The gLite Workload Management System

Marco Cecchi (INFN-CNAF)
gLite WMS team
gLite

- Develop a **lightweight stack of generic middleware** useful to a variety of applications (mainly HEP, but also Biomedics, Earth Sciences, AstroPhysics, Fusion...)
  - Pluggable components – cater for different implementations
  - Follow SOA approach, WS-I compliant where possible

- Build on **experience and existing components** from VDT (Condor, Globus), EDG/LCG, AliEn, and others

- Focus is on **re-engineering and hardening**

- Business friendly **open source license**
• The Workload Management System (WMS) comprises a set of Grid middleware components responsible for the distribution and management of tasks across Grid resources, in particular Computing Elements (CE), in such a way that applications are conveniently, efficiently and effectively executed.

• Multiple processes
• Reliable communication, with persistency where needed
• Compliance to formal and de-facto standards (JSDL, WS-I)
• Actions are done on behalf of the user, i.e. with delegated credentials
Supported Job Types

- Batch-like
- DAG workflow
- Collection
- Parametric
- MPI
- Interactive
Job Description Language

• Job Description Language (JDL)
  – gLite approach to Request Description
  – ClassAds-based language (key/value pairs)
  – Fully extensible & flexible high-level

• **Allow the user to specify job execution needed information**
  – Characteristics of the application
    (Executable, Arguments, Input/Output Sandbox files, ...)
  – Requirements/preferences about resources
    (Computational, storage)
  – Management hints for the WMS (number of retries, proxy renewal, ...)

• Investigating Job Submission Description Language (JSDL)
Key features

• Mechanisms for error prevention and recovery
  – Persistent data structures
  – Load limiting
  – Resubmission of failed jobs in various forms
    ▪ A job is shallow resubmitted if failed before having started the execution on the WN. This improves the job success rates preventing multiple instances of the same job over the Grid.
    ▪ Deep resubmission as opposed to shallow, occurs in the other case.

• Fuzzy ranking – smooth distribution of the best resource selection

• Support for MPI jobs even without a shared fs between CE and the WN

• Gang-matching – including SEs in the MM
  – Send jobs only where the data are
• Faster authentication via explicit delegation
  – Automatic delegation only when submitting a single job

• Proxy renewal (including VOMS AC)

• Interoperation with different resource Information Providers
  – BDII (synch), CeMon (synch, asynch), R-GMA (synch)

• Job Wrapper
  – Shell script wrapping the user's job execution, providing support for sandbox management, logging, environment etc.
  – Generic customization hooks available for users, VOs and site admins
  – Interoperability with OSG
• **Job Sandbox**
  – It's a reduced amount of relatively small files (conf, log, I/O) accompanying the job
  – Automatic compression
  – Different jobs can share the same sandbox,
    ▪ reduce network traffic / save time and bandwidth

• **Sandbox Remote Specification**
  – User can store files directly on a remote machine
  – No intermediate copies – WN will directly download
  – Reduced server load

• **Supported File Transfer**
  – Full support (uploading/downloading) for protocols:
    ▪ gridftp, https
• **Service Discovery**
  – Provide additional information by performing queries to external databases of different kinds (RGMA, BDII)
    ▪ Client side
      • *Queries for available WMS endpoints on the Grid*
      • *Do not need manual reconfiguration*
    ▪ Server side
      • *Queries for available LB servers where to Log Job information*

• **Job Files Perusal**
  – Perform a monitoring activity on the actual output files produced by a job while running
  – Add useful information not available by simple status monitoring, once available only at job completion
Key features /5

• **WMS Job submission is done through:**
  
  – **Condor-G**: supports submission to:
    
    ▪ LCG (GT2 GRAM)
    ▪ gLite (GT2 GRAM + Condor-C)
    ▪ ...

