Analyzing the EGEE production grid workload: application to jobs submission optimization

Diane Lingrand[1], Johan Montagnat[1], Janusz Martyniak[2], David Colling[2]

[1] University of Nice - Sophia Antipolis / CNRS - FRANCE
{lingrand, johan}@i3s.unice.fr
http://www.i3s.unice.fr/~lingrand/
[2] Imperial College London, The Blackett Lab - UK
{janusz.martyniak,d.colling}@imperial.ac.uk

Abstract. Grids reliability remains an order of magnitude below clusters on production infrastructures. This work is aimsed at improving grid application performances by improving the job submission system. A stochastic model, capturing the behavior of a complex grid workload management system is proposed. To instantiate the model, detailed statistics are extracted from dense grid activity traces. The model is exploited in a simple job resubmission strategy. It provides quantitative inputs to improve job submission performance and it enables quantifying the impact of faults and outliers on grid operations.

1 Introduction

In response to the growing consumption of computing resources and the need for global interoperability in many scientific disciplines, inter-continental production grid infrastructures have been deployed over recent years. Grids are understood here as the federation of many regular computing units distributed world-wide, taking advantage of high-bandwidth Internet connectivity. Productions grids are systems exploiting dedicated resources administrated and operated 24/7, as opposed to desktop grids that federate more volatile individual resources. The production systems operated today (*e.g.* EGEE[1], OSG[2], NAREGI[3]...) have emerged as a global extension of institutional clusters. They federate computing centers which operate pools of resources almost autonomously. The grid middleware is designed to sit on top of heterogeneous, existing local infrastructures (typically, pools of computing units interconnected through a LAN and shared through batch systems) and to adapt to different operation policies.

These complex systems have passed feasibility tests and are exploited as the backbone of many research and industrial projects today. They provide users with an unprecedented scale environment for harnessing heavy computation tasks and building large collaborations. Their exploitation has led to

---

[1] Enabling Grids for E-sciencE, http://www.eu-egee.org
[2] Open Science Grid, http://www.opensciencegrid.org
[3] NAREGI, http://www.naregi.org

new distributed computational models. However, they also introduce a range of new problems directly related to their scale and complex software stacks: high variability of data transfer and computation, heterogeneity of resources, many opportunity for faults, hardware failures, difficulty with bug tracking, etc. The workload management system of grid infrastructures is probably one of the most critical and most studied service provided. Despite the tremendous efforts invested in guaranteeing reliable and performing workload managers, the current records demonstrate that grid reliability remains an order of magnitude lower than clusters reliability, and performances may be disappointing when compared to the promise of virtually unlimited resources aggregation. As a consequence, grid users are directly exposed to system limitations. They adopt empirical application level strategies to cope with the problems most commonly encountered.

Production grid infrastructures remain to a large extent complex systems with behavior that is little understood and for which "optimization" strategies are often empirically designed. The reason for this cannot be attributed to the youth of grid systems alone. The complexity of software stacks, the split of resources over different administrative domains and the distribution at a very large scale makes it particularly difficult to model and comprehend grid operations. Structured investigation techniques are needed to analyze grids behavior and optimize grid performances. Considering the grid workload management systems in particular, users are often in charge of manually resubmitting jobs that failed. They need assistance to adopt smart resubmission strategies that improve performance according to global criteria.

1.1   Objectives and organization

In this paper we analyze the operation of the EGEE production grid infrastructure and more particularly its Workload Management System (WMS) in order to assist users in performing jobs submission reliably and improving application performance. Experience shows that EGEE users are facing a significant ratio of faults when using the WMS and their applications' performance is impacted by very variable latencies. Each job submitted to the grid may succeed, fail, or become an outlier (*i.e.* get lost due to some system fault). The execution time of successful jobs is impacted by the system latency. Faulty jobs and outliers are similarly introducing variable delays before the error is detected and the jobs can be resubmitted. From the user point of view, the overall waiting time, including all necessary resubmissions, should be minimized. Ad-hoc fault detection and resubmission strategies are typically implemented on a per-application basis. Determining the optimal grace delay before resubmission is difficult though, due to the absence of notification of outliers and the impact of faults. The objective of this study is to provide quantitative input and optimal resubmission timing.

Previous works have demonstrated that statistics collection on the live grid system and derived probabilistic models could help in optimizing grid performance according to user-oriented and system-oriented criterions [1, 2]. However, the statistics utilized so far were collected through invasive probing of the grid

infrastructure, thus leading to rather sparse and incomplete data retrieval, difficult to update although grid workload is non-stationarity. This work describes a more structured approach leveraging on the international effort to set up a *Grid Observatory*[4] which tackles the problems related to grid operation traces collection in order to provide accurate, dense and relevant statistics for modeling and optimizing the infrastructure.

In the remainder, the EGEE grid architecture, and more specifically its Workload Management System, is introduced. The Grid Observatory implementation, based on grid service log files analysis and merging, is then described. The data extracted and its exploitation for deriving a novel probabilistic model of the grid job latencies is presented. Finally, a simple job resubmission strategy is optimized, based on the probabilistic model proposed.

1.2 Related work

Frachtenberg and Schwiegelshohn [3] have pointed that in case of failure, rescheduling is needed in order to reduce submission cost. They also pointed out that very few real production grid workload traces and models are available. A few local pieces os work have been done however, such as on the Auvergne regional part of EGEE by Medernach [4]. An initiative of data publication and organization is the Grid Workloads Archive [5] which proposes a workload data exchange format and associated analysis tools in order to share real workload data from different grid environments.

Early work such as [6] have set up a methodology of statistical workload modeling from real data with the characteristics observed on Grids: heavy tailed distribution and rare events. More recent works have proposed to model different parameters such as job inter-arrival time, job delays, job size and their correlation on different platforms: the EGEE grid [7, 8] or the Dutch DAS-2 multi-cluster environment [9] for different periods of time from 1 month to 1 year.

Real workload models are mandatory to test new algorithms at different stages of jobs life-cycle such as submission (client side) or scheduling (middleware side). Authors of [8] have used their data to compare two user-level schedulers algorithms.

Workload models are also used for platform analysis and comparison. For example, authors of [7] have compared their results on the EGEE Grid with a real local cluster and an ideal cluster.

Finally, workload models will also enable more realistic simulations when used in Grid simulator such as SimGrid [10].

2   EGEE grid infrastructure

EGEE is an unprecedented large scale federation of computing centers, each operating internal clusters in batch mode. EGEE today accounts for more than

---

[4] Grid Observatory, http://www.grid-observatory.org

80,000 CPU cores distributed in greater than 250 computing centers of various sizes. With more than 9,000 users authorized to access the infrastructure and more than 200,000 computing tasks handled daily, EGEE experiences very variable load conditions and strong latencies in user requests processing, mostly due to the middleware latency and the batch queuing time of requests.

