconsin-Madison

2/22/2007

Condor DRMAA 1.0 Implementation – Experience Report

Status of This Document

This document provides information to the Grid community about the adoption of the OGF specification GFD-R-P.022 in the Condor workload management 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 Condor workload management system. The document reports about issues that where identified during implementation and test of a DRMAA C library for Condor, which was evaluated successfully with the DRMAA working group compliance test for C bindings. We will give suggestions for improvement of the specification, mainly concerning readability of the GFD-R-P.022 specification document.

Contents

1 Introduction

The Distributed Resource Management Application API specification (GFD-R.P.022) [1] specifies a generalized API to distributed resource management systems (DRMS) 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 implementations for DRM systems (Sun Grid Engine, Condor, Torque) as well as different languages (C, Java, Perl, Python).

This document reports about experiences in the implementation of a DRMAA C library for the Condor workload management system.

2 The Condor DRM System

Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

While providing functionality similar to that of a more traditional batch queuing system, Condor's novel architecture allows it to succeed in areas where traditional scheduling systems fail. Condor can be used to manage a cluster of dedicated compute nodes (such as a "Beowulf" cluster). In addition, unique mechanisms enable Condor to effectively harness wasted CPU power from otherwise idle desktop workstations.

Condor can be used to build Grid-style computing environments that cross administrative boundaries. Condor's "flocking" technology allows multiple Condor compute installations to work together. Condor incorporates many of the emerging Grid-based computing methodologies and protocols. For instance, Condor-G is fully interoperable with resources managed by Globus.

Condor is the product of the Condor Research Project at the University of Wisconsin-Madison (UW), and was first installed as a production system in the UW-Madison Department of Computer Sciences nearly 15 years ago. This Condor installation has served since as a major source of computing cycles to UW-Madison faculty and students. According to usage statistics on a typical day, Condor delivers more than 650 CPU days to UW researchers. Additional Condor installations have been established over the years across the UW-campus and the world. Hundreds of organizations in industry, government, and academia have used Condor to establish compute installations ranging in size from a handful to up to over one thousand workstations.

The Condor system is distributed under the Condor public license, version 1.1 (October 30 2003) [2].

2.1 Implementation of the Condor DRMAA Library

The first version of the Condor DRMAA library was released with Condor 6.7.0, in April 2004. Nicholas Geib, a member of the Condor project team, implemented most of the specified DRMAA functionality. His work was based on the GFD-R-P.022 document, as well as on the DRMAA C language binding v0.9 specification. The nearly complete feature set of this library version enabled the first DRMAA programming tutorial at GGF 12 [3].

From October 2005, Peter Tröger finalized the Condor DRMAA library in a SourceForge project, under consideration of the latest DRMAA C language binding v1.0 specification [4]. The resulting fully DRMAA-compliant library version was released with Condor 6.7.14.

It is the general impression of the Condor DRMAA authors that the DRMAA specification is explained in great detail and with a clear focus on implementation issues and semantic. The successful DRMAA compliance test clearly proves the overall validity of the specification document. However, during detailed implementation work, some specific issues were identified, which are explained in the following sections. We give suggestions for according improvements of the spec.

2.2 Integration Overview

The integration of a job submission and monitoring API in Condor demanded a decision for the coupling approach to the Condor system daemons. As many other Condor-based tools, the DRMAA library utilizes the Condor command line tools for job submission and partially for status querying. While Condor does not provide an own steering API, it ensures parameter input and output compatibility between different Condor versions with a special format flag. It supports single and bulk job submission, which ensures that for each possible DRMAA job submission scenario there is only one *condor_submit* call needed.

Most of the monitoring information is taken from the job log file, which is situated on the submission host, written by the Condor *condor_schedd* daemon. The Condor library takes nearly all information about job status, error conditions, resource usage and job-related events from these files. There are only a few special cases where the command line monitoring tools (*condor_q*, *condor_status*) must be also considered.

A) Pluggability Concept

The Condor system has the concept of ‘job universes’ [5]. A universe describes a class of jobs, which should be treated in a specific manner by the system. The ‘standard’ universe acts as default, and contains all jobs which where compiled with the *condor_compile* command. This special compilation process allows jobs to be checkpointed and continued during their execution. The ‘vanilla’ universe is for binaries that do not support Condor checkpointing, e.g. existing rendering or computation software. Other universes support Globus job submission, parallel jobs or the execution of Java binaries.

The Condor DRMAA implementation in version 6.7.0 to 6.7.13 used the ‘standard’ universe for job submission. This decision lead to the fact that every DRMAA application needed to be re-compiled with *condor_compile* in order to be accepted for job submission. This behavior prevented the re-use of DRMAA applications binaries as well as pre-compiled language bindings on-top-of the C library (e.g. DRMAA Perl interface). Therefore it was decided to submit jobs to the ‘vanilla’ universe in the final version of the library. Users are now able to use their existing DRMAA applications in Condor without any change.

