DRMAA Working Group

Cochairs

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

GGF7

Tokyo, Japan

March, 2003

Meeting Agenda

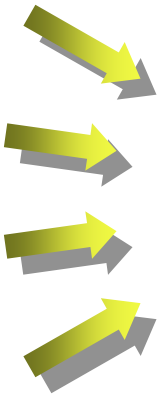
- **Review GGF Meeting guidelines**
  - Intellectual Property rules
  - Attendance sheet
  - Meeting minutes
- **Introduction to DRMAA**
  - Why DRMAA?
  - The DRMAA-WG and DRMAA charter
  - DRMAA scope and purpose
- **The DRMAA specification 1.0 proposal**

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



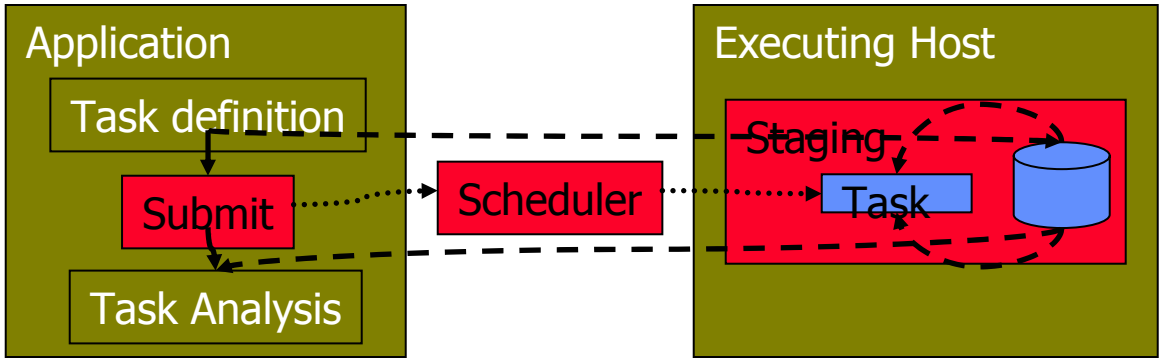
All are DRM Systems

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 *LSF*	Veridian *OpenPBS*	IBM *LoadLeveler*	TurboLinux *Enfuzion*
Veridian *PBS Pro*	Univ of Wisc *Condor*	Sun *Sun Grid Engine*	Entropia United Devices Parabon
Condor Inc. *Condor*	Sun *Grid Engine*		

Resource Management Systems Differ



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

DRMAA Working Group

- **DRMAA discussed at DRMAA BOF, GGF3**
  - Two first-draft proposals
    - **Hrabri Rajic, Intel**
    - **Andreas Haas, Fritz Ferstl, Sun Microsystems**
  - Comments received
  - WG status requested
  - WG status granted by GGF Steering Committee
- **Much committee work to integrate the proposals!**
  - Multiple working sessions at GGF4, GGF5, GGF6 and GGF7
  - Biweekly and weekly con calls
  - Long hours and much effort from many people – many thanks!
- **Thanks to Cadence, HP, IBM, Intel, Platform, NASA, Robarts, Sun, Veridian Systems, and more**

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 API Guidelines I

- 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.
- Data structures are not exposed.

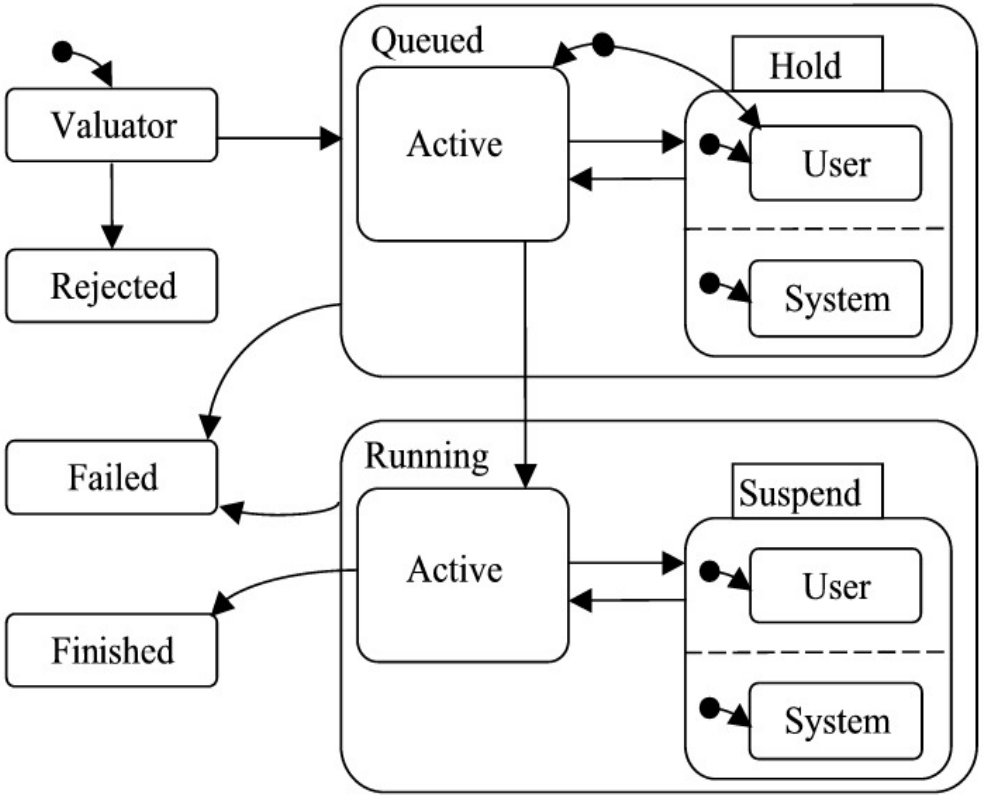
DRMAA API Guidelines II

- **Consistent API structure**
  - Err return parameter, internal errors pass in buffer
- **Job template**
  - Required attributes
  - Optional Attributes
- **Job categories**
- **Native specification**
- **No explicit file staging**

