Enabling Grids for E-sciencE

Information Society and Media

EGEE

SHORT DEADLINE JOBS WORKING GROUP

FIRST REPORT

Document identifier:	**SDJ-WG-TEC-v1.0.doc**
Date:	**30/03/2006**
Activity:	**SDJ WG**
Document status:	**DRAFT**
Document link:	

Abstract:

Document Log

Issue	Date	Comment	Author/Partner
0-0		First draft	Cécile Germain-Renaud and the SDJ WG

Document Change Record

Issue	Item	Reason for Change

CONTENT

1. INTRODUCTION

1.1. PURPOSE

The general objectives of the working group is to address the problem of short jobs. We consider jobs of a few minutes computation times, where the QoS requirement is mainly latency. The issues are: to provide QOS guarantees for SJ, while the current scheduling allows only for best effort; and to limit the grid submission and scheduling overhead. The mandate of this group is to analyze the application needs for short jobs, to explore the tradeoffs between application requirements and minimal evolution of existing software and services, and finally to propose corresponding technical solutions.

The report is organized as follows:

- An executive summary (section 2)
- Recommended actions (section 3)
- The following sections contain details about
  - The definition of short jobs (we will see that the definition is not so obvious).
  - A portofolio of applications which require or may profit from SDJ, together with the detail of their functional requirements and current state of development.
  - Some analysis of the current bottlenecks in gLite software which motivate the need for specific actions or services targeting SDJ.

1.2. APPLICATION AREA

TCG

1.3. REFERENCES

[R1]	
[R2]	

1.4. DOCUMENT EVOLUTION PROCEDURE

1.5. TERMINOLOGY

Glossary

Definitions

2. SUMMARY

2.1. DEFINITION

A Short Deadline Job (SDJ) is:

- A job with a deadline constraint; which provides some guarantees about its behavior; which is unable to proceed through prior explicit reservation. Because they have a short execution time and because they are unexpected and urgent, they cannot be dealt with only on a best effort basis in full production regime.

- A plain EGEE job in the following sense: it is submitted, scheduled, and returned to the user through the standard mechanisms governing the usage of the resources: UI, Broker, CEs,…. In particular, it can be inspected by the usual tools (WMS trace), and is fully accounted for.

2.2. USE CASES

All are more or less related to related to some human *expectation* about interactive turnaround time.

- Visualization: visualize results, e.g. through a Web interface
- Inspection: for test jobs, where the user will inspect the job state, its related files etc.
- Interaction: augmented or virtual reality, computational steering,…

For CMS, the major use case is the test/debug phase preceding production. More generally, physics analysis has been presented as a possible user of SDJ for interactive analysis, especially in the framework of DIANE. Many biomedical applications (GPS@, CDSS, gPTM3D) consider SDJ as a **critical requirement**. This is also true for DILIGENT. The workload of some applications (CDSS, gPTM3D) is only composed of SDJ, while for other one (GPS@, DILIGENT), the workload also includes long-running jobs.

2.3. REQUIREMENTS

In order to get the required QoS, the following requirements have been identified.

- Scheduling policy: immediate access to the processors, which means no queuing.
- Middleware implementation: smaller latencies in the various stages of the job lifecycle.
- High-level services:
  - A generic user-level (agent) scheduling service.
- Low-level Services, or software components
  - Connectivity between the UI and WNs
  - Dynamically attach a shell to a running process (especially for debugging purposes)
  - Interactive inspection of the state of the queues

For the current application panel, there was no need for inter-site connectivity *as far as SDJ are concerned*.

Explicit advance reservation is not compatible with the users' practice.

2.4. SCHEDULING POLICY

The SDJ queue developed at LAL/LRI avoids queuing for short jobs: SDJ jobs run concurrently with batch jobs. This solution is relatively independent of the submission system. The fraction of the resource usage devoted to SDJ is locally configurable. The current status is as follows:

- A Torque/MAUI configuration has been successfully tested at LAL. No proposal has been made to port it to other batch systems. Testing the configuration on more sites is ongoing.

- Some minor modifications of the WMS are required. JRA1 will provide them for gLite3.2

- In the long run, a modification of the Glue Schema is desirable.

- The process for deployment (on sites which are not directly participating in the WG) is an open issue.

2.5. MIDDLEWARE DELAYS & AGENT SCHEDULING

The overhead of LCG and the current version of gLite are recognized by all users participating in this WG as a major issue, which is often considered as preventing large-scale use of the EGEE-enabled applications that they have developed.

- From 'submission' to 'scheduled' states: gLite 3.0 could provide significant improvements, but not enough performance results are currently available to know its impact on our applications. For reasons detailed in the report, bulk submission is not adequate for these applications.

- From job completion (LRMS) to notification (WMS): this is a very serious problem, first because gLite3.0 will not improve on this point, and second it has some impact on the SDJ queues system.

An alternative solution is "agent scheduling" or "overlay" systems, which provides user-level task management. Many such systems have been developed; they are all fully "EGEE-compliant" (do not disturb load balancing, full accountability); some are presented in this report. Nevertheless, the vast majority of users claim that they prefer regular scheduling.

The discussion on this point has not gone into sufficient detail. The WG proposes to continue it; **information about the ongoing work on overlay system at INFN-Bologna is urgently needed**.

2.6. CONNECTIVITY

Two kind of requirements have been expressed in this area

- Interactive connection to a running process for debugging and inspection. Systematic code instrumentation for reporting through R-GMA is not convenient because the problem requiring debugging is not likely to have been anticipated. In some cases, job submission as an interactive job can help (the performance is announced to be much better now), but may be a limitation (e.g. for jobs that already redirect input/output).

- Port forwarding, in order to tunnel efficiently any protocol trough the firewalls blocking worker nodes.

Tools are (or will be) available for dynamically attaching a remote (limited) shell. Thus job inspection should not be a real issue.

On the other hand, these tools are not convenient for data streaming, because they target shell commands, not plain data streams. The glogin package may offer a comprehensive solution if it could be integrated with the WMS.

The following actions have been identified to facilitate the use of the EGEE infrastructure by those applications needing SDJ support. The probable actors needed to implement the requirement are indicated.

- (JRA1) Add a Boolean attribute in the JDL which flags short deadline jobs. Based on the value/presence of this flag, the job wrapper requirements will be modified to select SDJ CE and will prevent non-SDJ jobs from being scheduled on SDJ computing elements. For

now, SDJ CE will be identified by their name ".*sdj$". These two modifications should be developed and tested by JRA1.

- (JRA1) For a clean implementation, a modification of the information model would be required. A new CE attribute (per queue not per cluster) should be created which identifies SDJ CEs. This would allow system administrators to identify SDJ queue regardless of the name of the queue.
-
- (JRA1, glogin developers) The current state of glogin (without full integration with WMS) should become a standard part of the release a soon as possible. Rationale: ease user experiments for their needs about stream-oriented connection and remote shell facilities.
-
- (JRA1) Dissemination of performance tests. A large number of performance tests are currently performed by various teams, on the different services and components, including the CREAM CE. These are of particular interest to the SDJ users, and a faster and more systematic diffusion of the results of these tests would be very useful.
-
- (SA1, SA3) Support for the deployment of the SDJ CE. The SDJ configuration should be available as a standard part of the release. Besides user jobs, the SDJ CE is convenient for the dteam jobs.
-
- (SDJ) Interaction with the software development "overlay system" (in relation with the agent scheduling issue, see section 8)
- (SDJ) Interaction with the TCG WG on Priorities (see section5.1.2)
- (SDJ) Define a common policy about the SDJ CE limits.