EGEE operates the gLite middleware[5]. gLite is a collection of interoperating services that cover all functionality provided, including grid-wide security, information collection, data management, workload management, logging and bookkeeping, etc. A typical gLite deployment involves many hosts distributed over and communicating through the WAN. The main services provided by gLite are: the security foundational layer (based on GLOBUS Toolkit 2), the Information System collecting status information on the platform hierarchically, the Data Management System providing a unified view of files distributed over many sites, and the Workload Management System (WMS) in charge of dispatching and monitoring computing tasks. Each of these systems is a compound, distributed architecture in its own right.

EGEE is a multi-sciences grid and EGEE users and resource are grouped into Virtual Organizations (VOs) which define both communities of users sharing a common goal and an authorization delineation of the resources accessible to each user group.

2.1   EGEE Workload Management System

The EGEE WMS is seen from the user as a two-level batch system: the User Interface (client) connects to a Workload Manager System (WMS). The WMS is interfaced to the grid Information System to obtain indications on the grid sites status and workload conditions. It queues user requests and dispatches them to one of the sites connected. The sites receive grid jobs through a gateway known as Computing Element (CE). Jobs are then handled through the sites' local batch systems. To comprehend the complexity of the system, a more complete view of the WMS architecture, extracted from the WMS user guide [11] is depicted in figure 1.

When submitting a job, the client User Interface connects to the core Workload Manager through a WMProxy Web Service interface on the Network Server. The Workload Manager queries the Resource Broker and its Information Super-Market (repository of resource information) to determine the target site that will handle the computation task, taking into account the job specific requirements. It then finalizes the job submission through the Job Adapter and delegates the job processing to CondorC. The job evolution will be monitored by the Log Monitor (LM) which intercepts interesting events (affecting the job state machine) from the CondorC log file. Finally, the Logging and Bookkeeping service (LB) logs job events information and keeps a state machine view of the job life cycle. The user can later on query the LB to receive information on her job evolution.

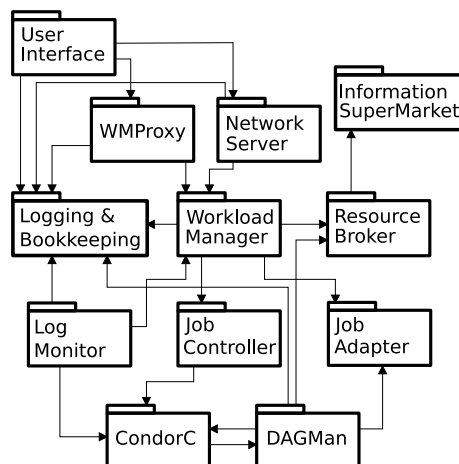[5] gLite middleware, http://www.glite.org

Fig. 1. gLite Workload Management System architecture; source: WMS user guide.

For load balancing and system scalability, the EGEE infrastructure operates around a hundred of similar WMS. However, if those WMSs share the same population of connected CEs, they are not interconnected and they do not perform internal load balancing. It is up to the clients to select their WMS at submission time. The client User Interface implements a simple round-robin WMS selection policy to assist users in their job submission process.

In the remainder we are particularly interested in the impact of the grid middleware on the job execution time, *i.e.* the *latency* induced by the middleware operation, that is not related to the job execution itself. This latency is a measure of the middleware overhead. In case of faults (scheduling problems, middleware faults...) this latency will arbitrarily increase and the job needs to be considered lost after a long waiting time enough to prevent application blocking.

2.2   Job's life cycle

The job's life cycle is internally controlled through a state machine displayed in figure 2 (source: WMS user guide).

The normal states assigned to a job are underlined in boxes with thick borders (they correspond to the case of a job completed successfully):

SUBMITTED the job was received by the WMS and the submission event is logged in the LB

WAIT the job was accepted by WM, waiting to match a CE

READY the job is sent to its execution CE

SCHEDULED the job is queued in the CE batch manager

RUNNING the job executes on a worker node of the target site
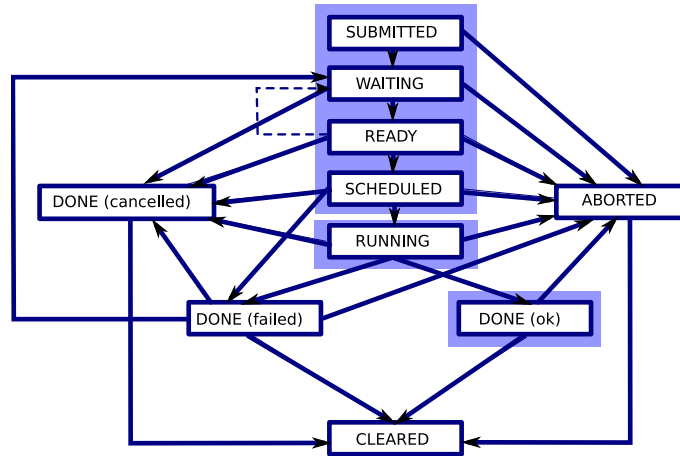
DONE (ok) the job completed successfully.

Fig. 2. Jobs life cycle state machine; source: WMS user guide.

Other states may also be encountered:

ABORT in any state, the middleware can abort the operation. An additional status reason is usually returned.

DONE (failed) some errors may prevent correct job completion. An additional status reason is usually returned.

DONE (cancelled) in any state, the submission user can cancel her job.

CLEARED after outputs of a completed job have been retrieved by the user, the job is cleared.

2.3   Grid observatory

The basis of our work on the WMS behavior modeling is the collection of statistical information on job evolution on the live grid infrastructure. The relevant information for performance modeling is the duration of jobs, including fine details on the intermediate times spent between transitions of the state diagram. This information collection step is difficult by itself. In a previous work [12], we collected such information through poll jobs submission on the infrastructure and monitoring of the polls life-cycle. Although this strategy is easy to implement (all that is needed is a user interface connected to the infrastructure), it is both restrictive (the polls are specific short duration jobs, the jobs are limited to the resources accessible to the specific user performing submission) and has limited accuracy (only a limited number of polls can be simultaneously submitted to avoid disturbing normal operation, the traces are collected by periodic polling and the period selected impacts the accuracy of results).

A more satisfying approach is to collect traces from regular jobs submitted on the grid infrastructure during normal grid operation, thus assembling a com-

plete and accurate corpus of data. However, there are much more difficulties in implementing this approach than would be expected, including:

- traces are recorded by different inter-dependent services (WM, CondorC, LM, LB...) that are tracing partly redundant and partly complementary information;
- traces are collected on many different sites (operating different WMSs) administrated independently: agreement to collect the data has to be negotiated with the (many) different site administrators;
- different versions of the middleware services co-exist on the infrastructure and traces are produced by slightly varying sources (including changes in states, labels, spell fixing in messages returned, etc);
- traces are recorded on different computers which clocks are not always well synchronized (although NTP *should* be installed on every grid hosts);
- traces collected are incomplete as parts of them can be lost (log files loss, disk crashes, etc) and all job states are not always recorded (middleware latency and faults cause some transition losses);
- as it will appear in the rest of this paper, the traces recorded often do not match precisely the information documented in the existing guides (state name changes, etc).

