

GGF-DRMAA

John Tollefsrud [j.t@sun.com], Sun Microsystems
Hrabri Rajic [hrabri.rajic@intel.com], Intel

GGF6
Chicago, IL
October, 2002



DRMAA WG at GGF6

Presentation Session

10/15/02

12:00-13:30

- Presentation
- C22 attribute precedence rules
- C26 DRMAA_PS_QUEUED_ON_HOLD

Working Session

10/16/02

08:00-9:30

- 20 mins N1 DRMAA specific info tunneling
- 20 mins C18 Validity of job ids across different sessions
- 20 mins C17 Interface names and global issues

----- 30 mins time permitting -----

- C11 inability to wait for results of control actions
- C12 timeout for drmaa_wait routine
- C13 “reaping” job ids
- C14 timeout for drmaa_synchronize routine

Working Session

10/16/02

10:00-11:30

- C16 error handling

- `stdio/out/err`



First things first

- **DRMAA scope and purpose:**
 - Submit, control & monitor, and query status of jobs.
 - DRMAA library could be implemented on top on OGSA and DRM systems.
- **Need two volunteers for taking this session minutes**
- **Sign-up sheet**
- **DRMAA needs a secretary**



Past DRMAA Activity

- Weekly con calls
 - Toll Free: (877)288-4427 Code: 691169 (Please email to j.t@sun.com)
- E-mail: drmaa-wg@gridforum.org
- Archive: http://www-unix.gridforum.org/mail_archive/drmaa-wg/threads.html

- **Two sessions at GGF5**
- **Working document DRMAA-1.9**



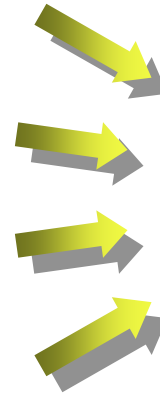
Why DRMAA?

- Adoption of distributed computing solutions in industry is both widespread and 'early adopter'
 - **Commercial applications by independent software vendors (ISVs)**
 - **Commercial distributed resource management (DRM) systems**
 - **Scripted command-line integration by end users**
 - **Very little direct interfacing of ISV apps to DRM systems**
- Adoption is self-limiting to industries where gain exceeds the pain
- Fundamental shift in the adoption pattern requires shifting the DRM integration to the ISV



Distributed Resource Management (DRM) Systems

- Batch/job management systems
- Local Job schedulers
- Queuing systems
- Workload management systems