3. DEFINITION

3.1. SJ AND SDJ

The most obvious definition of a Short Jobs would be jobs that have a short execution time (at the grid scale), typically a few minutes. However, this definition fails to capture the functional aspects and usage contexts of such jobs. A broader definition is related to Quality of Service. A Short Deadline Job (SDJ) is a job

- with a deadline constraint;
- which provides some guarantees about its behavior;
- which is unable to proceed through prior explicit reservation.

However, SDJs retain something of the "shortness" in the simpler definition; they are either too short in absolute value, or too unexpected and urgent, to be dealt with only best effort in full production regime.

The deadline is *soft*, in two senses:

- Technical: the constraint allows for some slackness, e.g. a 2min deadline could be extended to 3min, but no more.
- Practical: if the deadline is violated, the user will be irate, but nobody dies….

The second point may lead to think that this is no more than best effort, but this is not true. The line of demarcation could be as follows:

- If there is no fault in the path of the SDJ, then (subject to admission control) the deadline is met.
- However, if faults are considered (and they must be…), nothing can be guaranteed in the most general case, thus the second point. This by no means imply that the system should not be able to deal with faults (e.g. through timeouts/resubmission), but the constraint could be violated in this case.

This definition allows for situations ranging from light interactive tasks (e.g. gPTM3D) to dealing with semi-emergency situation where a large fraction of the resources should be dedicated to a particular set of jobs. If absolute QoS of service is required (e.g. intra-operative situations), then advance reservation is the only solution.

Note that defining SDJ by a deadline provides a basis for dealing with jobs that require a constant instantaneous computational bandwidth (by scaling the theory and practice of real-time scheduling at the grid level).

3.2. SDJ AS EGEE JOBS

The SDJ must be EGEE jobs in the following sense: they are submitted, scheduled, and returned to the user through the standard mechanisms governing the usage of the resources: UI, Broker, CEs,… In particular, they can be inspected by the usual tools (WMS trace), and are fully accounted for. This is for instance different from the architecture used in DIRAC or PROOF.

Each of these jobs may be composed of microtasks. However, this is internal to the application, and the microtasks are not visible to the EGEE middleware.

4. SDJ-RELATED APPLICATIONS

SDJ applications: they are more or less related to related to some human *expectation* about turnaround time in the interactive scale:

- Weak sense: visualize results, e.g. through a Web interface; or for test jobs, where the user will inspect the job state, its related files etc.
- Strong sense: augmented or virtual reality, computational steering,….

4.1. PHYSICS ANALYSIS

Physics analysis should be carried on the same grid as production, because the datasets are the same, but asks for different timescales and human-computer interaction. While Distributed Analysis issues and software are much complex than only fast turnaround time, this feature is an important ingredient. An extended set of projects and software has already been developed inside HEP experiments.

4.1.1. CMS

The CMS use case is basically testing and debugging an application so that it can be sent in 1000 instances without all of them crashing at the first line. Usually 5min CPU is plenty, and turnaround of less than 5min required (assuming less than 1 min CPU) to avoid frustration. The number of such jobs is limited, typically one user would submit a single one, look at log, fix bug, do it again, all day long, but not with N jobs submitted at same time. Thus bulk submission is no use for these jobs (but is of course useful for production).

A SDJ CE with e.g. maxjobs=10 and MaxCPUTime=5min would probably do just what we need. So most important for this application will be a fast Resource Broker. Another issue is the case of real applications that last 1hour instead of 48 hours, and thus should get some QoS, but there it is a matter of shares and priorities, not WMS turnaround.

It must be noticed that this requirement is much weaker than those of the SDJ CE currently considered for gLite3.2.

With respect to current implementation, an SDJ configuration for LSF would be needed.

4.1.2. Physics Analysis in general

From the ATLAS requirement web page

(https://uimon.cern.ch/twiki/bin/view/Atlas/ShortQueuesForDistributedAnalysis)

associated to the TCG WG Task Priorities, it appears that there is significant intersection between our concerns.

“In an optimal scenario CPUs will be kept busy with Production jobs until Analysis jobs arrive. The classical mechanism to achieve this goal is preemption. In our case this scenario seems to be excluded, as other required resources - connection to Storage Elements and databases - currently do not support such a model. So analysis jobs have to be run in parallel to simulation jobs.

Taking into account in particular the memory requirements for ATLAS production jobs (which is around 600 MB), it seems possible to fit additional analysis jobs on machines with sufficient memory. Assuming a minimum configuration of a dual processor PC with 2 GB of memory, additional one or two *virtual slots* could be defined for analysis.

Due to our requirements these *virtual slots* would be connected to queues limited in CPU (1 hour) and memory (500MB).

Deployment of such a configuration might require some learning phase. In case ATLAS production will start early to use all available resources, it might nevertheless be necessary to reserve some small number of slots exclusively for analysis in an initial phase.

However, the solution proposed in this document to the issue of separating long-running jobs from analysis one is based on VOMS, GPBox, and roles.

4.2. DILIGENT

The goal of the Diligent project is to provide a Digital Library (DL) Infrastructure on gLite middleware. User interaction takes place using portlets. An important aspect is that more or less everything in a DL is a moving target: data can be made available or renewed dynamically, or data can even be produced on the fly (e.g. by using workflows for assembling information from different providers into a new "live document".) Some sample use cases follow to illustrate the usefulness of or need for short jobs in Diligent:

Feature extraction: When multimedia content, such as pictures, is made available in the system, features are extracted from these images and indexed. This phase is not time-critical and could very well be handled using bulk submission. However, when a user wants to search for images similar to one she provides (query-by-example), feature extraction for the (single) user input has to take place, before the actual search can be carried out. Here, the user will be interactively waiting for a job to terminate.

Watermarking: For DRM reasons, multimedia data may have to be individually watermarked for the user requesting them. Same as above, the end user has to wait for a (short) job before being able to continue working.

General: Diligent is being built as a SOA; all resources and "data processing services" are made available as Web Services, with an explicit goal being the ability to combine the services into workflows (e.g. even the fundamental search functionality will be built using workflows). A job execution, from this point of view, will be "just another service invocation" and should be executed in a timely manner.

As can be seen from the first use case, there are two very distinct cases where grid jobs come into play. The first is, generally speaking, "non-user-driven" and therefore not (really) time-critical: the addition of resources to a DL, system-internal functionality such as indexing etc. In the second case, a user is actively waiting for data produced by a job. Since the jobs themselves are rather simple (seconds, maximum minutes), they clearly fall into the category of short jobs where (for now) the middleware overhead outweighs the actual execution time. Since every execution of a SDJ is triggered by an end-user, it cannot be foreseen when it will take place; therefore, advance reservation would be impossible to schedule, and bulk submission is no solution either.