Implementation requirements

- **C-API library interface - no protocol**
  - Simplifies utilization by ISV
  - Not transactional
  - Object oriented wrappers/classes specification in the works
- **Shared library binding**
  - Prerequisite to allow end user to select DRM technology of their choice
- **One session at the time**
- **Library supports only one DRM system per implementation**
  - Simultaneous support of different DRM systems is beyond the scope of our project

DRMAA State Diagram



The remote job could be in following states:

- **system hold**
- **user hold**
- **system and user hold simultaneously**
- **queued active**
- **system suspended**
- **user suspended**
- **system and user suspended simultaneously**
- **running**
- **finished (un)successfully**

API groups

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

Init/Exit

- drmaa_init(contact, drmaa_context_error_buf)
  - must be called before any other DRMAA calls
  - Opens a session
    - No session nesting
    - One session job ids could be used in other session if resources shared
  - No session nesting
  - Contact could be NULL, default connection

- drmaa_exit(drmaa_context_error_buf)
  - **Last call to disengage DRMS**

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);
- **Informational interfaces**
  - drmaa_get_attribute_names(char* names)
  - drmaa_get_vector_attribute_names(char* vector names)

DRMAA Job Attributes

Mandatory job attributes:

- Remote command to execute
- Remote command input parameters, a vector parameter
- Job state at submission
- Job environment, a vector parameter
- Job working directory
- Job category
- Native specification
- Standard input, output, and error streams
- E-mail distribution list to report the job completion and status, a vector parameter
- E-mail suppression
- Job start time
- Job name to be used for the job submission

Optional job attributes:

- transfer files
- absolute job termination time
- wall clock time limit
- soft wall clock time limit
- job run duration hlimit
- job run duration slimit

Job Submission

- **Jobs submitted to the DRM system are identified via a job identifier**
- **For flexibility reasons a job identifier should be a string**

- **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)

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 like functions drmaa_wifexited, etc. to get more information about failed jobs.

Auxiliary Routines

- **Error/logging interfaces**
  - error drmaa_strerror (int error);

- **Informational interfaces**
  - int drmaa_version(int *major, int *minor);
  - DRM_system drmaa_get_DRM_system( );
  - DRM_contact contact drmaa_get_contact( );

Site specific requirements

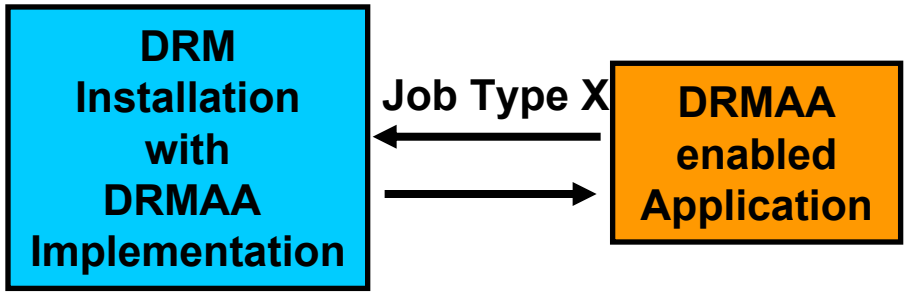
- **Application Developers and DRMS vendors are not involved in the local environment specification**
  - Execution policies
  - Physical environment

- **Two hierarchical mechanisms**
  - Job categories
    - Vendor determines the name and application parameter guidelines
    - Administrators, installation people

  - Native specification
    - Opaque string that DRMAA impl. resolves
    - The burden is on the end users to define the execution environment
      - Need to know DRM
      - Need to know the remote application installation

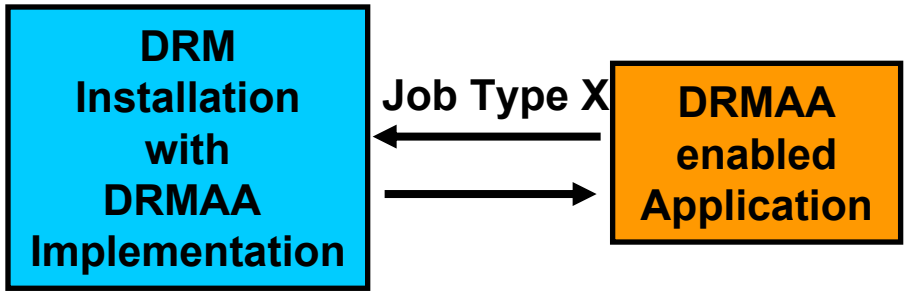
Job Categories

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

What Are the Issues?

- Language bindings
  - C/C++
  - Perl, Python
  - Fortran, Java
- General features
  - DRMAA sessions
  - Asynchronous jobs
  - Protocol
  - Transactional interface
  - Scalability
  - Internationalization
- Libraries
  - Serial / thread safe
  - Tracing / diagnosis
- Advanced features
  - Debugging support
  - Data streaming
  - Security
  - Local env interaction

Conclusion

- **DRMAA v1 is submitted to GFSC for review**
- **Integration with other SA/GGF groups**
- **v2 work**
  - Within DRMAA WG
  - Together with other WG/RGs, new WG
  - Both of the above approaches

Backup slides

- **Additional details**