DRMAAv2 - An Introduction

Peter Tröger
Hasso-Plattner-Institute, University of Potsdam
peter@troeger.eu

DRMAA-WG Co-Chair

http://www.drmaa.org/
History

• DRMAA group established in 2002

• Goal: Standardized API for distributed resource management systems (DRMS)
  
  • Low-level **portability** for cluster and grid infrastructure applications

  • Simple design, **bottom-up philosophy**

• Different DRMAA 1.0 documents

  • June 2004 - DRMAA 1.0 Proposed Recommendation (GFD.22)

  • April 2008 - Shift to IDL based root specification, some clarifications (GFD.130)

  • June 2008 - DRMAA 1.0 Grid Recommendation (GFD.133)

  • Official language binding documents for C, Java, Python (GFD.143)

  • Experience reports, tutorials, unofficial language bindings for Perl, Ruby and C#
OGF Specs

Features

Portability

End User Application / Portal

SAGA API / OGSA / OCCI

- DRMAA API
- Other OGF Standards
- Proprietary API

Meta Scheduler

SAGA API + Backends

- DRMAA API
- Other OGF Standards
- Proprietary API

Grid Engine

Load Leveler

...
• Product-quality implementations of DRMAA C and Java, broad industrial up-take
  
  • *Grid Engine* since 2006
  
  • *Condor* since 2006
  
  • *PBS/Torque, LSF, LoadLeveler, Kerrighed, SLURM, Globus* (via *GridWay*)
    (third-party implementations)

• Large and silent user community

  • Meta schedulers accessing DRM resources
    (Galaxy, QosCosGrid, MOAB, eXludus, OpenDSP, EGEE, Unicore, SAGA, ...)

  • Applications or portals accessing DRM resources
    (BLAST, workflow processing, finance sector, seismic data, Mathematica, ...)

  • Hundreds (?) of unknown Grid Engine users
#include "drmaa.h"

int main(int argc, char **argv) {
    char error[DRMAA_ERROR_STRING_BUFFER];
    int errnum = 0;
    drmaa_job_template_t *jt = NULL;

    errnum = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER);
    if (errnum != DRMAA_ERRNO_SUCCESS) return 1;
    errnum = drmaa_allocate_job_template(&jt, error, DRMAA_ERROR_STRING_BUFFER);
    if (errnum != DRMAA_ERRNO_SUCCESS) return 1;
    drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, "sleeper.sh", error, DRMAA_ERROR_STRING_BUFFER);
    drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, args, error, DRMAA_ERROR_STRING_BUFFER);
    char jobid[DRMAA_JOBNAME_BUFFER];
    errnum = drmaa_run_job(jobid, DRMAA_JOBNAME_BUFFER, jt, error, DRMAA_ERROR_STRING_BUFFER);
    if (errnum != DRMAA_ERRNO_SUCCESS) return 1;
    errnum = drmaa_delete_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER);
    errnum = drmaa_exit(error, DRMAA_ERROR_STRING_BUFFER);
    return 0;
}
DRMAA v1 Got Old

• DRM systems got better
  • Concept of resources, session persistency, advance reservation, parallel jobs, many-core systems, queues, state models, WS-* stuff, ...

• Some obsolete / never implemented features
  • Date / time handling, host-to-host file staging, specialized job states, ...

• Awkward design decisions
  • Job synchronization, job monitoring, data reaping, native specification, ...

• DRMAAav2 work started 2009
  • Public survey, Sun customer feedback, implementation experiences, ...
  • Close collaboration with SAGA, GAT, and OCCI WG; considered JSDL & BES
  • Performant investigation of current DRM systems (poor Mariusz)
DRMAAv2 Design Approach

• All behavioral aspects in the IDL-based root specification
  • Same model as W3C DOM, OGF SAGA, ...
  • What functions are offered? How are they grouped?
  • What are possible error conditions?
• For language binding designers and implementors
  • Language binding just defines syntactical mapping
    • „DRMAA IDL construct A maps to programming language construct B“
    • Root spec demands some design decisions (e.g. „UNSET“)
• End users should get separate man pages
Establish session with the DRM system

Work with jobs / advance reservation / monitoring features of the DRM system
Optional vs. Implementation-Specific