The most accurate source of traces available on the EGEE grid today is the Real Time Monitor[6] (RTM) implemented at the Imperial College London for the need of real time grid activity monitoring and visualization. The RTM gathers information from EGEE sites hosting Logging and Bookkeeping (LB) services. Conforming to our college policy, information is cached locally at a dedicated server at Imperial College London and made available for clients to use in near real time.

The system consists of 3 main components: the RTM server, enquirer and an apache Web Server which is queried by clients. The RTM server queries the LB servers at fixed time intervals, collecting job related information and storing this in a local database. Job data stored in the RTM database is read by the enquirer every minute and converted to an XML format which is stored on the Web Server. This decouples the RTM server database from potentially many clients which could bottleneck the database.

The RTM also provides job summary files for every job as text files ("Raw Data"). These data are analysed off-line and fixed record length tuples are created on daily basis, one file per LB server. These files are used for the analysis presented in this paper.

The systematic collection of grid traces for studying grid systems has been recognized as a key issue and significant effort has been recently invested in setting up a *Grid Observatory*[7] which aims at collecting information and easing access to it through a portal. The grid observatory has long term objectives of curating and harmonizing the data. It currently provides access to first data corpuses collected in Paris regional area (GRIF) and by the RTM.

---

[6] Real Time Monitor, http://gridportal.hep.ph.ic.ac.uk/rtm
[7] EGEE Grid Observatory, http://www.grid-observatory.org

3   Statistical data

The data considered in this study are RTM traces of the EGEE grid activity during the period from September 2005 to June 2007. 33,419,946 job entries were collected, each of them representing a complete job run. Among the information recorded in an entry can be found: the job ID, the resources used (UI, RB, CE, WN), the VO used, the job specific requirements, the job life cycle concatenated field and a complementary "final reason" text detailing the reason for the final state reached. Different epoch times are given, allowing to measure the duration of each step of the job life cycle:

epoch_regjob_ui: registration of a job on a User Interface
epoch_accepted_ns: job accepted by the network server
epoch_matched_wm: job matched to a target CE
epoch_transfer_jc: job accepted and being transfered to the CE
epoch_accepted_lm: job accepted by the CE
epoch_running_lm: job started running (logged by the LM)
epoch_done_lm: job completed (successfully or not)
epoch_running_lrms: job started running (logged by the LRMS)
epoch_done_lrms: job completed (successfully or not)

The last two couples of epoch data can be redundant: one is given by the LM while the other is given by the local resource management system (LRMS) or batch system. The LM data is less accurate than the LRMS, but the LRMS data does not exist for all CEs.

The `life cycle` field holds information on the different states the job has encountered during its life cycle (see figure 2). It is composed of the concatenation of the different state names, considering some minor variations in names (e.g. RAN corresponds to a past RUNNING state; REGISTERED corresponds to a job registered on the UI it has been SUBMITTED to). In the data considered in this paper, 77 different values of the `life cycle` field have occurred with different frequency. They correspond to different situations: job successfully terminated and data retrieved (REGISTERED_DONE_RAN_CLEARED), job aborted (REGISTERED_ABORT), etc. The top 5 life cycle values with their frequencies are given in table 1.

REGISTERED_DONE_RAN_CLEARED	41.3 %
REGISTERED_ABORT	29.4 %
REGISTERED_DONE_RAN	22.8 %
REGISTERED_DONE	2.7 %
REGISTERED_ABORT_RAN	0.9 %

Table 1. Top 5 life cycle values with their frequencies

To give more information on the reason for the final state of a job (especially in case of error), the `final_reason` field provides a user readable message. Un-

fortunately, the set of possible values is larger, due to the diversity of cases that may occur but also to the different versions of middlewares, sometimes displaying different messages for the same reason. Combined with the `life cycle` field, we counted 315 different cases. Some `final_reason` fields were shortened to exclude non relevant specific information such as particular file name or site name appearing in the message. For instance, the text `"cannot retrieve previous matches for https://lcgrb01.gridpp.rl.ac.uk:9000/XCKb1dsA3fXbzsY7Q"` was replaced by `"cannot retrieve previous matches for"`.

RDRC	REGISTERED-DONE-RAN-CLEARED
RDR	REGISTERED-DONE-RAN
RA	REGISTERED-ABORT
RAC	REGISTERED-ABORT-CLEARED
RD	REGISTERED-DONE
UA	UNDEFINED-ABORT
Una	UNDEFINED-na
RE	REGISTERED-ENQUEUED
RnaR	REGISTERED-na-RAN
UDR	UNDEFINED-DONE-RAN
UDRC	UNDEFINED-DONE-RAN-CLEARED
RRR	REGISTERED-RUNNING-RAN
RT	REGISTERED-TRANSFER

Table 2. Abbreviations for some type values.

Before exploiting the data, some curation was needed for proper interpretation. Namely: data sources were selected when redundant information was available (LM and LRMS traces redundancy); specific text `final_reason` fields were truncated; and rare events were neglected in order to reduce the number of cases to analyze (an experimental justification if given in paragraph 5.3). As a result, table 3 details the 37 most frequent cases, representing 99.4% of the total data. This selection is a compromise between data completeness and number of cases to analyze. The last column of table 3 proposes a classification of the cases into 3 classes that are detailed below.

3.1   Successful jobs

The first class corresponds to jobs that have started running and either terminated successfully or were canceled by the user. We consider that these jobs were possibly successful job even if the intervention of the user changed the final status or if some produced files were not retrieved nor used. For these jobs, we denote by R the job latency, *i.e.* the time between the epoch of registration on the UI and the epoch where the job starts running. Due to some clock synchronization problems it may happen that a latency value R is negative: such entries have been excluded from the study. Of course, such problem may also alter some