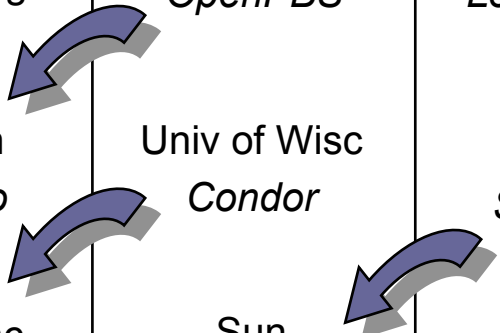
All are DRM Systems



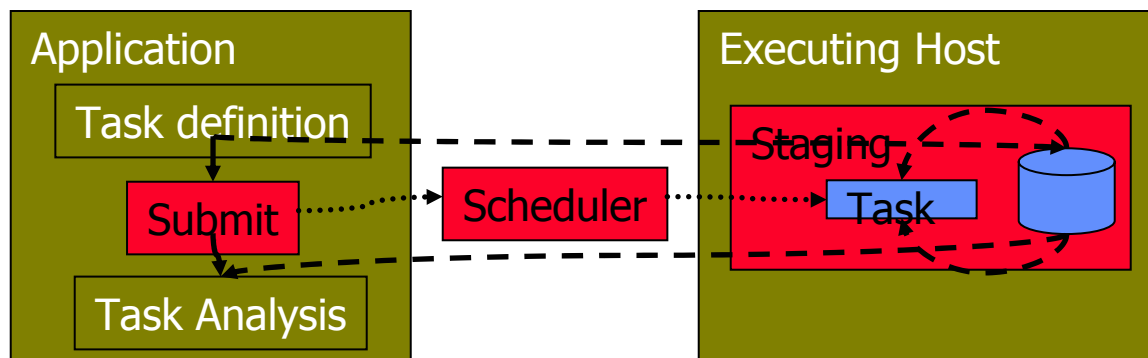
Motivation for DRMAA

There are many DRM solutions available to end users and things keep changing

Independent Suppliers	Open Source / University	OEM Proprietary	Peer-to-Peer
Platform Computing <i>LSF</i>	Veridian <i>OpenPBS</i>	IBM <i>LoadLeveler</i>	TurboLinux <i>Enfuzion</i>
Veridian <i>PBS Pro</i>	Univ of Wisc <i>Condor</i>	Sun <i>Sun Grid Engine</i>	Entropia United Devices Parabon
Condor Inc. <i>Condor</i>	Sun <i>Grid Engine</i>		



Resource Management Systems **Differ**



- **Core services are fundamentally the same**
 - especially from the users perspective
- **DRM programming interfaces differ**
 - ISVs are disinclined to use



DRMAA Charter

- Develop an API specification for the submission and control of jobs to one or more Distributed Resource Management (DRM) systems.
- The scope of this specification is all the high level functionality which is necessary for an application to consign a job to a DRM system including common operations on jobs like termination or suspension.
- The objective is to facilitate the direct interfacing of applications to today's DRM systems by application's builders, portal builders, and Independent Software Vendors (ISVs).



Characterizing DRMAA

- High level attributes
 - Application centric
 - Ease of use for end users
 - Focused on programming model
- Benefits
 - Faster distributed application deployment
 - Opportunity for new applications
 - Increased end user confidence
 - Improvements in Resource Management Systems
 - Distributed application portability



Scope: Run a Job API

(Steps from: "Ten Actions when SuperScheduling", GGF SchedWD 8.5, J.M. Schopf, July 2001)

- Phase 1: Resource Discovery
 - Step 1 Authorization Filtering
 - Step 2 Application requirement definition
 - Step 3 Minimal requirement filtering
- Phase 2 System Selection
 - Step 4 Gathering information (query)
 - Step 5 Select the system(s) to run on
- Phase 3 Run job
 - Step 6 (optional) Make an advance reservation
 - **Step 7 Submit job to resources**
 - Step 8 Preparation Tasks
 - **Step 9 Monitor progress (maybe go back to 4)**
 - **Step 10 Find out Job is done**
 - Step 11 Completion tasks



DRMAA Guidelines

- It should lead to straightforward programming model.
- The API calling sequences should be simple and the API set small.
- The routine names should convey the semantic of the routine.
- Avoid duplicated functionality, i.e. interface overloading.
- All jobs manipulation per process is available without explicit job iterating.
- The servers names are hidden, the DRMS is a black box.
- Consistent API structure
 - Err return parameter, internal errors via global errno parameter
- Data structures not exposed



What have been the Issues?

- Language bindings

- C/C++
- Perl, Python
- Fortran, Java

- General features

- DRMAA sessions
- Asynchronous job monitoring
- Protocol based
- Scalability
- Wide characters

- Libraries

- Serial / thread safe
- Tracing / diagnosis

- Advanced features

- Debugging support
- Data streaming
- Security
- Categories



Implementation characteristics

- **C-API library interface - no protocol**
 - Simplifies utilization by ISV's
- **Shared library binding**
 - Prerequisite to allow end user to select DRM technology of their choice
- **Library supports only one DRM system per implementation**
 - Simultaneous support of different DRM systems is beyond the scope of our project



API groups

- **Init/exit**
- **Job template interfaces**
 - Allocate/delete
 - Setter/getter job template routines
- **Job submit**
 - Individual jobs
 - One time
 - Multiple times – templates 9 (version 2)
 - Bulk jobs, implicit parameterization
- **Job monitoring and control**
- **Auxiliary or system routines**
 - trace file specification
 - error message routines
 - informational interfaces



Job Template

- **Functions to create/delete job template**

- `job_template *drmaa_allocate_job_template (void)`
- `void drmaa_delete_job_template (job_template *jt)`

- **Setter/getter job template routines**

- `int drmaa_set_attribute(job_template *jt, char *name, char *value);`
- `int drmaa_set_vector_attribute(job_template *jt, char *name, char **values);`
- `char* drmaa_get_attribute(job_template *jt, char *name);`
- `char** drmaa_get_vector_attribute(job_template *jt, char *name);`