- Make non-mandatory things explicit
- `DrmaaS Capability` enumeration
  - One enum per optional feature + `SessionManager::supports` method
- `DrmaaS Reflective` interface
  - Lists of implementation-specific attributes + generic getter / setter functions

```cpp
enum DrmaaCapability {
    ADVANCE_RESERVATION, RESERVE_SLOTS, CALLBACK, BULK_JOBS_MAXPARALLEL, JT_EMAIL,
    JT_STAGING, JT_DEADLINE, JT_MAXSLOTS, JT_ACCOUNTINGID, RT_STARTNOW, RT_DURATION,
    RT_MACHINEOS, RT_MACHINEARCH
};

interface DrmaaReflective {
    readonly attribute StringList jobTemplateImplSpec;
    readonly attribute StringList jobInfoImplSpec;
    ...

    string getInstanceValue(in any instance, in string name);
    void setInstanceValue(in any instance, in string name, in string value);
    string describeAttribute(in any instance, in string name);
}
```
Stakeholders

- Distributed resource management (DRM) system
  - Distributes computational tasks on execution resources
  - Central scheduling entity
- DRMAA implementation / library: Implementation of a language binding, semantics as in the root spec
- Submission host: Resource that runs the DRMAA-based application
- Execution host: Resource that can run a submitted computational task
- Job: One or more operating system processes on a execution host
DRMAAv2 Session Manager

- Create multiple connections to one (or more) DRM system(s) at a time
  - JobSession / ReservationSession
    - Persistent storage of session state (at least) on submission machine
    - Can be reloaded by name, explicit reaping
  - MonitoringSession
    - Read-only semantic, global view on all machines

- Maps nicely to SAGA and friends - management vs. monitoring
- Security remain out of scope for DRMAA
- Explicitly supports the portal / command-line tools use case
- Old concept of contact strings from DRMAAv1
DRMAAv2 Session Manager

interface SessionManager{
    readonly attribute string drmsName;
    readonly attribute Version drmsVersion;
    readonly attribute string drmaaName;
    readonly attribute Version drmaaVersion;
    boolean supports(in DrmaaCapability capability);
    JobSession createJobSession(in string sessionName, in string contact);
    ReservationSession createReservationSession(
        in string sessionName, in string contact);
    JobSession openJobSession(in string sessionName);
    ReservationSession openReservationSession(in string sessionName);
    MonitoringSession openMonitoringSession (in string contact);
    void closeJobSession(in JobSession s);
    void closeReservationSession(in ReservationSession s);
    void closeMonitoringSession(in MonitoringSession s);
    void destroyJobSession(in string sessionName);
    void destroyReservationSession(in string sessionName);
    StringList getJobSessionNames();
    StringList getReservationSessionNames();
    void registerEventNotification(in DrmaaCallback callback);
};
DRMAAv2 Event Callback

- Optional support for event push notification
- Application implements DrmaaCallback interface
- Demands some additional rules from the language binding
- Push notification at least supported in Grid Engine
- Heavily demanded by SAGA and end users
- Polling by the library is allowed
- Nested callbacks are forbidden
- Scalability is out of scope

```c
interface DrmaaCallback {
    void notify(in DrmaaNotification notification);
};
struct DrmaaNotification {
    DrmaaEvent event;
    string jobId;
    string sessionName;
    JobState jobState;
};
enum DrmaaEvent {
    NEW_STATE, MIGRATED, ATTRIBUTE_CHANGE
};
```
DRMAA v2 Job Session

• New: Fetch list of supported job categories (portal case)

• New: Job filtering

• New: JobArray

• New: maxParallel restriction for bulk jobs

• Old synchronize() was replaced by class-based state waiting approach

  • More responsive

  • Get rid of partial job failure problems

interface JobSession {
  readonly attribute string contact;
  readonly attribute string sessionName;
  readonly attribute StringList jobCategories;
  JobList getJobs(in JobInfo filter);
  JobArray getJobArray(in string jobArrayId);
  Job runJob(in JobTemplate jobTemplate);
  JobArray runBulkJobs(in JobTemplate jobTemplate, in long beginIndex, in long endIndex, in long step, in long maxParallel);
  Job waitAnyStarted(in JobList jobs, in TimeAmount timeout);
  Job waitAnyTerminated(in JobList jobs, in TimeAmount timeout);
}

native ZERO_TIME;
native INFINITE_TIME;
Job Categories

- Consider deployment properties
  - Path settings, environment variables, software installed, application starters, ...