case	type and final_reason	occurrences	%	class
1	RDRC Job terminated successfully	11,563,331	34.6%	R (9,999,928)
2	RDR Job terminated successfully	5,639,638	16.9%	R (5,035,776)
3	RA Job RetryCount (0) hit	3,838,380	11.5%	outlier
4	RA Cannot plan: BrokerHelper: no compa	3,422,319	10.2%	F
5	RDRC -	1,299,235	3.89%	R (1,138,324)
6	RDRC There were some warnings: some file	1,004,800	3.01%	R (884,932)
7	RDR There were some warnings: some file	911,500	2.73%	R (813,405)
8	RDR -	877,229	2.62%	R (750,473)
9	RD Aborted by user	863,094	2.58%	R (875,89)
10	RA Job RetryCount (3) hit	582,152	1.74%	outlier
11	RA -	557,055	1.67%	F
12	RA Job proxy is expired.	495,519	1.48%	F
13	RA cannot retrieve previous matches fo	322,839	0.97%	F
14	RAR Job proxy is expired.	267,890	0.80%	F
15	Una -	235,458	0.70%	R (10,632)
16	RDR Aborted by user	188,421	0.56%	R (15,3479)
17	RA Job RetryCount (1) hit	165,231	0.49%	outlier
18	UA Error during proxy renewal registra	149,095	0.45%	F
19	RA Unable to receive	115,867	0.35%	F
20	RE -	109,089	0.33%	F
21	RA Cannot plan: BrokerHelper: Problems	89,553	0.27%	F
22	UA Unable to receive	70,215	0.21%	F
23	RA Job RetryCount (2) hit	63,595	0.19%	outlier
24	RnaR -	56,044	0.17%	R (53,055)
25	RD -	45,400	0.14%	R (2,091)
26	RAC cannot retrieve previous matches fo	35,887	0.11%	F
27	UDR Job terminated successfully	31,722	0.09%	R (236)
28	RAR -	26,268	0.08%	F
29	RDRC There were some warnings: some outp	25,868	0.08%	R (20,341)
30	RDR There were some warnings: some outp	23,178	0.07%	R (18,315)
31	RRR -	22,983	0.07%	R (19561)
32	RA Submission to condor failed.	22341	0.07%	F
33	RA Job RetryCount (5) hit	22,260	0.07%	outlier
34	RT Job successfully submitted to Globu	18,972	0.06%	R (3,768)
35	RT unavailable	18,065	0.05%	F
36	RA Job RetryCount (7) hit	17,328	0.05%	outlier
37	RA hit job shallow retry count (0)	16,863	0.05%	outlier

Table 3. The 37 most frequent cases of type and final reason field values are totalizing 99.4% of the total data. Type values have been abbreviated for readability of the table, using table 2. The last column distinguishes correctly running jobs with latency (R with number of data entries remaining after cleaning), failed jobs (F) and outliers.

positive values. However, these events are rare and the synchronisation difference are small compared to the values considered.

It also happens that either LM or LRMS traces are available. Since LRMS values are more accurate, we decided to keep only data where LRMS values were available. The number of remaining data is given for each case in table 3 inside the parenthesis after the R symbol. This class is composed of 18,991,905 entries.

Figure 3 displays the distribution of latency values for all successful cases from table 3. We observe that all profiles are similar although the frequencies differ, and the first class represents most of the data. Figure 4 displays the probability density function (pdf) of the latency on top and its cumulative density function (cdf) on bottom. This laws are known to be heavy tailed [13] meaning that the tail is not exponentially bounded (see figure 5):

$$\forall \lambda > 0, \lim_{t \to \infty} e^{\lambda t}(1 - F_R(t)) = +\infty$$

3.2   Failed jobs

The second class corresponds to jobs that have failed for different reasons, leading to abortion by the WMS (no compatible resources, proxy error, BrokerHelper problem, CondorC submission failure...). Most jobs are aborted after a delay, denoted by the variable F, computed from the epoch of job registration until the done state epoch corresponding to the abortion instant. The delay F is one of the subjects of this study. Similarly to the previous class, some synchronization clock problems led to exclude some data. Moreover, the terminal "done" status may not be reached in some cases, as for example 18 and 22. We have decided to assume that the fault was immediately reported to the system in these cases. This class is finally composed of 5,607,329 entries.

The different fault latency profiles (F) for the different cases of table 3 labelled as faults are displayed in figure 6. Contrarily to the study of successful jobs, we observe that the profiles of the curves corresponding to each case conducting to fault are quite different. The corresponding pdf and cdf of F are plotted in figure 7. They corresponds approximately to the profile of case number 4 even if they have been computed on all failed jobs: case number 4 is predominant (10.2% of entries compared to second larger, case number 11 with 1.67% of entries).

