# Modeling Resubmission in Unreliable Grids: The Bottom-Up Approach

Vandy Berten[1] and Emmanuel Jeannot[2]

[1] Université Libre de Bruxelles
[2] INRIA, LORIA

**Abstract.** Failure is an ordinary characteristic of large-scale distributed environments. Resubmission is a general strategy employed to cope with failures in grids. Here, we analytically and experimentally study resubmission in the case of random brokering (jobs are dispatched to a computing elements with a probability proportional to its computing power). We compare two cases when jobs are resubmitted to the broker or to the computing element. Results show that resubmit to the broker is a better strategy. Our approach is different from most existing race-based one as it is a bottom-up one: we start from a simple model of a grid and derive its characteristics.

## 1 Introduction

Computational grids such as EGEE [6] or TeraGrid [13] are routinely used for executing scientific jobs. However, such environment are subject to failures. Indeed, a 9 months study [5] (from Feb. 2006 to Nov. 2006) of the SEE virtual organization of EGEE shows that only 30% of jobs finished with an `OK` status at the first try and 10% ultimately failed. Another study [10] covers the submission of 230 474 jobs in the the EGEE/LCG Grid during 280 days, among which 23 208 (9.93%) failed.

In order to tackle reliability problem, the main strategy commonly employed is to resubmit a failed job [3,1,8]. There are several ways to resubmit a job. It is possible to resubmit the job to the scheduler or to the computing element allocated to it.

To the best of our knowledge, comparing these two strategies in the context of computational grids has never been rigorously conducted. The goal of this work is therefore to model these strategies and to provide experimental insights on when and how a strategy is better than the other. We focus on a special kind of a grid environment directly inspired from EGEE. Jobs are submitted to a resource broker that dispatches them to a computing element where they are queued and treated according to their arrival date.

In general, scheduling algorithms that allocate jobs to resources assume that the arriving time and/or the duration of the jobs are known in advance. However, such an assumption is not always realistic (the duration of the job can only be known after its execution, and the arrival depends on the clients hence is not

H.X. Lin et al. (Eds): Euro-Par 2009 Workshops, LNCS 6043, pp. 150–161, 2010.

always deterministic). To cope with this uncertainty, we use a stochastic model where the job inter-arrival follows a Poisson law and the job duration follows an exponential law. Therefore, the resource brokering algorithm is a random one where each job is allocated to a computing element with a probability proportional to its accumulated speed. Thanks to this approach, very few assumptions are required, which leads to a general case that could easily be applied to production environments.

Moreover, to model the unreliability of the environment, we assume that each node has a probability of failure (*i.e* the probability that a job is not correctly executed on this node). Furthermore, to increase realism, we do not assume that this characteristic is known by the resource broker.

The contribution of our paper is the following. First, we model the behavior of the resubmission strategies. Second, we assess the quality of these models by comparing the predicted values with the real ones. We show that in almost every case our models are very precise. Last, we compare the strategies with regards to the execution time of a given set of jobs and we are able to determine when it is better to use a given strategy. It is important to note that our methodology is bottom-up one while most of the resent studies are top-down. This means that in this paper we start from a simple model of a grid and try to analytically compute the model while in the top-down approach, the model is derived from traces.

## 2 Related Work

Modeling a grid system (waiting time, throughput, failure) has already been studied in the literature. Concerning job duration and submission reference studies are [10] concerning EGEE, or [7,9] concerning different traces. Concerning modeling failures [12] covers high performance computing systems and [11], study the Condor system.

The common methodology of all these approaches is that they tackle the problem using a top-down approach. The top down-approach (or trace-based approach) consists in deriving the characteristics through an analysis of the traces. This approach is very useful to derive accurate understanding of a given setting but is limited when one change the target environment. In our approach (bottom-up) we start from a very general model of the environment and compute analytically a model of the behavior of this environment. This study is therefore an attempt to bridge the gap between the two approaches (trace-based/top-down vs. analytical/Bottom-up).

## 3 System, Job, and Resubmission Models

### 3.1 Grid Model

The model of computational grid we propose here is directly inspired from existing ones such as EGEE [6] and has already been presented in [2]. This system is

composed of $\mathcal{N}$ sites (or computing elements) $\mathbb{C}_i$. Each site $\mathbb{C}_i$ is assumed to be homogeneous: it is composed of $c_i$ homogeneous computing nodes of speed $s_i$. The *power* of a computing element is the product of the number of nodes times the speed of its node: $c_i \times s_i$.