- Job category references a site-specific configuration

- Non-normative set of recommendations on DRMAA home page
  - Idea is to perform updates without spec modification
  - Site operators should be allowed to create their own job categories (configuration of the implementation)
  - Initial set derived from JSDL SPMD Extension (GFD.115) + experience
  - Proposals welcome ...
Everybody Loves State Models

- Number of states reduced
- New sub-state concept
  - Similar to OGSA-BES
  - Allows simple DRMAAv1 mapping
- Wait functions work only on class level (timing issues)
- No more different hold modes
DRMAAv2 Job Template

• Basic rules
  • DRM systems manage machines, which have a CPU and physical memory
  • Machines can be booked in advance reservation
  • DRM systems manage queues and slots as resources, opaque to DRMAA

• Jobs can be submitted ...
  • ... to specified candidate machines or a queue
  • ... to machines matching an OS type, architecture type, or memory requirement
  • ... as 'classified' job with special treatment by the DRMS (jobCategory)
    • Examples: MPICH2 job, OpenMPI job, OpenMP job
DRMAAv2 Job Template

- Most things remain the same, but:
  - Relative start / end times are gone, switched to RFC822
  - Only copying between submission and exception machine, no host names
  - Standardized job category names (based on GFD.115, see web page)
  - Standardized resource limit types (*setrlimit*)
  - Several new resource specification attributes

```c
struct JobTemplate {  
  string remoteCommand;  
  OrderedStringList args;  
  boolean submitAsHold;  
  boolean rerunnable;  
  Dictionary jobEnvironment;  
  string workingDirectory;  
  string jobCategory;  
  StringList email;  
  boolean emailOnStarted;  
  boolean emailOnTerminated;  
  string jobName;  
  string inputPath;  
  string outputPath;  
  string errorPath;  
  boolean joinFiles;  
  string reservationId;  
  string queueName;  
  long minSlots;  
  long maxSlots;  
  long priority;  
  OrderedStringList candidateMachines;  
  long minPhysMemory;  
  OperatingSystem machineOS;  
  CpuArchitecture machineArch;  
  AbsoluteTime startTime;  
  AbsoluteTime deadlineTime;  
  Dictionary stageInFiles;  
  Dictionary stageOutFiles;  
  Dictionary resourceLimits;  
  string accountingId;
};
```
DRMAAv2 Job / JobArray

- Heavy cleanup
  - Dedicated control methods
  - Explicit job information fetching

waitStarted() and waitTerminated() as on JobSession level

- Support for the new job sub-state model

- JobArray offers control operations on bulk jobs

- Rejected: Signaling, modifying running jobs, ...

```java
interface Job {
    readonly attribute string jobId;
    readonly attribute string sessionName;
    readonly attribute JobTemplate jobTemplate;
    void suspend();
    void resume();
    void hold();
    void release();
    void terminate();
    JobState getState(out any jobSubState);
    JobInfo getInfo();
    Job waitStarted(in TimeAmount timeout);
    Job waitTerminated(in TimeAmount timeout);
}

interface JobArray {
    readonly attribute string jobArrayId;
    readonly attribute JobList jobs;
    readonly attribute string sessionName;
    readonly attribute JobTemplate jobTemplate;
    void suspend();
    void resume();
    void hold();
    void release();
    void terminate();
}
```
DRMAAv2 JobInfo

• No surprises with JobInfo
  • Used for both filtering and job status representation
  • Snapshot semantics
  • No promises for terminated jobs

• Implementations can give a meaning to the reported list of machines being used

```c
struct JobInfo {
  string jobId;
  long exitStatus;
  string terminatingSignal;
  string annotation;
  JobState jobState;
  any jobSubState;
  OrderedSlotInfoList allocatedMachines;
  string submissionMachine;
  string jobOwner;
  long slots;
  string queueName;
  TimeAmount wallclockTime;
  long cpuTime;
  AbsoluteTime submissionTime;
  AbsoluteTime dispatchTime;
  AbsoluteTime finishTime;
};
```
DRMAAv2 Advance Reservation

• Completely new
• Fits nicely to the rest of the spec

interface ReservationSession {
    readonly attribute string contact;
    readonly attribute string sessionName;
    Reservation getReservation(in string reservationId);
    Reservation requestReservation(in ReservationTemplate reservationTemplate);
    ReservationList getReservations();
};

interface Reservation {
    readonly attribute string reservationId;
    readonly attribute string sessionName;
    readonly attribute ReservationTemplate reservationTemplate;
    ReservationInfo getInfo();
    void terminate();
};
DRMAAv2 Advance Reservation