On the other hand, the frequency of such short jobs could be quite high (see use cases). While Diligent has "interactivity" requirements in the sense of users "sitting and waiting for jobs to terminate", the end-user interacts with the DL system through a web portal only; users should not be able to "manipulate a running job", as pilot jobs or agent-based systems (or gLogin) would permit. Although pilot jobs might help in some particular cases, their usefulness seems limited especially because of the unpredictability of the point in time where an actual execution is needed. As a consequence, the regular way of submitting jobs is preferred.

4.3. BIOMED APPLICATIONS

Some of the biomedical applications have a need for support for short jobs. However, their motivations are somewhat different.

4.3.1. gPTM3D

gPTM3D is a grid-enabled interactive medical image visualization and processing tool, based on the PTM3D software. With respect to short jobs, the more important characteristics of gPTM3D are

- real short duration: 2min required for a typical job, but which is in fact a collection of very fine-grained independent tasks. Self-scheduling of these tasks is required, because of the large and unpredictable dispersion of their duration

- strong interactivity: the user interacts with the computation on a local (Windows) PC running a highly-specific interface, and holding all or part of the dataset (computational steering).

- The front-end can only be assumed to run a lightweight client of grid services, not to be fully grid-compliant (especially with respect to authentication).

In summary, gPTM3D is the typical example of transparent integration of grid into a local and fully interactive session. In particular, it exemplifies the need for agent scheduling created by the strong interactivity case. Any interactive system must produce a stream of intermediary results which make sense for the end user (otherwise there would be no place for interaction); translated in the grid framework, this means that the overall job is composed of a set of tasks. These tasks are very tiny, if interaction has its usual sense. As it cannot be reasonably expected that the grid middleware latency drops to the order of the second, agent scheduling is required.

gPTM3D has developed an architecture for interaction based on an agent scheduler system dedicated only to the management of tasks (vs jobs). The agents by themselves are submitted as regular EGEE jobs. Thus they could suffer from the long submission latency. On the other hand, they can use any "short queue" system. In fact, the SDJ CE has initially tested by gPTM3D jobs.

4.3.2. GPS@

GPS@ (http://gpsa.ibcp.fr/) goal is to be an integrated grid portal devoted to molecular bioinformatics. GPS@ is a porting experiment of the NPSA (Network Protein Sequence Analysis) services onto the EGEE grid (NPSA is a production web portal hosting proteins databases and algorithms for sequence analysis).

NPSA/GPS@ submits a large number of short jobs. The long waiting time induced by LCG2 middleware events provides an unacceptable response time for the users. GPS@ requires short execution time for short jobs. Also, on the NPSA portal, bioinformaticians do not authenticate individually to run jobs. GPS@ requires a service certificate so that the server can start bioinformatics jobs on behalf of the users as it is routinely done in this community.

4.3.3. Clinical Decision Support System (CDSS)

The objective of the application is to train and use in production, classification engines according to large medical databases. Training is a very high-throughput, compute-intensive task and fits other types of jobs better (MPI jobs). However, production usage is a different use case: The user browses through the information system (R-GMA or MDS currently) the sites where the classification engines are installed. Those sites publish information about the features of the classifier (corpus, target, accuracy etc.). The user picks-up the most appropriate ones according to the test (s)he has to make. Then, there are several parameters that could be tuned in advance, so the user makes several executions with just one single entry (it takes around 30-50 seconds each). Once the results are satisfactory, the complete database is tested (requiring several hours). The first kind of jobs must be

scheduled with a low overhead and with the minimum queuing time. Large submission can go through normal procedures.

So the jobs can be characterized as:

- A few, short-duration jobs having very little impact on a large batch system.
- Asynchronous and sequential.

Then the requirements are:

* Minimum latency and grid overhead. Bulk submission is not a solution.

* Minimum queuing time. Impact is low, so co-location of jobs in batch queues satisfies the requirements.

* Maximum transparency. Although manual resubmissions can be dealt with, a more automatic solution is desired.

* connectivity is not an issue in our case; thus, standard scheduling (with improved performance) is our preferred choice (vs agent scheduling

4.4. OTHER

Other applications involving from short to ultra-short turnaround time have been presented at the special jobs session at the User Forum, targeting financial applications (ultra-short jobs), remote instrument control and game playing. However, these applications did not contact the SDJ WG.

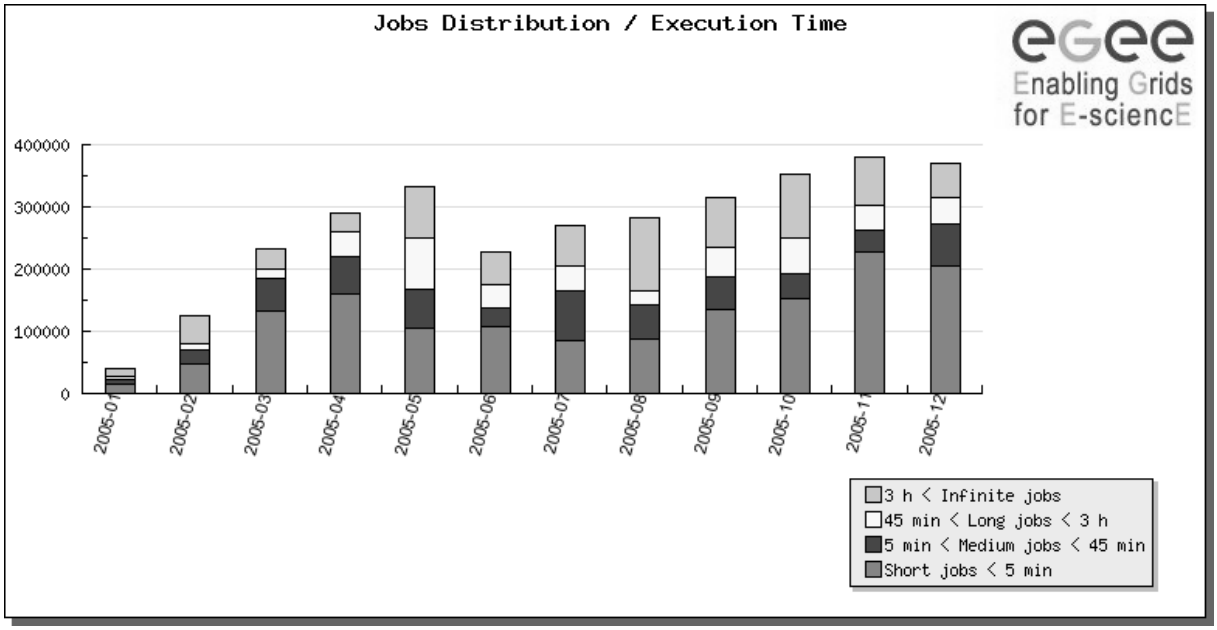
4.5. SUMMARY OF REQUIREMENTS

Application	Typical (or range of) Execution time	Expected request frequency	Other requirements related to "shortness"	Amenable to bulk submission	Workflow	Scheduling
CMS	< 10mn	Frequent	None	No	No	Regular preferred
gPTM3D	2mn (a)	Infrequent	Strong interactivity	No	No	Agent required
Diligent	Second to minute	Frequent	None	No	Yes	Regular preferred
CDSS	< 1mn	Infrequent	None	No	No	Regular preferred
gps@						