3.3   Outliers

Jobs with type "REGISTERED-ABORT" and final reason "Job RetryCount (any number) hit" are jobs that have failed at least once at a site and been submitted to other sites until the user defined maximum number of retries is reached at which point the WMS gives up on the jobs. The WMS is aware of such failures either because it is notified of the job failure or because the job times out.
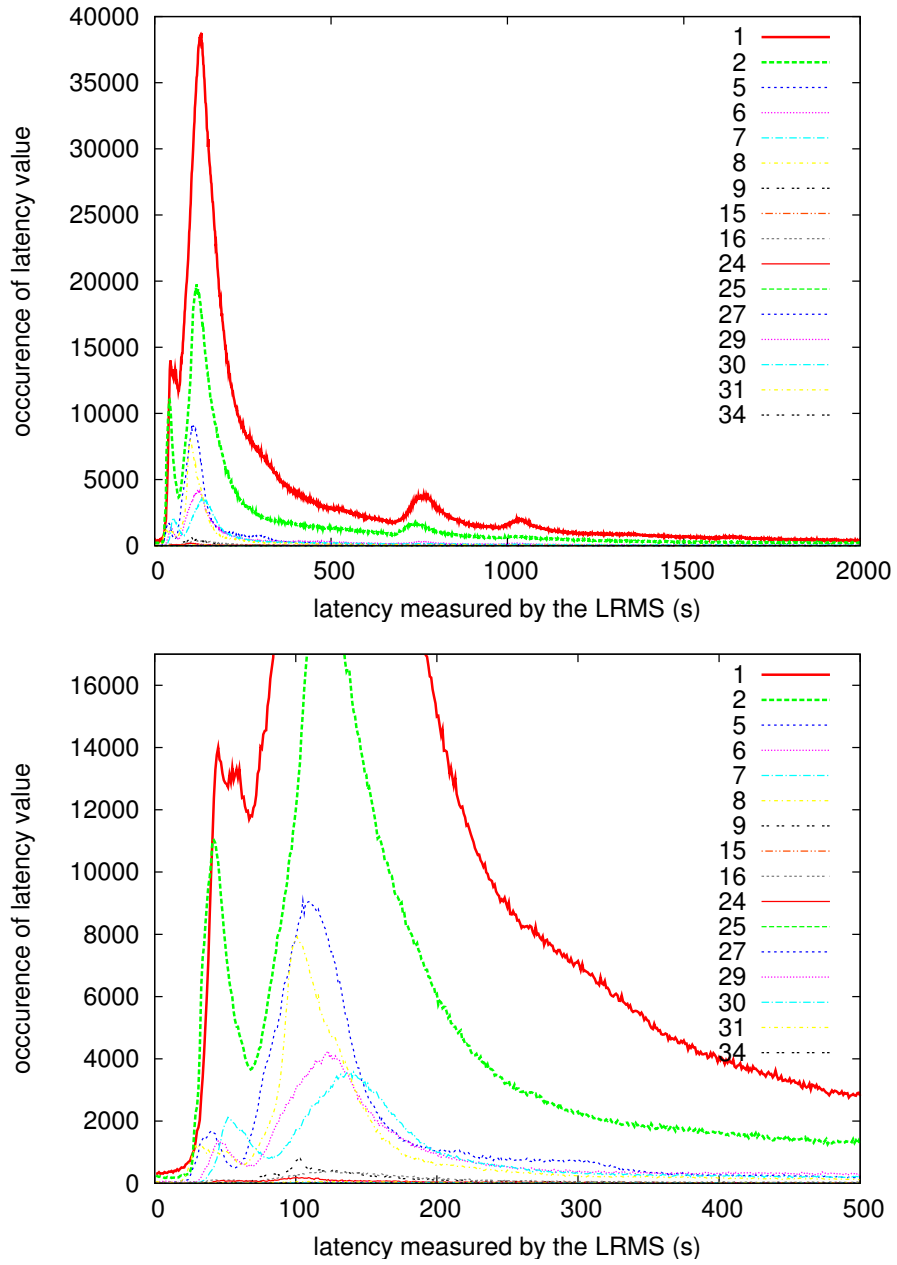
Fig. 3. Occurrences of latency values for different cases (see table 3) of successful jobs. The figure below gives more details for low values. The first two cases (1 and 2) are plotted thicker for an easier reading.
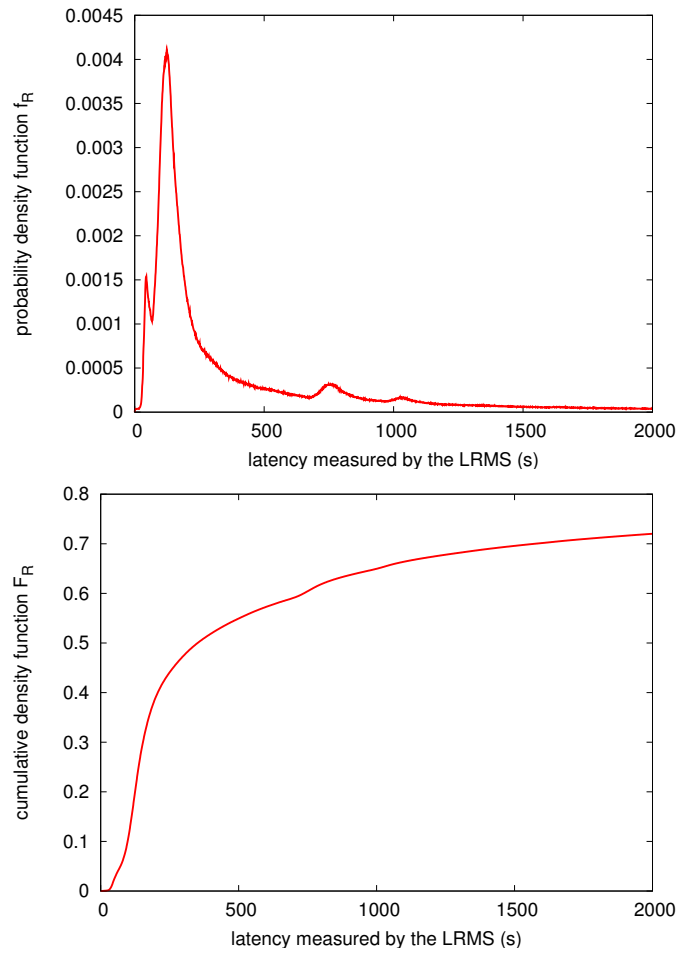
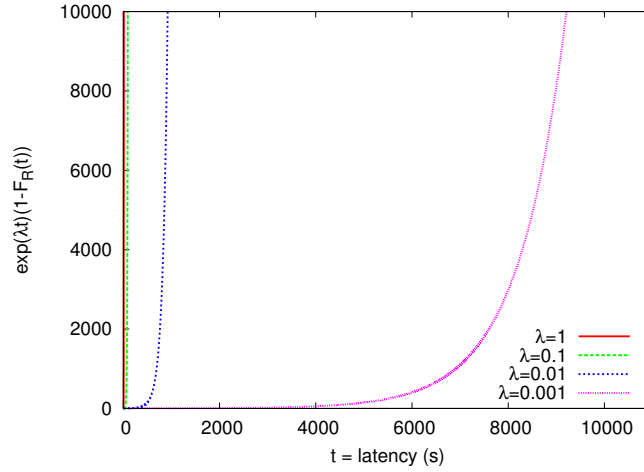**Fig. 4.** cdf (top) and pdf (bottom) of the latency in all the cases displayed in figure 3.

Fig. 5. Product $e^{\lambda t}(1 - F_R(t))$ for different values of λ. This illustrates that the distribution of latency is heavy tailed.

The final reason for a large part of these jobs is known after a very long delay (few 100000s seconds) when compared to other failed jobs. They correspond to jobs that never return due to some middleware failure or network interruption (jobs may have been sent to a CE that has been disconnected or crashed and the LB will never receive notification of the completion). They are usually detected using a timeout value by the WMS. This last class of jobs, labelled as outliers, contains 4,705,809 entries.

3.4   Summary

We denote as ρ the ratio of outliers and ϕ the ratio of faulty jobs. In the complete data set considered, we measure:

$$
\begin{aligned}
\text{outliers} \quad &: \quad \rho = 16.1\% \\
\text{faults} \quad &: \quad \phi = 19.1\% \\
\text{successful} &: 1 - \rho - \phi = 64.8\%
\end{aligned}
$$