Since the DRMAA specification only suggests a late binding strategy at runtime, both solutions would have been in accordance to the specification. We suggest the DRMAA group to provide a strict statement whether the API interface should allow exchangeability of DRMAA

implementations at compile time, or at load time (LD_LIBRARY_PATH) respectively runtime. The last possibility would also demand a clarification of the search strategy for DRMAA libraries in a system, which could be realized similar to the ODBC driver manager approach.

B) Error code specifics

During the implementation of the Condor DRMAA library, multiple specification flaws regarding error conditions and the respective return codes where detected. All these issues where discussed either on the mailing list or in the DRMAA phone conferences. We are happy to see that all issues are collected on GridForge, and expect the discussion results to be reflected in an updated version of GFD-R-P.022.

C) Job state definition

The manipulation of job state is one of the major objectives in a unified job submission API. The *drmaa_control* function allows suspending or resuming a job. The DRMAA suspend activity is mapped to a call of *condor_hold*, which kills the job currently executed and puts it back again in the job queue as hold job. A *drmaa_control* call with the 'release' argument frees the job from its hold state, by using the according Condor functionality.

While this behavior seems not be the intended functionality, there is no other way to support the notion of job stopping and restarting in the vanilla universe. DRMAA leaves the interpretation of control semantics completely to the implementation; therefore the Condor way does not violate the specification. However, users might be surprised if a DRMAA suspend call leads to checkpointing in one DRMS, and to job termination on another DRMS. It might be a good idea to define some basic semantics for the *drmaa_control* operations, beside the flow of job state changes.

D) Job category

The DRMAA concept of job categories appears to be a useful paradigm, which is somehow comparable to the Condor idea of 'universe' settings. The actual Condor DRMAA implementation supports a DRMAA-specific configuration file that can contain additional submission settings for job categories.

In order to allow the DRM-independent usage of job categories, there should be a list of predefined job category names in the DRMAA specification. The Condor 'universes' can act as good example for high-level, but well understood operation modes of a DRM system (e.g. 'parallel', 'checkpointed', 'vanilla', 'Java').

E) Date Format

DRMAA defines an own date format for (maybe partial) time stamps. The time zone portion of this timestamp string is not compliant to the formulation in ISO 8601 / RFC 822, which defines the time zone format without a colon delimiter between local differential hours and minutes [6]. A change of this specification would allow an easy usage of C library functions like *strptime*, but breaks existing implementations. We therefore suggest this change for a post-1.0 version of the specification.

F) Remote Files

The DRMAA job template contains mandatory attributes for input, output, and error stream files. The attribute syntax is designed in a way that a user can specify a hostname as source for the file, by concatenating the host name with a colon delimiter and the file name. In case of a local file on the submission host, DRMAA still expects the delimiter colon as first character of the attribute string, which seems to be not necessary and was the source for a bug in the first Condor DRMAA version.

The definition of any host as source for the files is a feature not supported by the Condor DRMAA

implementation. The specification combines the availability of such a mechanism with the general availability of a file transfer mechanism in the DRMS (*drmaa_transfer_files* attribute). This seems to be a problematic design decision – Condor has support for the transfer of job-related files, but only between the submission host and the execution host.

For this reason, we suggest to make the host name portion of the attributes still optional, but independent from the availability of a *drmaa_transfer_files* attribute. Implementations should be able to identify the availability of this feature in another way. In addition, we suggest binding the colon delimiter in the attribute string to the existence of a host name portion.

G) API Description Format

The GFD-R-P.022 document is mostly written in a programming-language neutral way, using an IDL-like syntax for description. The Condor DRMAA implementation in C language therefore relies on combined information from the GFD-R-P.022 specification and the C language binding specification.

We clearly support the idea of a language-neutral specification, especially with the ongoing developments for different programming languages like Java, Python and Perl. However, we suggest the usage the official IDL description standard [7] for a post-1.0 version of the DRMAA specification, as already announced by the DRMAA group at GGF14 [8]. This allows syntactic checks of the specification, usage of numerous existing tools for language binding generation, and proven semantics for the IDL language constructs.

3 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 Sun Grid Engine 6 DRMAA test code, and was extended and tuned during the tests with the Condor DRMAA implementation. The current version of the testsuite reflects all functional properties of the specification to be submitted for GFD recommendation status.

3.1 Test protocol

The feature-complete Condor DRMAA implementation, available with Condor 6.7.14, was tested with the DRMAA test suite version 1.4.2. We performed 3 complete test runs on each platform. In all cases, all tests where successful.

