

Poznan Supercomputing
and
Networking Center

DRMAA implementation
for
IBM LoadLeveler

Michał Matłoka

michal.matloka@student.put.poznan.pl

Date: June 30, 2010, Poznań

Contents

1	Introduction	1
2	Input Documents	1
3	Testbed	2
4	Mapping of DRMAA interface on LoadLeveler API	2
4.1	DRMAA job attributes mapping to LoadLeveler command file keywords	2
4.2	DRMAA states mapping	2
4.2.1	States given by drmaa_job_ps	2
4.2.2	States acquired by monitor_program	3
4.3	drmaa_control operations mapping	4
4.4	Feasibility study, getting exit status and signals	4
5	Test LoadLeveler programs	4
5.1	Run job and wait for its end	4
5.2	Run bulk jobs	4
5.3	Check job status	5
5.4	Control job	5
6	DRMAA library implementation using DRMAA Utils	5
7	End User Manual	5
8	DRMAA testsuite	5
8.1	Checkpointing	5
9	Scalability testsuite	6

1 Introduction

PSNC DRMAA for LoadLeveler is an implementation of Open Grid Forum DRMAA 1.0 (Distributed Resource Management Application API) specification for submission and control of jobs to IBM Tivoli LoadLeveler. Using DRMAA, grid applications builders, portal developers and ISVs can use the same high-level API to link their software with different cluster/resource management systems.

This software also enables the integration of SMOA Computing with the underlying LoadLeveler system for remote multi-user job submission and control over Web Services.

2 Input Documents

- DRMAA 1.0 Grid Recommendation '
<http://www.ogf.org/documents/GFD.133.pdf>
- DRMAA C Binding v1.0
https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.drmaa-wg/docman.root.ggf_13/doc5545
- IBM LoadLeveler Documentation
<http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246038.html?OpenDocument>
- LONI Documentation
https://docs.loni.org/wiki/Main_Page

3 Testbed

LONI (Louisiana Optical Network Initiative) LoadLeveler clusters were used during the work.

4 Mapping of DRMAA interface on LoadLeveler API

4.1 DRMAA job attributes mapping to LoadLeveler command file keywords

IBM LoadLeveler uses command files for every submitted job. This affects both LL command line and LL API. Because of that for every submitted job is created temporary command file /tmp/drmaa_cmd_XXXXXX where every X is replaced by randomly chosen character so that generated name is unique. After submission the file is deleted.

DRMAA	LoadLeveler	Comment
drmaa_block_email	notification never	
drmaa_deadline_time	OPTIONAL ATTRIBUTE	NOT IMPLEMENTED
drmaa_duration_hlimit	OPTIONAL ATTRIBUTE	NOT IMPLEMENTED
drmaa_duration_slimit	OPTIONAL ATTRIBUTE	NOT IMPLEMENTED
drmaa_error_path	error	
drmaa_input_path	input	
drmaa_job_category	library configuration, putting appropriate line from .ll_drmaa.conf into cmd file	Simplified syntax "@keyword1 = value1 value2 @keyword2 = value3"
drmaa_job_name	job_name	
drmaa_join_files	same values of error and output	
drmaa_js_state	hold	
drmaa_native_specification	putting argument value into cmd file	Simplified syntax "@keyword1 = value1 value2 @keyword2 = value3"
drmaa_output_path	output	
drmaa_remote_command	executable	
drmaa_start_time	startdate	
drmaa_transfer_files	OPTIONAL ATTRIBUTE	NOT IMPLEMENTED, practically not used because each cluster has shared file system
drmaa_v_argv	arguments	
drmaa_v_email	notify_user	
drmaa_v_env	environment	
drmaa_wct_hlimit	wall_clock_limit	
drmaa_wct_slimit	wall_clock_limit	value given after comma
drmaa_wd	initialdir	

4.2 DRMAA states mapping

4.2.1 States given by drmaa_job_ps

The DRMAA states list was compared with LoadLeveler states retrieved by API and with those returned by llq command.

LoadLeveler	DRMAA	Comment
Canceled	DRMAA_PS_FAILED	
Checkpointing	DRMAA_PS_RUNNING	
Completed	DRMAA_PS_DONE	
Complete Pending		Transient state
Deferred	DRMAA_PS_QUEUED_ACTIVE	
Idle	DRMAA_PS_QUEUED_ACTIVE	Used with hold_type. If value !=0 then we have DRMAA_PS_X_ON_HOLD
Not Queued	DRMAA_PS_FAILED	
Not run	DRMAA_PS_FAILED	
Pending		Transient state
Preempted	DRMAA_PS_SYSTEM_SUSPENDED	SYSTEM_SUSPENDED because normal user cannot suspend a job
Preempt Pending		Transient state
Rejected	DRMAA_PS_FAILED	
Rejected Pending		Transient state
Removed	DRMAA_PS_FAILED	
Remove Pending		Transient state
Resume Pending		Transient state
Running	DRMAA_PS_RUNNING	
Starting		Transient state
System Hold	DRMAA_PS_SYSTEM_ON_HOLD	
Terminated	DRMAA_PS_FAILED	
User & System Hold	DRMAA_PS_USER_SYSTEM_ON_HOLD	
User Hold	DRMAA_PS_USER_ON_HOLD	
Vacated	DRMAA_PS_FAILED	ll drmaa option: terminate_job_on_vacated - if true when job will get vacated state will be killed
Vacate Pending		Transient state

For transient states is returned:

- last stored state
- DRMAA_PS_QUEUED_ACTIVE when no state where previously remembered is returned.