When comparing the distribution of F to the one of R, we observe that, even if faults are not always known immediately, they are usually identified in a shorter time than the latency impacting most successful jobs. We will now study the impact of the delay before faults detection on the total latency of a job.
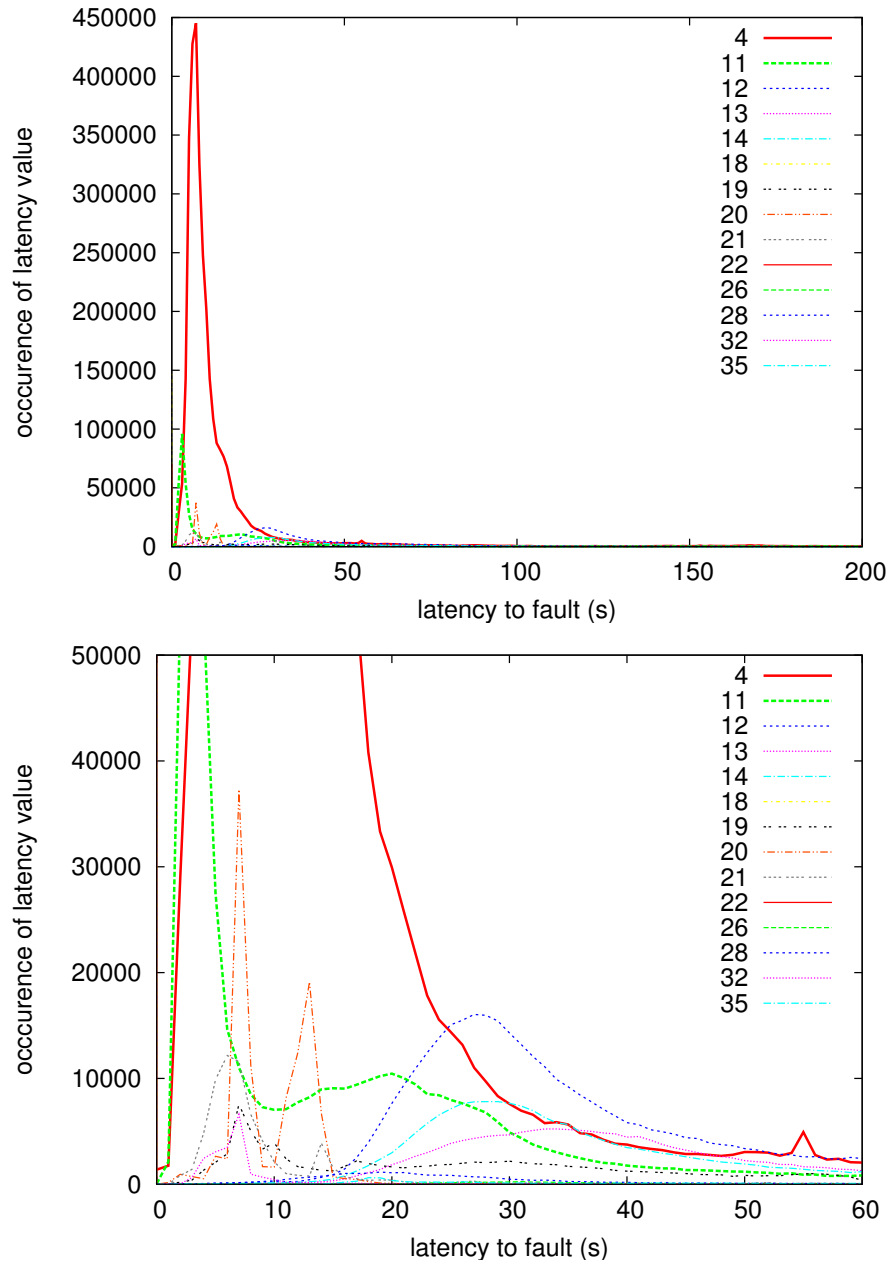
Fig. 6. Occurrences of latency to fault values for different cases (see table 3) and detail for low values. The first two cases (4 and 11) are plotted thicker for an easier reading.
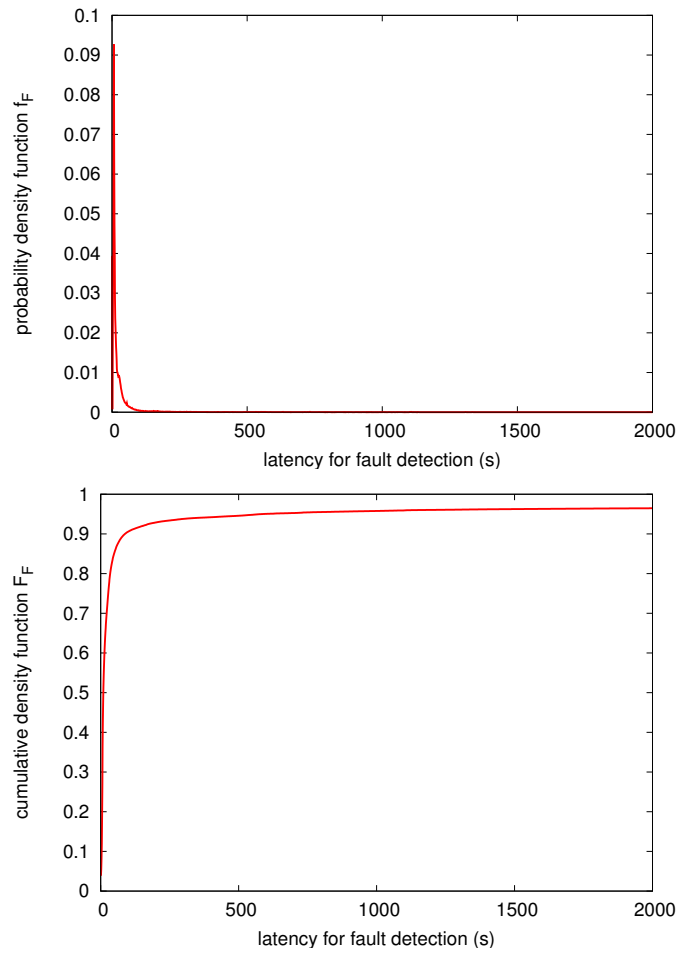
Fig. 7. pdf (top) and cdf (bottom) of the latency for fault detection in all the cases examined in figure 6.

4   Resubmission after fault

4.1   Probabilistic modeling

In the remainder, a capital letter X traditionally denotes a random variable with pdf f_X and cdf F_X. Let R denote the latency of a successful job and F denote the failure time. Assuming that faulty jobs are resubmitted without delay, let L denote the job latency taking into account the necessary resubmissions. L depends on the distribution of the jobs failure time. With ρ the ratio of outliers and ϕ the ratio of failed jobs, the probability, for a job to succeed is $(1 - \rho - \phi)$.

A job encounters a latency $L < t$, t being fixed, if it is not an outlier and either:

- the job does not fail (probability $(1-\rho-\phi)$) and its latency $R < t$ (probability $P(R < t) = F_R(t)$); or
- the job fails at $t_0 < t$ (probability $\phi f_F(t_0)$) and the job resubmitted encounters a latency $L < (t - t_0)$

The cumulative distribution of L is thus defined recursively by:

$$F_L(t) = (1 - \rho - \phi)F_R(t) + \phi \int_0^t f_F(t_0).F_L(t - t_0)dt_0$$

where the distributions of R and F are numerically estimated from the statistical data set described in the previous section. However, in this equation, the cdf F_L appears both in left and right sides. Moreover, its value at time t does appear in both terms.