Startup command

```
/test_drmaa ALL_AUTOMATED /bin/sleep test_exit_helper test_kill_helper
root 2>&1 |tee log.txt
```

Test platform 1

Operating system

Linux 2.6.8-2-686-smp #1 SMP Thu May 19 17:27:55 JST 2005
i686 GNU/Linux (Debian Sarge)

Condor version

$CondorVersion: 6.7.13 Nov 7 2005 $
$CondorPlatform: I386-LINUX_RH9 $

Testsuite compiled with

```
gcc -g -O2 -DHAVE_CONFIG_H -I. -o test_drmaa test_drmaa.c
-L. -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
Thread model: posix
gcc version 3.3.5 (Debian 1:3.3.5-12)
```

Test platform 2

Operating system

Linux 2.4.27-2-itanium-smp #1 SMP Tue Feb 1 18:19:12 MST 2005
ia64 GNU/Linux (Debian Sarge)

Condor version

$CondorVersion: 6.7.13 Nov 7 2005 $
$CondorPlatform: IA64-LINUX_SLES81 $

Testsuite compiled with

```
gcc -g -O2 -DHAVE_CONFIG_H -I. -o test_drmaa test_drmaa.c
-L. -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
--with-system-libunwind --enable-clocale=gnu
--enable-debug --enable-java-gc=boehm
--enable-java-awt=xlib --enable-objc-gc ia64-linux
Thread model: posix
gcc version 3.3.5 (Debian 1:3.3.5-13)
```

Test platform 3

Operating system

Darwin Kernel Version 8.3.0: Mon Oct 3 20:04:04 PDT 2005;
root:xnu-792.6.22.obj~2/RELEASE_PPC Power Macintosh powerpc

Condor version

$CondorVersion: 6.7.10 Aug 3 2005 $
$CondorPlatform: PPC-OSX_10_3 $

Testsuite compiled with

```
gcc -g -O2 -DHAVE_CONFIG_H -I. -o test_drmaa test_drmaa.c
-L. -ldrmaa –lpthread
```

GCC version

```
Target: powerpc-apple-darwin8
Configured with:
/private/var/tmp/gcc/gcc-5026.obj~19/src/configure
--disable-checking --prefix=/usr --mandir=/share/man
--enable-languages=c,objc,c++,obj-c++
--program-transform-name=/^[cg][^+.-]*$/s/$/-4.0/
--with-gxx-include-dir=/include/gcc/darwin/4.0/c++
--build=powerpc-apple-darwin8 --host=powerpc-apple-darwin8
--target=powerpc-apple-darwin8
Thread model: posix
gcc version 4.0.0 (Apple Computer, Inc. build 5026)
```

4 Conclusion

We presented our experiences in the successful implementation of the Distributed Resource Management Application API (DRMAA) specification GFD-R-P.022 for the Condor workload management system. The specification, together with the C language binding document v1.0, has shown its feasibility in order to provide a unified interface for job submission and monitoring in DRM systems. Recent user requests on the Condor mailing list show an increasing interest in a programming interface for Condor job submission. DRMAA enables application developers to rely on a standardized interface for DRMS usage, without taking care of the particular DRM system in use. Prominent examples for such projects are the high-level language binding implementations (Java, Python, Perl) and several commercial and research projects [9].

To prepare the GFD-R-P.022 for recommendation status, the group should concentrate on the improvement of the major description flaws (A, B). This is largely reflected in according GridForge tracker items.

All functional issues identified (E, F) did not prevent us from implementing a fully DRMAA-compliant library for the Condor system. We therefore suggest covering these issues in a past-1.0 version of the DRMAA specification. The third class of issues (C, D, G) must also be seen as a 'wishlist' for later versions of the document.

5 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.

6 Contributors

Peter Tröger
Hasso-Plattner-Institute, University of Potsdam
Prof.-Dr.-Helmert-Str. 2-.3
14482 Potsdam
Germany

Becky Gietzel
University of Wisconsin-Madison
Department of Computer Sciences
1210 W. Dayton St. Rm # 4237
Madison, WI 53706

7 Intellectual Property Statement

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.

8 Full Copyright Notice

9 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] Condor Team. Condor Public Licence Version 1.1. October 30, 2003. Available at http://www.cs.wisc.edu/condor/downloads/
[3] Material for the GGF 12 DRMAA Tutorial, available at https://forge.gridforum.org/docman2/ViewCategory.php?group_id=69&category_id=881
[4] 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
[5] Condor Project. Condor Manual. Available at http://www.cs.wisc.edu/condor/manual/
[6] David H. Crocker. Standard for the format of ARPA internet text messages (RFC 822). August 13, 1982.
[7] Object Management Group. Common Object Request Broker Architecture: Core Specification, Chapter 3, March 2004
[8] Daniel Templeton, Peter Tröger, Roger Brobst, Andreas Haas, Hrabri Rajic. Distributed Resource Management Application API - IDL Bindings 0.35. presented at GGF14. May 2005
[9] DRMAA Wiki, Information about DRMAA users. Available at http://www.drmaa.org/wiki/index.php/DrmaaUsers