A set of jobs (or tasks) is submitted to this grid. All jobs, as most of those submitted to EGEE, are sequential (*i.e.* executed by a single processor).

A job is submitted to a *resource broker* or *job dispatcher*. The broker uses a random strategy to dispatch jobs to the computing elements. Computing element $\mathbb{C}_i$ has a probability $\gamma_i = \frac{c_i \times s_i}{C}$ to be chosen, where $C = \sum_{i=1}^{\mathcal{N}} c_i \times s_i$ is the total power of the considered grid system. Therefore, jobs are submitted to a computing element with a probability proportional to its power. The intuition beyond this strategy is that the more powerful a computing element, the more jobs should be submitted to it. Moreover, it is important to note that this strategy is very simple to implement and does not require to estimate job duration.

We use a stochastic model for the submission date of jobs and their execution duration. The arrival of jobs to the resource broker follows a poissonian distribution with an inter-arrival rate $\lambda$. In this paper we focus on computational grids and therefore, we only consider coarse-grain jobs, where the run-time is dominating the whole execution time (*i.e.* network latencies and transfer times are neglected and we thus do not take into account the data localization). Hence, each submitted job $l$ is given a number of operations to perform $\mu_\ell$ and the execution time on computing element $\mathbb{C}_i$ is $\mu_\ell \times s_i$. The only assumption made about $\mu_\ell$ is that it follows an exponential distribution of parameter $\mu$. Each computing element is equipped with a FIFO queue where incoming jobs are stored and wait for a node to become available for execution.

A grid system is never perfectly reliable: job can fail due to several factors. To model that, each node of site $\mathbb{C}_i$ has a probability $p_i < 1$ to produce a faulty result. As computing this probability is not always easy or possible, the resource broker does not use this parameter when dispatching jobs to resources. Finally, we suppose that failures are transient: when a job is erroneous it is possible to use again the processors that has executed this job.

### 3.2 Fault-Tolerant Strategies

In order to cope with failures, we propose 2 fault tolerant strategies. **Global resubmission**: when a job has failed, it is resubmitted to the resource broker. The resource broker then chooses a new computing element using the same strategy as described above. **Local resubmission**: when a job has failed on a given computing element, it is resubmitted and queued to the same computing element.

We will consider two variants. One with a limited number of $R$ submissions, the other with an unbounded number of resubmissions ($R = +\infty$). The system stops resubmission when the job is correct or after $R$ submissions. When $R$ is finite, it is not guaranteed that the job will finally be correctly executed, while when $R$ is unbounded, it can take a very long time for the job to be correctly executed.

## 4 Analysis of the Strategies

We propose to study 3 metrics. The *saturation load* which is the load above which the system cannot treat all the incoming jobs (the queue sizes are growing with time). The *average waiting time* which is the time during which a job stays in the system. The last metric is the *failure probability* which gives the chance that this job is not correctly executed.

### 4.1 Unlimited Global Resubmission

In this section, we assume that a job can be resubmitted an unlimited number of times. This guarantees that, when it exits the system, a job is correct.

**Saturation Load.** Let $\nu = \frac{\lambda}{\mu C}$ be the input load (the input rate divided by the computational rate). Let $\alpha = \frac{1}{C}\sum_{i=1}^{N} c_i p_i s_i = \sum_{i=1}^{N} \gamma_i p_i$ be the probability that a job scheduled by the Resource Broker fails. Let $\delta_i$ be what exits $\mathbb{C}_i$. A part of this flow is sent back to the resource broker, and is then added to the input rate $\lambda$. We assume that the input flow plus the resubmitted jobs can be considered as a poissonian flow (this is an approximation, but we assume that if the input flow is (much) larger than the feedback, this assumption is reasonable). We denote by $\lambda_E$ the effective input rate, *i.e.* $\lambda$ plus the resubmissions.

We are here interested by non saturated systems ($\frac{\lambda_E}{\mu C} < 1$), which means that the input flow is equal to the output flow. Indeed, the system does not saturate when the output rate $\lambda_E$ is lower than the computational rate $\mu C$. With this approximation, we have $\delta_i(\text{output rate of } \mathbb{C}_i) = \lambda_E \frac{c_i s_i}{C}(\text{input rate of } \mathbb{C}_i)$ and $\lambda_E = \lambda + \sum_k \delta_k p_k = \lambda + \sum_k \lambda_E \frac{c_k s_k}{C} p_k = \lambda + \lambda_E \frac{1}{C}\sum_k c_k p_k s_k = \lambda + \lambda_E \alpha = \frac{\lambda}{1-\alpha}$.