In order to compute the cdf F_L, we discretize this equation with some considerations:

- No successful job has a null latency: $F_R(0) = 0$
- We introduce the second as the discretization step for the variable t. Indeed, in practice we know that we cannot have a higher precision than the second for our measurements. The discretization step is chosen accordingly.
- Some jobs are immediately known to fail (for example if the fault occurs on the client side). We thus consider $F_F(0) \neq 0$

Since no job has a null latency, this is also the case with resubmitted jobs: $F_L(0) = 0$. Supposing now $t > 1$, we get:

$$F_L(t) = (1 - \rho - \phi)F_R(t) + \phi \sum_{t_0=0}^{t-1} f_F(t_0)F_L(t - t_0)$$

This equation is resolved differently in the cases $t = 1$ and $t > 1$. For $t = 1$, it simplifies to:

$$F_L(1) = (1 - \rho - \phi)F_R(1) + \phi f_F(0)F_L(1) \Rightarrow F_L(1) = \frac{1 - \rho - \phi}{1 - \phi f_F(0)}F_R(1)$$

For $t > 1$, we can write:

$$F_L(t) = (1 - \rho - \phi)F_R(t) + \phi f_F(0)F_L(t) + \phi \sum_{t_0=1}^{t-1} f_F(t_0)F_L(t - t_0)$$

leading to:

$$F_L(t) = \frac{1}{1 - \phi f_F(0)}\left[(1 - \rho - \phi)F_R(t) + \phi \sum_{t_0=1}^{t-1} f_F(t_0)F_L(t - t_0)\right]$$

On the right side of this equation, the terms in F_L are in the form $F_L(u)$ with $u \in [1 \; ; \; (t - 1)]$. $F_L(t)$ can therefore be computed recursively. The complete formula is given by equation 1:

$$
\begin{vmatrix}
F_L(0) & = 0 \\
F_L(1) & = \dfrac{1 - \rho - \phi}{1 - \phi f_F(0)} F_R(1) \\
F_L(t > 1) & = \dfrac{1}{1 - \phi f_F(0)}\left[(1 - \rho - \phi)F_R(t) + \phi \sum_{u=1}^{t-1} f_F(t - u)F_L(u)\right]
\end{vmatrix}
\tag{1}
$$

4.2   Exploitation of the grid traces

Figure 8 displays the cdfs of several variables. F_R and F_F have been estimated from the grid traces data. Equation 1 enables to compute F_L, the cdf of successful jobs including resubmission in case of failures. We clearly observe the impact of failures in this latency L when compared to R. F_L's curve is lower: the probability of achieving a given latency when faults occur is thus lower. In order to see more precisely the impact of failures, we also plotted $(1 - \rho)F_R$ which corresponds to the outliers and the successful jobs, ignoring failed jobs. This last curve is slightly above F_L: while L displays a probability of 50% for jobs to have a latency lower than 761 seconds, it reduces to 719 seconds when ignoring failures (or the difference of probability is 1% for the same latency value).

Having established the distribution properties of L, we will now focus on the exploitation of the data for implementing a realistic resubmission strategy that aims at reducing the latency experienced by users.

5   Resubmission strategy

5.1   Modeling

As seen in the previous section, the probability for a job to terminate before a given instant t is given by $F_L(t)$. We consider the resubmission strategy developed in [13] where a job is canceled and resubmitted if its latency is higher than
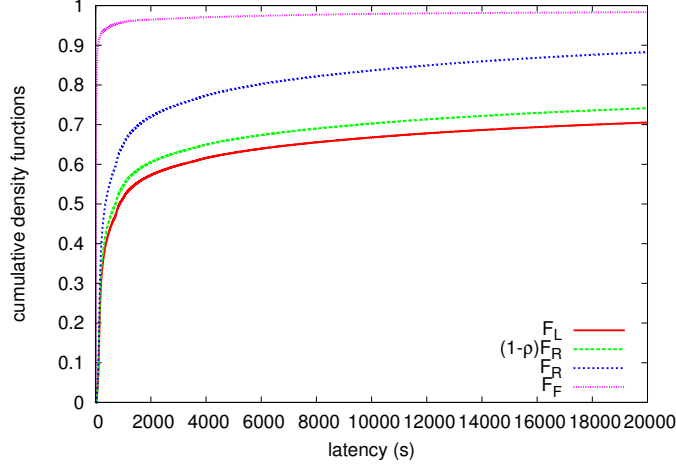
Fig. 8. cdfs: latency of fault detection (F_F), latency of successful jobs (F_R), latency of successful jobs using resubmission in case of failures (F_L). For comparison: $\tilde{F}_R = (1-\rho)F_R$.

a given timeout value t_∞ which value needs to be optimized. The work presented in [13] was based on probe jobs that neglected faults (they were excluded from the data) but some jobs did not return and were labelled as outliers. We denote $\tilde{F}_R(t)$ the probability for a job to face a latency lower than t. When neglecting faults, $\tilde{F}_R$ is related to the distribution of latency F_R and the ratio of outliers:

$$\tilde{F}_R(t) = (1 - \rho)F_R(t)$$

We denote J the total latency including resubmissions after waiting periods of t_∞. From [13], we can express the expected total latency E_J, considering resubmissions at t_∞ as:

$$E_J(t_\infty) = \frac{1}{\tilde{F}_R(t_\infty)} \int_0^{t_\infty} (1 - \tilde{F}_R(u))du \qquad (2)$$

Thanks to the more complete workload data studied in this paper, we can refine the model by taking the latency for fault detections into account. We thus consider the following resubmission strategy: jobs for which the latency L, including resubmissions due to failures, is greater than a timeout value t_∞ are canceled and resubmitted. Observing that $F_L(t)$ corresponds to the probability for a job to succeed with a latency lower than t, we can replace, in equation 2, $\tilde{F}_R$ by F_L:

$$E_J(t_\infty) = \frac{1}{F_L(t_\infty)} \int_0^{t_\infty} (1 - F_L(u))du \qquad (3)$$

Minimizing this equation leads to the estimation of the optimal timeout t_∞ value.

5.2   Experiments: taking into account faults in the model.

The profile of the expectation of the total latency, including all resubmissions and computed from equation 3 is plotted in figure 9. The curve reaches a minimum value $E_J = 584s$ for an optimal timeout value $t_\infty = 195s$.
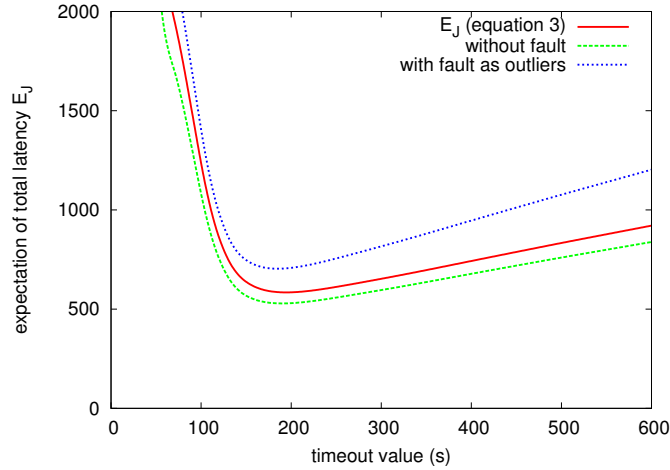

Fig. 9. Expectation of total latency with respect to timeout value t_∞. The first curve is obtained from equation 3. The result is compared with the case ignoring failures and the case where failures are accounted as outliers.

Two others profiles are plotted for comparison. The first one is the case ignoring the failures and corresponding to equation 2 with $\tilde{F}_R = (1 - \rho)F_R$. In that case, the minimum is reached at $t_\infty = 191s$, leading to $E_J = 529s$ which is under-evaluated.