  – **ICE**: supports submission to:
    
    ▪ CREAM (WS-I, OGSA/BES)
    ▪ Asynchronously receive notifications from CEMon

• **Bulk submission and bulk match-making**
Bulk submission: possibility to submit a bunch of jobs in one single interaction with the WMS
- (possibly) heterogeneous → collection
- Homogeneous → parametric
- Reduced submission time, managed by a single id

Bulk MM: to match “equivalent” jobs in one shot, i.e. with one single mm operation
- Natural completion of bulk submission
- Two jobs are equivalent if their significant attributes are literally the same
- The significant attributes are specified by the user
  - Typically Requirements, Rank, FuzzyRank, ...
Overall System

- System is complex
  - provide complex functionalities
  - support legacy components
Data structures for MM

- **Information Super-Market**
  - A repository of information about resources
  - Allow decoupling from information and its use
  - Updated by
    - Incoming notifications
    - Active polling of Information Providers
  - Support for “lazy” scheduling policies

- **Task Queue**
  - Hold submission requests when no resource is available
  - Pending requests
    - Either retried periodically until expiration (“eager” approach)
    - Or waiting to be called for a match by an incoming notification of available resource (“lazy” approach)
• **Web Service Interface**
  – Replaced the legacy proprietary network interface
  – WS-I compliant
  – Implemented as a FastCGI gSOAP application spawned by an Apache http server
  – Strong authentication

• **GridFTP, GridSite**
  – Secure file transfer for uploading/downloading the sandbox (gsiftp, https)
• New platforms and architectures are being addressed on the infrastructure
  – In particular Scientific Linux 4 and 64-bit architectures
  – Made easier by the migration to ETICS
    ▪ Sw configuration and build system
  – Ongoing activity: Integration & restructuring
    ▪ Code clean-up
    ▪ Removing/Reducing Dependencies on external software
• Bulk submission & MM were in the initial implementation transformed into a DAG and then managed with Condor DAGMan
  – Correct but overkill solution when nodes do not actually have dependencies
  – Major source of instability and complexity of the system
  – Some hacks needed to keep resource usage under control, i.e. global limit on the number of planners
  – Now direct management, much smoother behavior

• Improved memory management

• Load limiter
  – prevents submission if the WMS is overloaded
  – round-robin of WMSs on the UI: in case of overload the client can go to another instance of the service
• Intense testing and bug fixing over the last few months
  – Improved stability
  – Improved job submission rate

• Introduced the *Experimental Services*
  – Instances of the services attached to the production infrastructure
  – Scalability testing prior to release
  – Maintained by SA1 and SA3
  – JRA1 patches are installed immediately (before the certification)
  – Testing done by selected application users
  – Process controlled by the EMT
Acceptance criteria

- A single WMS/LB instance should demonstrate submission rates of at least 10 Kjobs/day sustained over 5 days, without the need to be restarted
- The number of stale jobs after 5 days must be < 0.5%

Acceptance test results (Easter ’07)

- 16K jobs/day (~11 jobs/min) over one week of submissions
  - No manual intervention on servers (WMS & LB)
  - Stable memory usage
  - 0.3% of jobs in non-final states
  - Aborted jobs mostly due to expired user credentials
Stress-testing bulk-submission:
\[ \sim 27 \text{kjobs/day} = \sim 18 \text{ jobs/minute} \]
• Another test, job-submission of single-jobs (as compared to compounds):
  – Use-case for the submission of a limited number of jobs from a huge number of different users
  – Also as a stress test for debug purposes
  – To study how submission & MM time do actually scale.
  – MM on the production BDII takes about 4 secs.
• >15000 jobs/day sustained over 11 days
  – Reaching peaks of some 22k jobs/day of throughput
    ▪ Disabling some secondary service (ISM dump, log levels)
    ▪ Disabling the load limiter
Conclusions

- Provide services on top of job submission
- Facing new larger scales
  - to satisfy applications use cases
- Striving to further improve reliability
  - Error recovery
  - High-availability
  - Fault-tolerance / Robustness
- Development continues
  - Reducing internal/external dependencies
  - Adding new features
- Stronger integration, scalability and interoperability with emerging standards
  - Further improvements (functionality and scale) using ICE
Number of jobs in each status vs. time