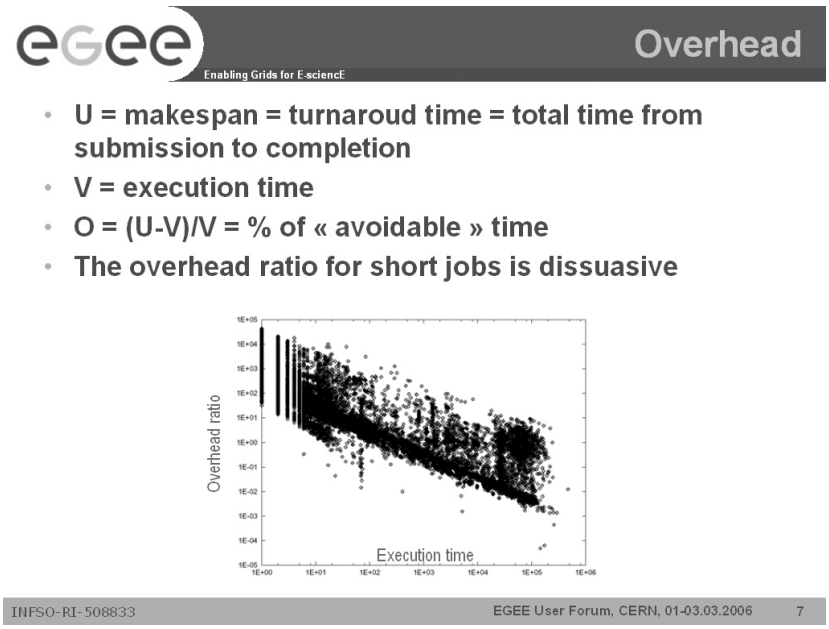
(a) A gPTMD job is a set of hundreds of microtasks, the 2 min limit is the aggregation of the CPU used by the microtasks.

4.6. SOME EXPERIMENTAL DATA

The frequency of short jobs is shown the next figure, from the JRA2 statistics site (dteam is excluded).



The current service given by the middleware to short jobs is not satisfactory, as shown by the next figure: for short jobs, the overhead is orders of magnitude larger than the execution time (the figure exploits data from one year L&B for the LAL broker).



4.7. SUMMARY OF USER REQUIREMENTS

In order to get the required QoS, the following requirements have been identified.

- Scheduling policy: immediate access to the processors, which means no queuing.
- Scheduling implementation: lower delays in the various stages of the job lifecycle.
- High-level services:
  - A generic user-level scheduling service.
- Low-level Services, or software components
  - Connectivity between the UI and WNs
  - Dynamically attach a shell to a running process (especially for debugging purpose)
  - Interactive inspection of the state of the queues

For the current application panel, there was no need for inter-site connectivity *as far as SDJ are concerned*[1].

Moreover, it was recognized that explicit advance reservation is not compatible with the users practice.

[1] But the same users may need it for non-SDJ jobs.

5. SCHEDULING POLICY

A prototype solution for *addressing the issue of sharing processors between batch jobs and SDJ jobs* has been developed at LAL/LRI. The conceptual solution is relatively independent of the submission system. It allows SDJ jobs to run concurrently with batch jobs. The fraction of the resource usage devoted to SDJ is locally configurable.

5.1. FUNCTIONAL DESCRIPTION

5.1.1.1. A SDJ is:

- a regular EGEE job;

- which should not be delayed by any other job (whether batch or SDJ) provided that compatible resource do exist at the time where the job is examined for scheduling (more on that later);

- if no compatible resource can be found, the job must be rejected, and notified of the rejection within a decent delay.

5.1.1.2. A SDJ compatible resource is a site

- which is prepared to run SDJs, that is compliant with the previous requirements;

- and complies to all other requirements expressed by the job, typically through the JDL (eg file location, software, site, etc), and site access policy (VO, user, etc.)

5.1.1.3. Constraints on implementation and deployment

- Only some sites will be willing or able to accept SDJ.

- SDJ implementation should ensure that

  - Modifications to the core middleware are extremely limited, and orthogonal to (have no impact on) existing features;
  - The delay incurred by batch jobs remain bounded, and to a reasonable bound;
  - The resource usage is not degraded, e.g. by idling processors
  - The reconfigurations of local management systems (eg MAUI, PBS, LSF,…) is orthogonal to existing policies governing resource sharing (VOs, EGEE and non-EGEE users)

- All the configuration process related to SDJ should

  - Be automated in order to provide a unique coherent process
  - Take place at the same time-scale as other reconfigurations

5.2. REFERENCE IMPLEMENTATION: TORQUE/MAUI CONFIGURATION FOR THE SDJ CE

The configuration of torque and maui described here adds a set of job slots dedicated to short deadline jobs. Jobs running in these job slots run concurrently with any other jobs scheduled on the same machine.

5.2.1. Required Software

The configuration described here requires torque version 1.2.0p3 or later. This can be found from the LAL repository [http://quattorsrv.lal.in2p3.fr/packages/mpi]. The version of MAUI distributed with the LCG release is sufficient.

5.2.2. Torque Configuration

5.2.2.1. Queue Configuration

Create a dedicated queue to handle the short deadline jobs. You **must** use the name "sdj" for the queue. RunTimeEnvironment flags are associated to a cluster rather than a queue, so cannot be used to distinguish unique queue characteristics. This creates a "class" within maui which will be used to control the scheduling of the short deadline jobs. The types of jobs typically take a small amount of total CPU time and possibly a significantly longer "wall clock" time. The queue parameters should reflect this. The total number of jobs will typically be much smaller than the total number of CPUs advertised; consequently, the maximum number of jobs should be specified. Typical parameters follow:

```
max_running = 4
resources_max.cput = 00:10:00
resources_max.walltime = 00:30:00
queue_type = Execution
enabled = True
started = True
```

The maximum number of running jobs may be smaller than the total number of job slots available. This is a way of limiting the impact of these jobs on the total system.

5.2.2.2. Submission Filter Script

Ideally short deadline jobs will either be scheduled immediately or fail. It is possible to tell maui that jobs should be removed from the queue if they cannot be scheduled immediately. Adding the required flag can be accomplished with a submission filter.

Modify or create the file /var/spool/pbs/torque.cfg by adding the line SUBMITFILTER /var/spool/pbs/submit_filter. This parameter will cause qsub to execute the named script for each job submitted to the system. This will be run as the user executing qsub so it must be readable and executable by all users (e.g. 0755).