4.2.2 States acquired by monitor_program

LoadLeveler notifies about submitted job state changes by monitor_program (The monitor_program path can be specified as ll_submit command argument). The monitor_program shipped with the DRMAA library communicates with DRMAA library main process by a UNIX socket (/tmp/drmaa_socket_XXXXXX for security reasons 0600).

LoadLeveler	DRMAA	Comment
JOB_STARTED	DRMAA_PS_RUNNING	
JOB_COMPLETED	DRMAA_PS_DONE	
JOB_VACATED	DRMAA_PS_FAILED	Option terminate_job_on_vacated
JOB_REJECTED	DRMAA_PS_FAILED	
JOB_REMOVED	DRMAA_PS_FAILED	
JOB_NOTRUN	DRMAA_PS_FAILED	

4.3 drmaa_control operations mapping

DRMAA	LoadLeveler	Comments
DRMAA_CONTROL_SUSPEND	ll_preempt_jobs - PREEMPT_STEP	Administrators only
DRMAA_CONTROL_RESUME	ll_preempt_jobs - RESUME_STEP	Administrators only
DRMAA_CONTROL_HOLD	ll_control - LL_CONTROL_HOLD_USER	
DRMAA_CONTROL_RELEASE	ll_control - LL_CONTROL_HOLD_RELEASE	
DRMAA_CONTROL_TERMINATE	ll_terminate_job	

4.4 Feasibility study, getting exit status and signals

monitor_program gives exit status. However its value needs modifications before using in DRMAA. There are two possibilities:

- `exit_status >> 8;`
`signal_number >> 16;`
- Using sys/wait.h macros
 - WIFEXITED
 - WEXITSTATUS
 - WIFSIGNALED
 - WTERMSIG
 - WCOREDUMP

All this macros were used in final LL DRMAA implementation.

5 Test LoadLeveler programs

These programs realizes basic DRMAA functionalities using LoadLeveler API. They have been written in order to test LL API specific aspects and check is it possible to implement DRMAA library based on LL API. Source codes are available at LL DRMAA SVN, experiments folder. <http://gforge.man.poznan.pl/svn/smoaincubator/drmaa/ll/experiments/>

5.1 Run job and wait for its end

This program creates simple command file, submits job with monitor_program location set as its argument and reads FIFO pipe (communication with monitor_program). Final version of LL DRMAA uses Unix Sockets. Data retrieved by monitor_program:

- argument given by submit function
- job_id
- job_state
- exit_status

5.2 Run bulk jobs

Submitting bulk jobs is possible by adding additional @queue keywords in job command file. It is also necessary to replace DRMAA_PH_INCR, DRMAA_PH_WD, DRMAA_PH_HD.

5.3 Check job status

This program tests the following LL routines:

- `ll_query`
- `ll_set_request`
- `ll_get_objs`
- `ll_get_data`
- `ll_next_obj`

Those functions can not retrieve informations about jobs which have ended.

5.4 Control job

Realized according to proposed mapping.

6 DRMAA library implementation using DRMAA Utils

Library was implemented in C based on the implementation of DRMAA for LSF (<http://gforge.man.poznan.pl/svn/smoaincubator/drmaa/lsf/>) and tools like GNU Compiler Collection and GNU Project Debugger accordingly to created mappings.

7 End User Manual

Documentation can be found in doc folder in the library package.

8 DRMAA testsuite

Library covers all DRMAA 1.0 specification with exceptions listed below. It was successfully tested with IBM Tivoli LoadLeveler 3.5.0.5 on AIX OS and passes 43/44 tests of the official DRMAA test-suite. All mandatory and nearly all optional job attributes (except job run duration soft limit, job run duration hard limit, `drmaa_transfer_files` and `drmaa_deadline_time`) are implemented.

Known limitations:

- `drmaa_control()` - `DRMAA_CONTROL_RESUME` and `DRMAA_CONTROL_SUSPEND` are not implemented as suspending jobs in LoadLeveler requires administrator privileges.

There are plans, depending on the end users feedback, to implement the `RESUME/SUSPEND` functionality by leveraging the checkpointing mechanism.

8.1 Checkpointing

Because LoadLeveler does not allow non-administrators to suspend their jobs appeared idea to use LL checkpointing mechanism. However this functionality has some limitations:

- job can not be in idle or hold states
- job must have its all descriptors closed
- it is only available under AIX OS

Using LoadLeveler API user might prepare his/her application for checkpointing. `ll_set_ckpt_callbacks` gives possibility to define functions which will be performed when specific event occurs

- checkpoint - need of saving and closing descriptors
- resume from checkpoint - open descriptors
- restart

This functionality was not implemented in this version of the library.

9 Scalability testsuite

A dedicated test program was written for scalability testing purposes. It simulates SMOA Computing service under continuous workload.

This program takes 4 arguments: SUB_INT, SLEEP_TIME, POLL_INTERVAL, WAIT_INTERVAL and starts 3 threads:

- first thread submits sleep job with SLEEP_TIME parameter
- second thread every POLL_INTERVAL checks status of every job
- third thread with WAIT_INTERVAL (timeout) and SESSION_ANY (job) parameters runs `drmaa_wait` and updates finished jobs list.