- Cancelled
- Cleared
- Aborted
- Done (Failed)
- Done (Exit)
- Done (Success)
- Running
- Scheduled
- Ready
- Waiting
- Submitted

Jobs vs. time (hours)
Some relevant job attributes:

- **JobType**: Several types supported (see later on)
- **Executable** (mandatory): the command name
- **Arguments** (optional): job command line arguments
- **StdInput**, **StdOutput**, **StdError** (optional): standard input/output/error of the job
- **Environment**: list of environment variables to be set on the Worker Node env
- **InputSandbox** (optional): list of files on the UI local disk needed by the job for running
- **OutputSandbox** (optional): list of files, generated by the job, which have to be retrieved

```plaintext
[  Executable = "my_exe";
  StdOutput = "out";
  Arguments = "a b c";
  InputSandbox = {"/home/user1/my_exe"};
  OutputSandbox = {"out"};
  Requirements = other.LRMSType=="Condor" && 
                  other.Architecture=="INTEL" && 
                  other.OpSys=="LINUX" && 
                  other.FreeCpus >=4;

  Rank = -other.GlueCEStateEstimatedResponseTime;
  RetryCount = 2
]
```

- **Job Attributes**
- **Resources attributes**
  used to build expressions of
  Requirements and/or Rank attributes
  by the user (have to be prefixed with "other.")

other.Architecture=="INTEL"

Rank = -other.ResponseTime
from “EGEE Middleware Architecture”, EU deliverable DJRA1.1, August 2004
https://edms.cern.ch/document/476451/
• Possible integration with external existing Workflow managers
  – Triana, GWES, Taverna, etc
  – Still to be discussed and planned for EGEE III

• Moreover, Workflow Mangement System (WfMS) Architecture Proposal for WMS
  – Running on top of gLite Middleware
  – Grid Middleware Undependent
  – Abstract and Generic Representation
  – Translation mechanisms from different language front ends
  – Will be exposed/discussed at next CoreGrid forum
Interactive Jobs

- Job's standard streams forwarded to the submitting client
- Opens X window, $DISPLAY$ must be set
• Runs in parallel on several processors
• Supports MPICH
• Submission to single CEs
- Parallel jobs = MPI jobs: MPICH implementation supported.

The submission of parallel jobs is very easy to specify:

One just needs to specify in the JDL:

- JobType = "MPICH"
- NodeNumber = n;

the number of requested CPUs.

```plaintext
[ Type = "job";
  JobType = "mpich";
  VirtualOrganisation = "iteam";
  // This is the minimum number of CPU needed by the job
  NodeNumber = 6;
  Executable = "cpi";
  StdOutput = "sim.out";
  StdError = "sim.err";
  OutputSandbox = {
    "sim.err",
    "sim.out"
  };
  // This attribute triggers the proxy-renewal mechanism
  MyProxyServer = "skurut.cesnet.cz";
  RetryCount = 3;
  InputSandbox = {
    "/home/fpacini/JDL2/fox/cpi"
  };
  requirements = other.GlueHostNetworkAdapterOutboundIP
  &&
  Member("IDL2.1",other.GlueHostApplicationSoftwareRunTime
  Environment);
  rank = other.GlueCEStateFreeCPUs;
]
```

- Matchmaking

- CE chosen by WMS has to have MPICH sw installed, and at least \( n \) total CPUs

- If there are two or more CEs satisfying all the requirements, the one with the highest number of free CPUs is chosen
Automatic upload and registration of datasets produced by the job

OutputData = {
    [
        OutputFile = "filename1";
        LogicalFileName = "lfn:mylfn1";
        StorageElement = "testbed007.cnaf.infn.it"
    ],
    [
        OutputFile = "filename2";
        LogicalFileName = "lfn:mylfn2"
    ]
}
With “standard” matchmaking only 2 “involved entities” the job and the CE

Gangmatching allows to take into account, besides CE information, also SE information in the matchmaking process

Typical use case for gangmatching:

- My job has to run on a CE close to a SE with at least 200 MB of available space:

  Requirements = anyMatch(other.storage.CloseSEs, target.GlueSAStateAvailableSpace > 200);