- Reservation template

  - Different combinations of startTime, endTime, and duration are allowed

- Sliding window + time frame support

- Support for authorization setup

```
struct ReservationTemplate {
  string reservationName;
  AbsoluteTime startTime;
  AbsoluteTime endTime;
  TimeAmount duration;
  long minSlots;
  long maxSlots;
  string jobCategory;
  StringList usersACL;
  OrderedStringList candidateMachines;
  long minPhysMemory;
  OperatingSystem machineOS;
  CpuArchitecture machineArch;
};

struct ReservationInfo {
  string reservationId;
  string reservationName;
  AbsoluteTime reservedStartTime;
  AbsoluteTime reservedEndTime;
  StringList usersACL;
  OrderedSlotInfoList reservedMachines;
};
```

```
struct SlotInfo {
  string machineName;
  long slots;
};
...
typedef sequence<SlotInfo> OrderedSlotInfoList;
...
DRMAAv2 Monitoring

- Completely new
- Global view on resources
- Stateless session instance
- Implementation has freedom to leave out information
- Resource information model matching to SAGA
- Virtual memory = physical memory + swap space
- Load = 1-min average load
- Snapshot semantics

```c
typedef sequence <Reservation> ReservationList;
typedef sequence <Job> JobList;
typedef sequence <QueueInfo> QueueInfoList;
typedef sequence <MachineInfo> MachineInfoList;
...
interface MonitoringSession {
    ReservationList getAllReservations();
    JobList getAllJobs(in JobInfo filter);
    QueueInfoList getAllQueues(in StringList names);
    MachineInfoList getAllMachines(in StringList names);
};
struct QueueInfo {
    string name;
};
struct MachineInfo {
    string name;
    boolean available;
    long sockets;
    long coresPerSocket;
    long threadsPerCore;
    double load;
    long physMemory;
    long virtMemory;
    OperatingSystem machineOS;
    Version machineOSVersion;
    CpuArchitecture machineArch;
};
```
More Stuff

• **DRMAA_INDEX_VAR**

  • Implementations should set the environment variable **DRMAA_INDEX_VAR**
  
  • Contains the name of the DRMS environment variable providing the job index
    
    • **TASK_ID** (Grid Engine), **PBS_ARRAYID** (Torque), **LSB_JOBINDEX** (LSF)
  
  • Jobs are enabled to get their own parametric index
## Notes: Job Specification Is Hard

<table>
<thead>
<tr>
<th>Resource requirement for job</th>
<th>JSDL</th>
<th>SGE Queue Properties</th>
<th>SGE Description</th>
<th>Condor Machine ClassAd</th>
<th>Condor submission file</th>
</tr>
</thead>
<tbody>
<tr>
<td>CandidateHosts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalResourceCount</td>
<td></td>
<td>slots</td>
<td>Number of processes (allowed) to run</td>
<td>MACHINE</td>
<td></td>
</tr>
<tr>
<td>FileSystem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>machine_count</td>
</tr>
<tr>
<td>ExclusiveExecution</td>
<td></td>
<td>(This is accomplished through a special complex as of 0.2.3.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OperatingSystem</td>
<td></td>
<td>(arch built-in complex)</td>
<td></td>
<td>OPSYS</td>
<td></td>
</tr>
<tr>
<td>CPUArchitecture</td>
<td></td>
<td>(arch built-in complex)</td>
<td></td>
<td>ARCH</td>
<td></td>
</tr>
<tr>
<td>IndividualCPUSpeed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IndividualCPUTime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IndividualCPUCount</td>
<td></td>
<td>(num_proc built-in complex)</td>
<td></td>
<td>CPUS</td>
<td></td>
</tr>
<tr>
<td>IndividualNetworkBandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IndividualPhysicalMemory</td>
<td></td>
<td>(mem_total built-in complex)</td>
<td></td>
<td>MEMORY</td>
<td></td>
</tr>
<tr>
<td>IndividualVirtualMemory</td>
<td></td>
<td>(virtual_total built-in complex)</td>
<td></td>
<td>VirtualMemory</td>
<td></td>
</tr>
<tr>
<td>IndividualDiskSpace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalCPUTime</td>
<td></td>
<td>s_cpu / h_cpu</td>
<td>Soft / hard limit for CPU time of all processes</td>
<td>DISK</td>
<td></td>
</tr>
<tr>
<td>TotalCPUCount</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalPhysicalMemory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalVirtualMemory</td>