Therefore, if the system input flow is $\nu = \frac{\lambda}{\mu C}$, we have an effective load of $\nu_E = \frac{\lambda_E}{\mu C} = \frac{\frac{\lambda}{1-\alpha}}{\mu C} = \frac{\nu}{1-\alpha}$. Hence, the system saturates for an input load of $\nu = \frac{\lambda}{\mu C} > 1 - \alpha$.

**Average traversal time of $\mathbb{C}_i$ ($F_i$).** For a load of $\nu_E$, we know [2] that the average queue size of computing element $\mathbb{C}_i$ is $Q_i(\nu_E) = \mathbb{E}[Q_i] = b\frac{{\nu_E}^{c_i+1} \cdot {c_i}^{c_i}}{c_i!(1-\nu_E)^2}$ where $b = \left[\sum_{k=0}^{c_i-1} \frac{(\nu_E c_i)^k}{k!} + \frac{(\nu_E c_i)^{c_i}}{c_i!}\frac{1}{1-\nu_E}\right]^{-1}$. On computing element $\mathbb{C}_i$, the average time before the end of an execution (called $F_i$), with an effective load of $\nu_E$ on $\mathbb{C}_i$ is then $\frac{Q_i(\nu_E)}{\mu_i c_i} + \mu_i^{-1}$. Hence, if the input load is $\nu$, $F_i(\nu) = \frac{Q_i(\frac{\nu}{1-\alpha}) + c_i}{\mu_i c_i}$. Let $\mathcal{F}$ be the average traversal time. We then have $\mathcal{F} = \sum_{i=1}^{N} \gamma_i F_i$, because $\gamma_i$ is the probability for a job to be sent on $\mathbb{C}_i$. Hence, $\mathcal{F} = \frac{1}{C}\sum_{i=1}^{N} c_i s_i F_i = \frac{1}{C}\sum_{i=1}^{N} \frac{c_i s_i}{\mu_i c_i}(Q_i(\frac{\nu}{1-\alpha}) + c_i) = \frac{1}{\mu C}\left[\sum_{i=1}^{N}\left(Q_i(\frac{\nu}{1-\alpha}) + c_i\right)\right]$.

**Average waiting time.** Let $W$ be the average time before a job exits the system, and $w_i$ the average waiting time for a job starting running on $\mathbb{C}_i$ (whatever the number of rounds). We have: $W = \sum_{i=1}^{N} \gamma_i w_i$ because $\gamma_i$ is the probability for a job to be sent on $\mathbb{C}_i$, and $w_i = F_i + p_i W$ because the average waiting time

can be split in the average waiting time on $\mathbb{C}_i$ ($F_i$), plus, if the job failed (with a probability $p_i$), the same waiting time than a job entering the system ($W$). We then have $W = \sum_{i=1}^{N} \gamma_i F_i + \sum_{i=1}^{N} \gamma_i p_i W = \mathcal{F} + \alpha W = \frac{1}{1-\alpha}\mathcal{F}$.

### 4.2 Limited Global Resubmission

In this section, we add the constraint that a job cannot be executed more times than some value $R$ given by the system.

**Saturation load.** We first compute the probability for a job to fail. If a job failed, it means that it failed at each rounds, included the $R^{\text{th}}$. We then have: $\mathbb{P}[\text{failure}] = \mathbb{P}[\text{failure at the first round} \wedge \cdots \wedge \text{failure at the } R^{\text{th}} \text{ round}] = \prod_{i=1}^{R} \mathbb{P}[\text{failure at the } i^{\text{th}} \text{ round}] = \prod_{i=1}^{R} \alpha = \alpha^R$. The probability for a job to exit the system at the end of its execution on $\mathbb{C}_i$ is the probability either to succeed $(1-p_i)$, or to have already $R-1$ failures before the current round $(\alpha^{R-1}p_i)$. The proportion of jobs exiting $\mathbb{C}_i$ but sent back to the input is then $p_i(1-\alpha^{R-1})$. We can now compute the input flow: $\lambda_E = \lambda + \sum_{k=1}^{N} \delta_k p_k (1-\alpha^{R-1}) = \lambda + (1-\alpha^{R-1}) \sum_{k=1}^{N} \lambda_E \gamma_k p_k = \lambda + (1-\alpha^{R-1})\lambda_E \alpha = \frac{\lambda}{1-(1-\alpha^{R-1})\alpha} = \frac{\lambda}{1-\alpha+\alpha^R}$.