Modify or create the named script to add a line like the following to the user's job script.

```
#PBS -W x="FLAGS:NOQUEUE"
```

All jobs submitted via the LCG/EGEE software have a line like:

#PBS -q sdj

which identifies the queue. We use this to add the NOQUEUE flag only for short deadline jobs. The following is a minimal perl script to accomplish this:

```
#!/usr/bin/perl
while (<STDIN>) {
    # By default just copy the line.
    $line = $_;
    # If there is a queue option, check to see if it is "sdj".
    # If so, then add the option to not allow such jobs to be
    # queued.
    if (m/#PBS\s+-q\s+sdj/) {
        $line .= "#PBS -W x=\"FLAGS:NOQUEUE\"\n";
    }
    print $line;
}
```

Test the script! If there are any errors all job submissions will fail.

Lastly a set of "virtual" processors must be created to accomodate these jobs. Maui will **not** schedule more jobs than processors declared in the `/var/spool/pbs/server_priv/nodes` file. In this file set the "np=X" parameter to the sum of the number of physical CPUs plus the permitted number of short deadline jobs. For example for a dual processor machine where two short deadline jobs are permitted, this would read "np=4".

5.2.3. Maui Configuration

Standing reservations within maui are used to reserve permanently the additional jobs slots for short deadline jobs. In the file /var/spool/maui/maui.cfg add a section like the following for each torque client machine:

```
# Short-deadline job reservation for grid26.lal.in2p3.fr
SRCFG[11] HOSTLIST=grid26.lal.in2p3.fr
SRCFG[11] PERIOD=INFINITY
SRCFG[11] ACCESS=DEDICATED
SRCFG[11] TASKCOUNT=1
SRCFG[11] RESOURCES=PROCS:2
SRCFG[11] CLASSLIST=sdj
```

The TASKCOUNT parameter should correspond to the number of short deadline jobs you are permitting on this host. You can verify the reservations with the "showres" command.

Note: The more logical configuration with TASKCOUNT=2 and PROCS:1 doesn't work correctly. Use a single task with the number of processors (slots) you want to reserve.

You must also add a line like:

```
CLASSCFG[sdj]            MAXPROC=4
```

to limit the total number of running short deadline jobs. This number should correspond to the max_running parameter of the torque configuration. (The torque/maui configuration will effectively use the minimum of the two parameters, but only the torque parameter will be published in the information system.)

As short scheduling latencies are needed for these jobs, the maui RMPOLLINTERVAL parameter should also be set to a short value (like 10 or 15 seconds).

5.2.4. Information System Configuration

This queue can be published in the information like any other queue. However, the access to this queue should be limited to a particular VO for testing (as long as the sdj attribute has not been implemented, see Section 4.4).

5.2.5. Example Job Description

Jobs wishing to take advantage of this functionality should include in the job description the following attributes:

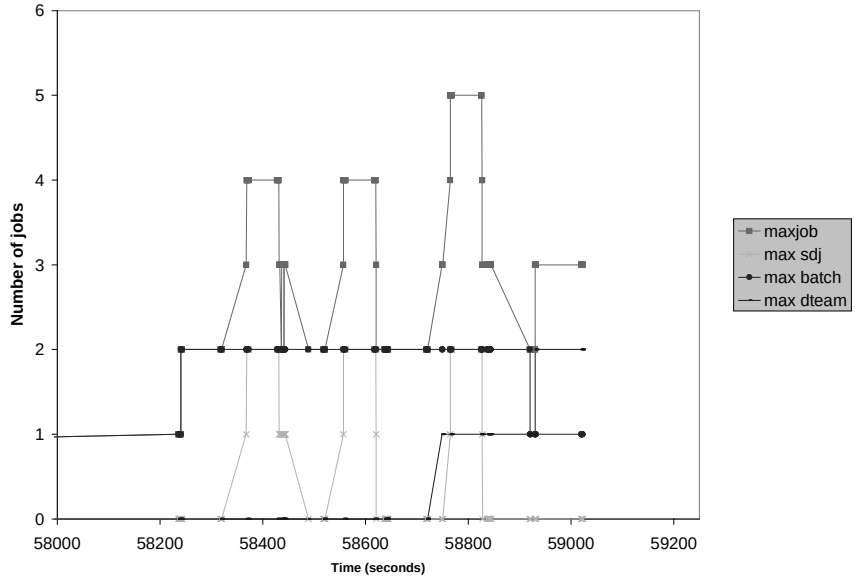
```
Requirements  = RegExp(".*jobmanager.*sdj$",other.GlueCEUniqueID) &&
    (other.GlueCEPolicyMaxCPUTime > 5) &&
    (other.GlueCEPolicyMaxWallClockTime > 15) &&
    (other.GlueCEStateRunningJobs < other.GlueCEPolicyMaxRunningJobs);
RetryCount    = 3;
```

where the numbers are adjusted appropriately. When the short deadline slots are full on a site, the batch system will reject the job and the resource broker will detect the error. Using the automatic resubmission will automatically try rescheduling the job on another resource (if there is more than one resource available).

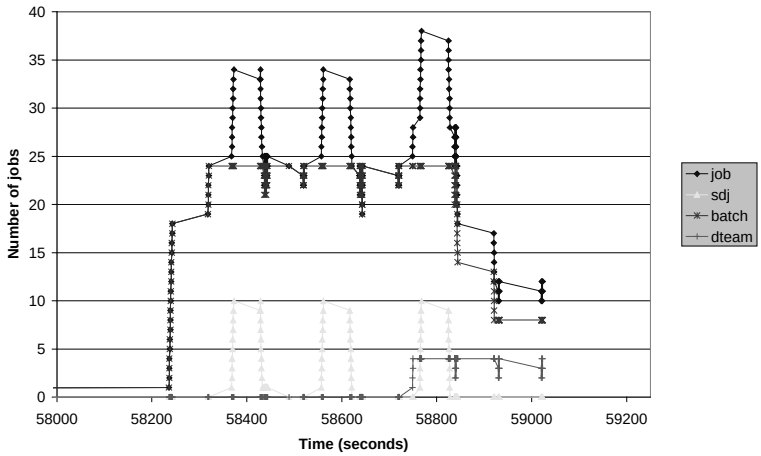
Note that the RegExp will be included at UI level in gLite 3.2 (see 4.4).

5.3. STATUS

This configuration has been extensively tested at LAL, and no errors have been detected. The first figure in this section shows the number of jobs run on one dual-processor machine, with the previous configuration, modified to accept also one dteam job. The data are obtained though automatic analysis of the MAUI log files. The limits on concurrent running jobs (at most 2 batch, 2 SDJ and one dteam) are never exceeded.



The second figure uses MAXPROC=10 as the limit to the total number of SDJ on the machine pool.