Job Submission

- **Jobs submitted to the DRM system are identified via a job identifier**
- **For flexibility reasons a job identifier should be of type char ***
- **Single job identifiers are returned by**
 - `int drmaa_run_job(job_template *jt, char *job_id)`
- **Bulk job submissions return multiple job identifiers**
 - `int drmaa_run_bulk_job(char **job_ids, job_template *jt, int start, int end, int incr)`



Native DRMS Options

- The end user interacts with the DRMS via `native_resource_options` parameter.
 - Simple solution
 - DRMAA implementation ignores the DRMAA DRMS implicitly used and disallowed options
 - Dist. Appls. Developers and DRMS vendors are not involved in the local environment spec.
- The burden is on the end users to define the execution environment
 - Need to know DRM
 - Need to know the remote application installation



Job Monitoring, Control, and Status

- **Monitoring/Control functions**

- `int drmaa_control(char *job_id, int action);`
- `int drmaa_synchronize(char **job_ids);`
- `int drmaa_job_ps(char *job_id, int *remote_ps);`

- **Blocking and non-blocking waiting for one or more jobs to finish (like wait4(2))**

- `char *drmaa_wait(char *jobid, int *status, int timeout, char **rusage);`
- Use Posix functions `drmaa_wifexited`, etc. to get more information about failed jobs.



Auxiliary Routines (proposal stage)

- **Error/logging interfaces**

- `int drmaa_set_trace_file(char *file_name);`
- `int drmaa_trace_text(char *text);`
- `int drmaa_perror(char *text);`
- `char *drmaa_strerror (int error);`

- **Informational interfaces**

- `int drmaa_version(int *major, int *minor);`
- `char *drmaa_get_DRM_engine();`
- `char *contact drmaa_get_contact();`



Backup slides

- **Additional details**



Execution environment

- **File system duality**

- Shared
 - Appls. comes to data
- Distributed (not part of DRMAA spec)
 - Data come to appls.
 - Unique dir per job
 - Copying files?

- **Environment passing**

- Use default behavior
- Export local env.
- Use remote env.
- Env specified via API

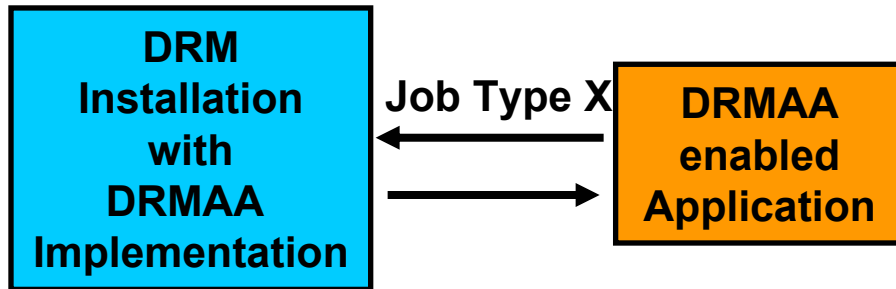
- **Handling of DRMS job execution options**

- Translation/consolidation
- Allowing native



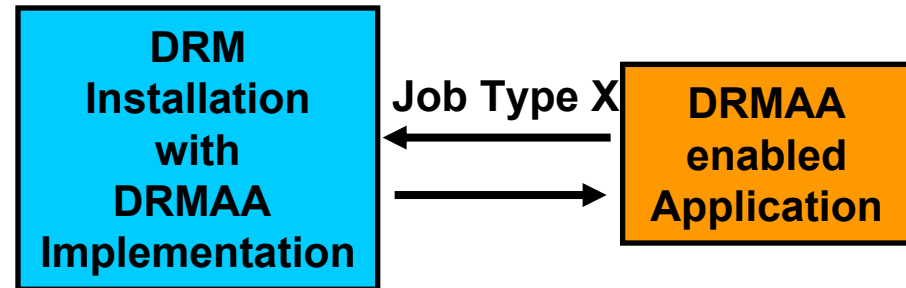
Job Categories (version 2.0 feature)

Site A



- Cluster consists of machines where X jobs run and others where they don't run

Site B



- X jobs run at all machines in cluster