Therefore, we can show that if the input load is $\nu$, we have an effective load $\nu_E$ of $\frac{\nu}{1-\alpha+\alpha^R}$. According to this, the system will then saturate at a load of $1-\alpha+\alpha^R$.

**Average traversal time on $\mathbb{C}_i$ ($F_i$).** With the same kind of arguments and notation as for the limited resubmission, we have $F_i = \frac{Q_i(\nu_E)}{\mu_i c_i} + \mu_i^{-1}$. Therefore, if the input load is $\nu$, $F_i(\nu) = \frac{Q_i\left(\frac{\nu}{1-\alpha+\alpha^R}\right)}{\mu_i c_i} + \mu_i^{-1}$. We then have $\mathcal{F} = \frac{1}{\mu C}\left[\sum_{i=1}^{N}\left(Q_i\left(\frac{\nu}{1-\alpha+\alpha^R}\right) + c_i\right)\right]$.

**Waiting time.** We will use the following notations: $W_k$ is the average time that a job entering the system takes to exit the system, if it can still be resubmitted $k$ times ($W_0$ is then the average run-time of a job which exits the system at the end of its execution, even if it fails). $w_{k,i}$ is the average time a job entering $\mathbb{C}_i$ takes to exit the system, if it can still be resubmitted $k$ times. Hence, when $R$ submissions are allowed $W_{R-1}$ is the average time for executing a job.

**Lemma 1.** *$W_{R-1} = \frac{1-\alpha^R}{1-\alpha}\mathcal{F}$ where $\mathcal{F}$ is defined in the above section.*

Due to lack of space, the proof is not given. Note that we have $\lim_{k\to\infty} W_k = W$, where $W$ is the average waiting time for unlimited resubmission.

### 4.3 Unlimited Local Resubmission

In this scenario, a failed job is re-submitted in the queue of the computing element where it was running. We use the same notation ($\lambda_i = \lambda\gamma_i$, $\lambda_i'$ and $\delta_i$) as in Section 4.1.

**Saturation load.** In order to evaluate the saturation load, we assume that the feedback flow ($\delta_i p_i$) is poissonian. We have $\lambda_i' = \delta_i$ (the flow coming in the queue

equals the flow going out), and $\lambda_i + \delta_i p_i = \delta_i$. Therefore, $\lambda'_i = \delta_i = \frac{\lambda_i}{1-p_i}$. The saturation load of $\mathbb{C}_i$ is then $1 - p_i$. And the saturation load of the system is then $\min(1 - p_i) = 1 - \max p_i$.

**Average traversal time on $\mathbb{C}_i$ ($F_i$).** With the same argument as for the global resubmission, we get $F_i(\nu) = \frac{Q_i(\frac{\nu}{1-p_i})+c_i}{\mu_i c_i}$.

We can see that the traversal time of the queue $i$ is only influenced by its own failure probability, while in the global case, this traversal time depends upon every $p_k$ (in $\alpha$).

**Average waiting time.** We use the $W$ and $w_i$ definition from global resubmission. We have $W = \sum_{i=1}^{N} \gamma_i w_i$ and $w_i = F_i + p_i w_i = \frac{F_i}{1-p_i}$.

Therefore, $W = \sum_{i=1}^{N} \gamma_i \frac{F_i}{1-p_i} = \sum_{i=1}^{N} \frac{c_i s_i}{C} \frac{Q_i(\frac{\nu}{1-p_i})+c_i}{\mu s_i c_i (1-p_i)} = \frac{1}{\mu C} \sum_{i=1}^{N} \frac{1}{1-p_i} \left( Q_i(\frac{\nu}{1-p_i}) + c_i \right)$.

### 4.4 Limited Local Resubmission

We can use the same argument as in the global case: the failure probability on $\mathbb{C}_i$ is ${p_i}^R$. The failure probability for a job entering the system is $\sum_{i=1}^{N} \gamma_i {p_i}^R$.