5.4. REQUIREMENTS TOWARDS JRA1

With the current software, this implementation will direct SDJ jobs to SDJ queues, **but will not prevent non-SDJ jobs to go to SDJ queues**. There is no way to express this requirement through the Glue 1.2 schema only at user level. JRA1 has proposed the following intermediate solution:

- Name SDJ CE so that they conform to a regular expression eg ” *.sdj” (site manager responsibility)

- Add a Boolean attribute in the JDL which flags short deadline jobs. Based on the value/presence of this flag, the job wrapper requirements will be modified to select SDJ computing elements and will prevent non-SDJ jobs from being scheduled on SDJ computing elements. (JRA1 responsibility)

The implementation of this requires minimal changes to the existing code.

The TCG decided that corresponding modification of the UI and WMS will be implemented in gLite 3.2.

This solution will work for some time, but is obviously not satisfactory in the long term. A CE should be described as dedicated to SDJ independently of its name. However, this implies a modification of the Glue schema.

5.5. KNOWN ISSUES

5.5.1. Consistency

Because of the relatively slow rate of BDII update in the push model, it might be possible that more jobs should be directed to a sdj queue than what it can accept. This effect is likely to be mitigated by the fact that the broker breaks ties by random choice. However, if this happens, the slow notification due to Condor(see below) will add a significant penalty, because a rejected job will have to wait for this notification to be retried.

5.5.2. Limits

Two different issues do exist about limits:

- whether or not to limit short-deadline jobs, either wall-clock, or CPU usage or both. Rationale: SDJ should not last long, otherwise they will block other SDJ; on the other hand, “interactive” sessions may last long, or agent schedulers and workers could be submitted as SDJs.
- if limits are used, what policy should apply when exceeding those limits: unconditional abort, or abort only if if there is another SDJ requirement (probably not feasible, needs a complex broker-wrlms protocol).

This has not yet been discussed in the WG.

5.6. PLANS

Two sites have created sdj queues (LAL and Valencia), and are currently willing to participate in testing the system when available.

When these second level test will have been completed, it is likely that the biomed VO will deploy sdj queues.

Further deployment might be an issue. GPBox will probably not help. SA1 and SA3 should be consulted to ensure that the SDJ configuration is available as a standard part of the release.

6. MIDDLEWARE LATENCIES

Glite 3.0 should provide better submission and matchmaking time. The following facts have been stressed:

- Since gLite 1.4 a lot of effort has been devoted in JRA1 to the profiling and optimization of released services, but the system limits (in terms of performance improvements) have been not yet hit.

- The speedups *do apply* to both simple jobs and bulk jobs. What is not completely clear is whether these improvements can be considered satisfactory for the needs of SDJ or not. When the new measurements made on the PPS will be available, the answer will be more clear.

- The notification delay (from end of execution to notification to the WMS and end user) will not improve with the current CE, because the delay is determined by the Condor polling frequency.

  For the last point, it was suggested to try configurations of both the broker and CE with increased frequency. This has been already explored (see details and performance plots at

  http://savannah.cern.ch/bugs/?func=detailitem&item_id=9692#comment7).

  It seems that any change in the configuration parameters has only adverse effects on performance. JRA1 will possibly provide more info, according to the comment (http://savannah.cern.ch/bugs/?func=detailitem&item_id=9692#comment8) ?

A large number of performance tests are currently performed by various teams, on the different services and components, including the CREAM CE. These are of particular interest to the SDJ users, and a faster and more systematic diffusion of the results of these tests would be very useful.

7. AGENT SCHEDULING

7.1. DEFINITION

This name refers to the various developments performed by many groups in order to enable user-level task management, with a scope limited (at run time) to a specific application. All are based on a master/slaves scheme (whether internally push or pull), but differ in the implementation technologies. The motivations are

- Avoid the middleware overhead.
- A more efficient scheduling, e.g. taking into account the user knowledge of the distribution of task execution time.
- User-level fault tolerance, e.g. through eager (or redundant) scheduling.

Agent scheduling is formally fully "EGEE compliant" in the sense that at least the slave processes are submitted as regular jobs, thus accountable. However, it might be not really work conserving in the sense that the slaves can in fact sit idle (waiting for tasks to be submitted by the master), while occupying slots. Some existing software is described below, but many others have been described at various EGEE events.

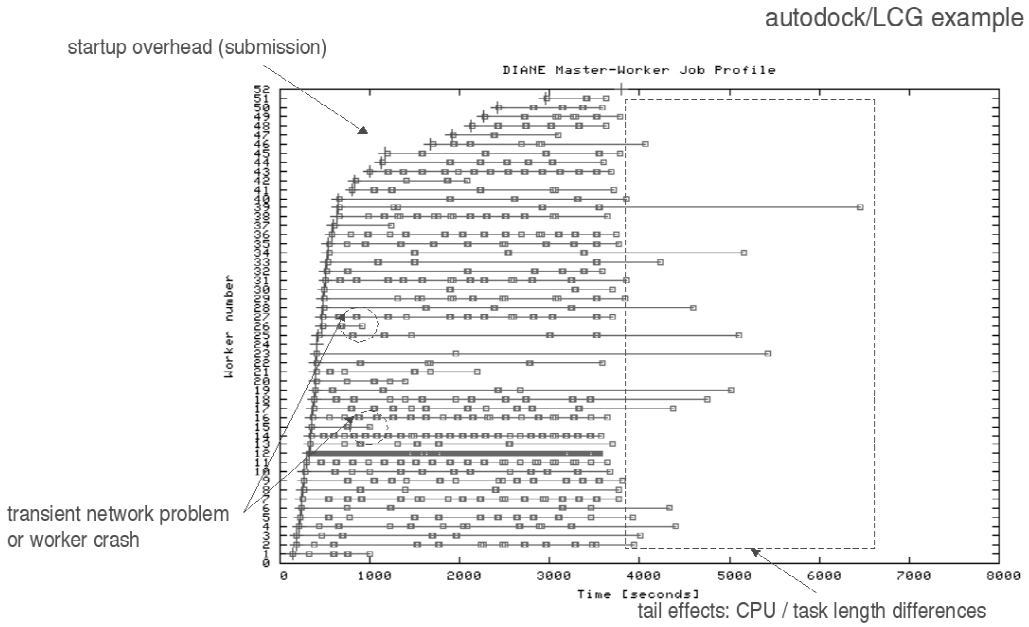
7.2. EXAMPLES

7.2.1. DIANE

The **DIstributed ANalysis Environment project** (DIANE) was started at CERN in 2001 to investigate the distributed analysis for High Energy Physics, but it soon involved applications from other research areas.

The current application portfolio includes: Atlas Athena AOD analysis, Geant 4 Simulation in medical physics and space science, Geant 4 release validation on the Grid, BLAST and Autodock applications in bio-informatics, special applications such as ITU frequency planning application for Regional Radio Conference 2006. DIANE exploits a number of submission backends: LCG/glite, Crossgrid, Condor, LSF/PBS and dedicated clusters. Internally DIANE uses Ganga as an abstraction layer to interact with the underlying backends.