<td></td>
<td>s_vmem / h_vmem</td>
<td>Soft / hard limit for job virtual memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalDiskSpace</td>
<td></td>
<td>s_fsize / h_fsize</td>
<td>Soft / hard limit for bytes on disk</td>
<td></td>
<td>Minimum time between</td>
</tr>
</tbody>
</table>
Notes: Job Submission Is Hard

<table>
<thead>
<tr>
<th></th>
<th>LSF</th>
<th>Torque</th>
<th>PBS Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wildcard Support</td>
<td>no</td>
<td>not recommended</td>
<td>yes</td>
</tr>
<tr>
<td>Other than submission host</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Appending file</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Directory Staging</td>
<td>no</td>
<td>not by default</td>
<td>yes</td>
</tr>
</tbody>
</table>

- Ensuring true application portability with a unified API is REALLY hard
  - Interoperability (OGSA, JSDL) does not provide portability
  - Profiles (!) with „SHOULD“ and „UnsupportedFeatureFault“ are not helpful
- DRMAA tries to define **mandatory** job template attributes and API functions that are **implementable** in most DRM systems
  - This is why DRMAA will never by exhaustive -> use SAGA !
Notes: Other OGF Specs

- JSDL is partially exhaustive, GLUE is conservative

<table>
<thead>
<tr>
<th>DRMAA OperatingSystem</th>
<th>JSDL jsdl:OperatingSystemTypeEnumeration</th>
<th>GLUE v2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPUX</td>
<td>HPUX</td>
<td>OSFamily.t:linux</td>
</tr>
<tr>
<td>LINUX</td>
<td>LINUX</td>
<td>OSFamily.t:macosx</td>
</tr>
<tr>
<td>IRIX</td>
<td>IRIX</td>
<td>OSFamily.t:solaris</td>
</tr>
<tr>
<td>TRUE64</td>
<td>Tru64.UNIX, OSF</td>
<td>OSFamily.t:windows</td>
</tr>
<tr>
<td>MACOS</td>
<td>MACOS</td>
<td>OSFamily.t:windows</td>
</tr>
<tr>
<td>SUNOS</td>
<td>SunOS, SOLARIS</td>
<td>OSName.t:aix</td>
</tr>
<tr>
<td>WIN</td>
<td>WIN95, WIN98, Windows.R.Me</td>
<td></td>
</tr>
<tr>
<td>WINNT</td>
<td>WINNT, Windows.2000, Windows.XP</td>
<td></td>
</tr>
<tr>
<td>AIX</td>
<td>AIX</td>
<td></td>
</tr>
<tr>
<td>UNIXWARE</td>
<td>SCO.UnixWare, SCO.OpenServer</td>
<td></td>
</tr>
<tr>
<td>BSD</td>
<td>BSDUNIX, FreeBSD, NetBSD, OpenBSD</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DRMAA CpuArchitecture</th>
<th>JSDL jsdl:ProcessorArchitectureEnumeration</th>
<th>GLUE v2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHA</td>
<td>other</td>
<td>Platform.t:i386</td>
</tr>
<tr>
<td>ARM</td>
<td>arm</td>
<td>Platform.t:amd64</td>
</tr>
<tr>
<td>CELL</td>
<td>other</td>
<td>Platform.t:itanium</td>
</tr>
<tr>
<td>PARISC</td>
<td>parisc</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>X86</td>
<td>x86.32</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>X64</td>
<td>x86.64</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>IA64</td>
<td>ia64</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>MIPS</td>
<td>mips</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>PPC</td>
<td>powerpc</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>PPC64</td>
<td>powerpc</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>SPARC</td>
<td>sparc</td>
<td>Platform.t:powerpc</td>
</tr>
<tr>
<td>SPARC64</td>
<td>sparc</td>
<td>Platform.t:powerpc</td>
</tr>
</tbody>
</table>
DRMAAv2 Implementations?

- C language binding discussion in the next session

- Implementations announced
  - Fork-based reference implementation (DRMAA Working Group)
  - Univa Grid Engine (Daniel Gruber)
  - Torque / PBS Pro (Mariusz Mamonski)
  - GridWay (Eduardo Huedo)

- Collection type handling in C should be re-usable

- Interfacing the DRM system is already part of the DRMAAv1 library
  - Just add more stuff (job arrays, session persistency, reservation, monitoring)
  - Restructure according to the new header file
Thanks to the Core Team !

• Roger Brobst - Cadence Design Systems, Inc.
• Daniel Gruber - Univa GmbH
• Mariusz Mamonski - Poznan Supercomputing and Networking Center
• Daniel Templeton - Cloudera Inc.
• ... plus ...
• Andre Merzky - SAGA Working Group
• Thijs Metsch - Platform Computing / OCCI Working Group