The second comparison is performed with the assumption that failures can be considered as outliers, thus leading to a total of 35% of outliers. In this case, E_J reaches a minimum at $t_\infty = 185s$, which is underestimated and conducts to minimal value $E_J = 704s$, highly overestimated.

This experiment shows that taking into account a model of latency for faults detection has an influence on the parameters for this particular resubmission strategy.

5.3   Experiments: reducing the number of cases.

In section 3, we have retained the 37 most frequent cases, displayed in table 3. Here, results obtained with different number of most frequent cases are com-

pared, in order to measure the relevance of reducing the number of cases to be taken into account. Figure 10 presents the variation of E_J with respect to the timeout value t_∞ for different numbers of most frequent cases. The optimal values of t_∞ leading to minimal E_J values are given in table 4. We observe that reducing the number of cases from 37 to 30 does not impact the results of the resubmission strategy, showing that not taking care of all possible cases (315 cases) does not impact the final result, since we are considering the most frequent ones. However, reducing the number to 20 or less cases impacts the final result. In table 4, results concerning the model including faults and the previous model without including faults are displayed. These results shows that reducing the number of cases to less than 20 cases impacts more than not considering the faults in the model.
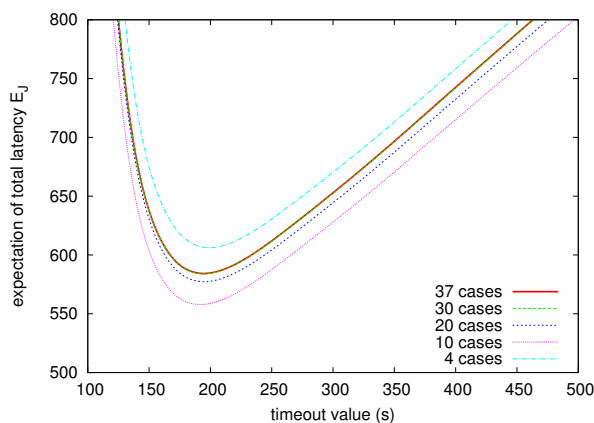

Fig. 10. Variations of the expected total latency (E_J) including resubmissions with respect to the timeout value, for different number of cases from table 3. We observe no visual difference between 37 and 30 cases. For less cases, we observe variations of E_J.

6 Conclusions and perspectives

Probabilistic models of the grid jobs latency enable us to capture the complex behavior of grid workload management systems. The model proposed in this paper relies on statistic collection of job execution traces in order to estimate the cdf of several parameters stochastically modeled. As compared to previous work, the model as been enriched to take into account normal operations, outliers and faults, which frequency is high on grids and therefore significantly impacts job execution time. The model is exploited to optimize a simple job resubmission strategy that aims at optimizing applications performance using objective information. The more jobs a grid application is composed with, the more it will be

nb. of	with faults (F_L)		without faults ($\tilde{F}_R$)	
cases	opt. t_∞	min. E_J	opt. t_∞	min. E_J
37	195s	584s	191s	529s
30	194s	584s	191s	529s
20	195s	577s	191s	524s
10	192s	558s	189s	530s
4	199s	606s	197s	570s

Table 4. Influence of the number of most frequent cases taken into account in the model on the estimation of optimal timeout value (t_∞) and minimal expectation of total latency including resubmission (E_J). Comparison of the results in two cases: with or without faults included in the model.

sensitive to such fault recovery procedures. In the future, more elaborate submission strategies commonly implemented on grids, such as multiple submission of a same task, will be considered.

This paper also emphasizes on the practical difficulties encountered when collecting and then exploiting traces on a large scale, heterogeneous production grid infrastructure. The set up of a Grid Observatory with well established procedures for traces collection, harmonization and curation is critical for the success of such grid behavioral analysis. It will allow to focus on modeling and experimentation without having to consider heavy-weight technical problems in the context of each new study. In addition, the Grid Observatory ensures dense data collection for accurate estimations without disturbing the normal grid operation.

The preliminary work detailed in this paper exploits a consistent but archived set of traces for *a posteriori* analysis. In the future, the Grid Observatory is expected to provide live information for tackling the non-stationarity of the grid workload manager and enabling relevant estimate of the grid running conditions.

Acknowledgments

References

1. Glatard, T., Montagnat, J., Pennec, X.: A probabilistic model to analyse workflow performance on production grids. In Priol, T., Lefevre, L., Buyya, R., eds.: IEEE

International Symposium on Cluster Computing and the Grid (CCGrid'08), Lyon, France, IEEE (May 2008) 510–517

2. Lingrand, D., Montagnat, J., Glatard, T.: Modeling user submission strategies on production grids. In: International Symposium on High Performance Distributed Computing (HPDC'09), Munchen, Germany (June 2009) (accepted)

3. Frachtenberg, E., Schwiegelshohn, U.: New Challenges of Parallel Job Scheduling. In: 13th Job Scheduling Strategies for Parallel Processing (JSSPP). Volume LNCS 4942., Springer (April 2008) 1–23

4. Medernach, E.: Workload Analysis of a Cluster in a Grid Environment. In: Job Scheduling Strategies for Parallel Processing (JSSPP), Cambridge, MA, USA (June 2005) 36–61

5. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.: The Grid Workloads Archive. Future Generation Computer Systems **24**(7) (July 2008) 672–686

6. Feitelson, D. In: Workload modeling for performance evaluation. Springer-Verlag - LNCS vol 2459 (September 2002) 114–141

7. Christodoulopoulos, K., Gkamas, V., Varvarigos, E.A.: Statistical Analysis and Modeling of Jobs in a Grid Environment. Journal of Grid Computing (JGC) **6**(1) (March 2008) 77–101

8. Germain, C., Loomis, C., Mościcki, J.T., Texier, R.: Scheduling for Responsive Grids. Journal of Grid Computing (JGC) **6**(1) (March 2008) 15–27

9. Li, H., Groep, D., Walters, L.: Workload Characteristics of a Multi-cluster Super-computer. In: Job Scheduling Strategies for Parallel Processing, Springer Verlag (2004) 176–193

10. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In: 10th IEEE International Conference on Computer Modeling and Simulation (UKSim), Cambridge, UK (April 2008) 126–131

11. Pacini, F.: WMS User's Guide. Technical Report EGEE-JRA1-TEC-572489, EGEE (May 2006)

12. Lingrand, D., Montagnat, J., Glatard, T.: Estimating the execution context for refining submission strategies on production grids. In: Assessing Models of Networks and Distributed Computing Platforms (ASSESS / ModernBio) (CCgrid'08), Lyon, IEEE (May 2008) 753 – 758

13. Glatard, T., Montagnat, J., Pennec, X.: Optimizing jobs timeouts on clusters and production grids. In: International Symposium on Cluster Computing and the Grid (CCGrid'07), Rio de Janeiro, IEEE (May 2007) 100–107