The goal of the DIANE framework is to provide a customizable environment for efficient execution of parallel jobs in the Grid. It enables non-expert users to easily parallelize the existing, sequential applications in a Master-Worker model and to customize the runtime behavior of the framework according to application-specific needs. The Master handles variable length tasks (SDJs) by taking advantage of natural parallelism of the applications in a way which makes least assumptions about the underlying computing environment and application execution patterns. The task scheduling and load-balancing policies may be dynamically defined on per-application or per-job basis. Customizable error recovery mechanisms allow to efficiency capture and handle application and system errors. The customization mechanisms are flexible enough to accommodate complex synchronization patterns between tasks, so that DIANE may act as an algorithmic skeleton.



Job execution profile example: Autodock application running on LCG. Crosses indicate the worker agent registration time succeeded by line segments which indicate the individual tasks (SDJs).

Currently, the most frequent usage of DIANE is to submit Worker agents to the Grid and starting a Master agent close to the Grid UI. For the duration of the job, DIANE creates an overlay network with the direct communication channels between Master and Workers. When the job is finished, the resources are returned back to the Grid. DIANE provides sustained computing capacity in dynamic, heterogeneous and non-reliable environments, shields the users from the instabilities in the matchmaking process on the Grid and allows to minimize the submission latency overhead. Current prototype has proved to scale up to 300 simultaneous workers and several tens thousands of tasks. According to everyday experience DIANE may significantly (2-10 times) reduce the job completion time on the Grid.

Another usage scenarios are also possible with DIANE, including resource reservation: a user may acquire, for a short period of time, a large number of computing nodes with instantaneous feedback. This may be a suitable use-case for semi-interactive or interactive applications on the Grid.

The installation and deployment of DIANE is extremely lightweight and may be performed entirely in the user space. On the Grid worker nodes, the framework downloads and installs itself as a part of the Worker agent initialization procedure.

7.2.2. gPTM3D

The scheduling scheme is currently limited to the application, in order to serve the requirements of extremely lightweight tasks. In particular: the master/slave connection is raw TCP; the master and the slaves must be co-located on the same cluster; a “super-master” (interaction bridge-IB) acts as a client for the UI, and a server for window front-ends.

7.3. REQUIREMENTS

JRA1 is currently developing an “overlay” service or component. The SDJ WG asks for interaction with this development. Two issues seem especially important.

7.3.1. Discovery

The process of establishing connections between the master (or IB) and the slaves is not trivial, due to firewalls, asynchronous launching etc. This cannot be solved by DAG: obviously when the master does not run on a WN; but even in that case, the master must run in parallel with the slaves, thus no channel can be created between them is the DAG scheme. Many technologies provide a solution, but they may end up in tunneling the subsequent master-slave connection through extremely slow protocols.

7.3.2. Usability

The goal of such schedulers is to provide a tradeoff between versatility (plugin of any useful scheduling mechanism), performance, and ease of interfacing with applications.

7.4. DISCUSSION

Agents raise a problem, which is more or less the same as the one with "advance reservation": We do not know when a job will have to be executed. Therefore, with an agent-based solution, it might well be that we have agents running, just sitting there and waiting for commands which do not arrive, and blocking/wasting resources. So the main concern here would be the *overall* efficiency of the approach, while clearly the gain in efficiency for the individual jobs would be undisputable.

Assuming that agents are submitted as SDJ, the resource they would waste when idle is negligible *in CPU time*, because they will run concurrently with a batch job (if there is one eligible). Thus the waste will be only the kernel scheduling time, which can be considered as a tolerable tradeoff with respect to the advantage in responsiveness.

However, idle SDJ agents will actually waste *SDJ slots*. When an idle agent is running as a SDJ, no other SDJ can run on the same slot. The general idea for solving this issue is to use the fair share policy at the level of the local scheduler. This in turn goes back to the more general problem of policy enforcing in a context of autonomous schedulers (which is an active research area in Globus…).

In any case, if agents are submitted as SDJ, an additional piece of software is needed to hide the short duration of SDJ.

8. CONNECTIVITY

8.1. REQUIREMENTS

Two kind of requirements have been expressed in this area

- Interactive connection to a running process: for debugging and inspection, systematic code instrumentation for reporting through R-GMA is not convenient (because the problem requiring debug is not likely to have been anticipated). In some cases, job submission as interactive can help (the performance are announced to be much better now), but may be a limitation (e.g. for jobs that already redirect input/output).
- Port forwarding, in order to tunnel efficiently any protocol trough the firewalls blocking worker nodes.

8.2. GLOGIN

glogin provides a solution for both requirements, together with encryption (which could be disabled if desirable). Basically, It allows to interactively login into any CE or WN where one is allowed to run a job on.

Conversely, glogin jobs are naturally interactive, thus scheduling and running them within a reasonable short time is an important issue.

The latest version of glogin (V1.1.1) is available at http://www.gup.uni-linz.ac.at/glogin/. It will be integrated somewhere between gLite 3.0 and 3.2.

A demo of using glogin (provided by Martin Polak) for creating a master-slaves system is shown below.

8.2.1. Example

Source code for the demo at

http://clio.gup.uni-linz.ac.at/vprog.c

http://clio.gup.uni-linz.ac.at/pth_glogin_demo.c

(1) Start "vprog" on your client (=UI) machine, which will get the output from the grid and print it on stdout

```
hr@clio$ ./vprog -n 4 -p 1111 altix1
```

it does nothing special but launching a remote portforwarder on a machine called altix1, which is the CE in our case, using glogin. So we have a GSS-encrypted tunnel between host clio and altix1. That is what "-p 1111 altix" does: listen on port 1111 on the CE and forward it to the local client (UI). The option "-n 4" tells vprog to wait for a number of 4 forwarded connected workers (because in the next step we will launch 4 worker tasks).

(2) So we submit our 4 worker processes to the CE (in this demo via globusrun, but in principle one can also use his favorite broker for achieving this):

```
clio$ globusrun -w -r altix1/jobmanager-pbs '&(executable=./pth_glogin_demo)(arguments=-p 1111)' & [1] 2154
clio$ globusrun -w -r altix1/jobmanager-pbs '&(executable=./pth_glogin_demo)(arguments=-p 1111)' & [2] 2155
clio$ globusrun -w -r altix1/jobmanager-pbs '&(executable=./pth_glogin_demo)(arguments=-p 1111)' &[3] 2156
clio$ globusrun -w -r altix1/jobmanager-pbs '&(executable=./pth_glogin_demo)(arguments=-p 1111)' & [4] 2157
```

This will result in our workers landing on some WNs.

The demo jobs will then connect via TCP to the CE by evaluating the GLOBUS_GRAM_JOB_CONTACT variable. How this is achieved can be seen in the sourcecode.

The demo-workers are very simple and just increment a counter every second and send its value through the created tcp connections.

(3) as soon as "vprog" has been contacted by all four workers, it prints their hostnames ...

hr@clio$ ./vprog -n 4 -p 1111 altix1

hn0=[altix3]

hn1=[altix2]

hn2=[altix4]

hn3=[altix1]

and then the counter values:

[altix3: 132582]

[altix1: 36029]

[altix2: 2068457]

[altix4: 2173531]

[altix3: 599249974]

[altix1: 597688388]

[altix4: 601182158]

[altix2: 600257231]

[altix3: 1198554392]

[altix1: 1195829145]

[altix4: 1200285888]

[altix2: 1199478310]

[altix3: 1797838840]

[altix1: 1794014300]

[altix4: 1799319154]

[altix2: 1798650906]

8.2.2. More details on glogin

The glogin process work as follows. It submits a job for launching its remote counterpart (if it is installed with the remote middleware) or stages itself to the remote machine if not installed. This job gets queued in the LRMS (so that is where glogin is related to SDJ-queues).

For instance

glogin -g egee-ce1.gup.uni-linz.ac.at brings you onto the CE (using fork-jobmgr)

glogin -g egee-ce1.gup.uni-linz.ac.at/jobmanager-lcgpbs to one of its WNs

the "-g" stages the executable to the place where it will be run und you will have a login-shell for your VO-mapped account on that machine

Then it tries to build two (1 inbound and 1 outbound) connections between the UI and the remote binary on either the WN or the CE (depending on where the job has been submitted to, using GLOBUS_TCP_PORTRANGE). The first of the two connections to succeed survives, the other gets canceled. The data on that connection is encrypted using gsswrap calls from glogin.

8.2.3. Performance

Using glogin and communication over the pty gives a CPU-imited maximum transfer rate of about 10 MB/s on decent CPUs. This is caused by the gss functions used for de/encryption, which depends on the used block size. They reach a maximum with data blocks of about 32 KB size, but standard ptys use 4KB for transfer units. When using tcp port forwarding or plain pipe communication, there should be no additional overhead.

8.2.4. Issues

8.2.4.1. Load balancing

If any VO member is allowed to to login interactively on any WN, this might disrupt batch load balancing and fair share?

Not really. Currently, the remote part of glogin is started using the Globus GRAM interface, which implies that it is submitted via the various jobmanagers installed at the CE. So if and only if the CE allows 'globus-job-submit'-type jobs from any (UI)-Machine, it will work. But the job will still have to go through the LRMS (it its queues) in order to get glogin started there. So it doesn't really disrupt local load balancing.

On the other hand, since the remote part of glogin performs a fork and changes the sid, some local resource manager falsely (such as PBS) assume this job being terminated, and schedule the next job on the resource. Therefore, an interactive glogin session can interfere with a subsequent job. However, this issue only demonstrates a weakness in the current implementations of local resource managers. Doing something as described above in your own application and staging it on a resource is easy, thus, 'curing', this issue can only be done by improved implementations of the local resource managers.

8.2.4.2. Traceability

Jobs launched through glogin are not currently in the scope of the WMS. How can I get information about them?

glogin currently only uses basic job submission without going through the WMS. Thus it is not an egee-job, but run as gssexec-job. So one could get a job contact to query its status within the LRMS too, but no info is available in the WMS. There is a practical impact for SDJ. Typically, SDJ jobs are not certain to be run (even on a fault-free hardware/middleware), because there is limited capacity for them. This notification goes through the WMS.

A full integration is worked on by the glogin team.

8.2.4.3. Usability

The following use case has been raised: use glogin to land on a given WN in order to inspect the environment of a running job (e.g. attach it with gdb and do a backtrace or similar). In this glogin could not really help, because there is no guarantee to land on a *given* WN.

8.3. JOB INSPECTION

When the problem is not connectivity in general, but limited to getting information or interacting with a running job, some other solutions do exist.

- The **Interactive** job type
- the **Job's File Perusal** functionality of the gLite 3.0 WMS that allows inspecting job's files while the job is running. There is no direct access to the CE/WN; the WMS acts as a bridge between the users and the job.
- The **JobInspect Service** that will come with the CREAM CE.
- The **JobMon** tool developed and currently in use by the CDF experiment

The initial motivation for inspection is to help the user to assess the quality of her execution, and thus targets long running jobs. However,

- It obviously provides faster turnaround of debug sessions, trial runs and other kinds of tests wrt only batch scheduling.

- It worth considering if the mechanisms that have been set up for this purpose can be of help for the strong interactive case of SDJ.

8.3.1. Summary of JobInspect

The general idea is interactive read-only access to a running job's environment. Thus, it is "one-way interactivity": information flows from job to user.

The goal is to let a user monitor her job's stdout, stderr and output files in real time.

This service is developed by the CREAM group at INFN PD. The goal of this sort of "restricted shell" service is to provide the grid user with a read-only access to the specific WN where her job is running, with the privileges of the local user, and only while the job is running (RUNNING status). The admitted operations will be more or less, in UNIX terms:

ps, top, ls, cat, head, tail, more/less/view

In particular any operation that could modify explicitly the status of the WN will be blocked (i.e., cp, mv, rm, kill, vi... will not be available).

JobInspect is realized as a web service (running on a CREAM CE) accepting user commands, interpreting, filtering and executing them on the WN through a SSH connection on the CE internal LAN.

A C++ command line will be provided to the user to send commands to the JobInspect service.

8.3.2. More details on JobInspect

- The CREAM JobId is the only parameter needed
- Remote ps, top, ls, cat and tail-like functionality on the Worker Node
- Intelligent browsing of remote files: client-side hex viewer and view-like functionality only transfers needed chunks of the remote file as needed
- GUI clients are possible, although not currently scheduled
- Implementation

C++ client <---(1)--- Specific CE webservice <---(2)--- Worker Node

(1) glite-authenticated SOAP messages, through the Internet

(2) SSH as the local user, through the CE LAN

- Security considerations
  - Access to the service is subject to the same authentication as CREAM is
  - The user has only access to worker nodes where one of her jobs is running
  - She may only issue a fixed set of commands, none of which can alter files
  - User-supplied arguments are strictly parsed against shell escaping
  - No inbound connectivity is requested on the WN
- Privacy considerations
  - SOAP messages, including all traffic payload, are encrypted with SSL
  - The set of files / directories / devices the user has read access to on the worker node is restricted by the same OS file permissions as her job's
  - Additional filters can restrict the commands to the job's working directory

- http://jobmon.sourceforge.net/

8.4. THE JOBMON TOOL

Same functionalities are provided by the JobMon tool developed and currently in use by the CDF experiment:

http://jobmon.sourceforge.net/

This is not really a 1:1 replacement, security and deployment is done quite differently. Eg JobMon e.g. does not rely on SSH on local cluster, but needs one Clarens server somewhere.

8.5. INTERACTIVE JOB TYPE

The Interactive type of jobs mixes Remote (restricted) shell and Stream oriented connection. sdtout and stderr are piped to the UI, thus stream-oriented. On the other hand, stdin is formally piped, but in reality, the Condor Bypass library is used to hook shell-related system calls (eg mkdir), in order to call its remote equivalent. The bandwidth performance have been tested at LAL/LRI in 2003 and were **extremely poor**, in the order of a few hundreds of bytes (not KB, bytes…) per second.

8.6. DISCUSSION

Many tools are available for "shell forwarding". Thus job inspection should not be a real issue anymore.

On the other hand, these tools are not convenient for data streaming, because they target shell commands, not plain data streams. glogin may offer a comprehensive solution if it could be integrated with WMS.