**Effective load.** The probability that a job exits the system is the probability that it was correct, or had already too many rounds: $\mathbb{P}[\text{resubmission}] = 1 - \mathbb{P}[\text{exit}] = 1 - \mathbb{P}[\text{succeed} \vee \text{final failure}] = 1 - ((1 - p_i) + {p_i}^R) = p_i(1 - {p_i}^{R-1})$.

Then the input flow of $\mathbb{C}_i$ is $\lambda'_i = \lambda_i + \delta_i p_i (1 - {p_i}^{R-1}) = \lambda_i + \lambda'_i p_i (1 - {p_i}^{R-1}) = \frac{\lambda_i}{1-p_i+{p_i}^R}$.

**Saturation load.** We can then get that the saturation load on $\mathbb{C}_i$ is $1 - p_i + {p_i}^R$. Then, the system one is $\min(1 - p_i + {p_i}^R) = 1 - \max(p_i - p_i^R)$.

For the Average traversal time of $\mathbb{C}_i$ ($F_i$), it is straightforward that $F_i(\nu) = \frac{Q_i\left(\frac{\nu}{1-p_i+{p_i}^R}\right)+c_i}{\mu_i c_i}$.

**Average waiting time**

**Lemma 2.** *Using the same notation as for the global resubmission we have:* $W_{R-1} = \frac{1}{\mu C} \sum_{i=1}^{N} \frac{1-{p_i}^R}{1-p_i} \left( Q_i \left( \frac{\nu}{1-p_i+{p_i}^R} \right) + c_i \right)$.

Due to lack of space, the proof is not given. Here again we have $\lim_{k\to\infty} W_k = W$, where $W$ is the average waiting time for unlimited local resubmission.

## 5 Experimental Validation

### 5.1 Experimental Settings

We have used the SimGRID simulator [4] to perform a set of experiments for measuring the different metrics proposed in the above sections as well as the total

running time for the execution of a given number of jobs. For these simulations, the unit of time is set as the average job run-time and hence $\mu = 1$. We have designed 6 platforms where we have executed 1000, 4000, 7000 and 10000 jobs. A sum-up of the six platforms is shown in Table 1. Platforms characteristics vary in number of clusters, number of nodes per cluster, speed of the nodes and failure probability of each nodes. The load submitted to each platform ($\lambda$) is set such as saturated and non-saturated modes are both observed. For each platform we use three *failure probability factors* of 1, 0.1 and 0.01. This is a multiplication factor that is applied to the $p_i$ value given in Table 1. The goal of these factors to obtain 3 different variants from low reliability (probability factor set to 0.1) to high reliability (probability factor set to 1). For instance, with a probability factor of 0.1 the probability failure of the 4 computing elements of platform 4 switches from (0.7,0.5,0.3,0.1) to (0.07,0.05,0.03,0.01)).

**Table 1.** Description of the different platforms used for the experiments

| Platform id | Nb clusters | Nodes per cluster $c_i$ | Node speed $s_i$ | Node proba of failure: $p_i$ | $\lambda$ min:inc:max |
|---|---|---|---|---|---|
| 1 | 3 | (3,2,5) | (1,3,2) | (0.5,0.5,0.5) | 1:0.5:15 |
| 2 | 6 | (20,30,10,30,3,50) | (0.1,0.1,0.2,0.1,2,0.05) | (0.5,0.4,0.5,0.2,0.9,0.1) | 1:0.5:25 |
| 3 | 4 | (6,6,6,6) | (0.1,0.3,0.5,0.7) | (0.1,0.3,0.5,0.7) | 1:0.5:25 |
| 4 | 4 | (6,6,6,6) | (0.1,0.3,0.5,0.7) | (0.7,0.5,0.3,0.1) | 1:0.5:25 |
| 5 | 4 | (6,6,6,6) | (0.1,0.3,0.5,0.7) | (0.9,0.9,0.01,0.01) | 1:0.5:25 |
| 6 | 4 | (6,6,6,6) | (0.4,0.4,0.4,0.4) | (0.99,0.99,0.01,0.01) | 1:0.5:25 |

We have set $R$, the maximum number of submission from 1 (no resubmission) to 10 and to infinity. The experiments were done by executing each setting (number of jobs, platform id, different load and 3 probability factors) 50 times to obtain relevant average values. At the end, a total of more than 5 millions experiments have been run.

### 5.2 Model Validation

For each experiments, We have measured the different metrics. Here, we compare our models with the experimental measures to assess their quality.

**Saturation.** When a system is not saturated, the measured average waiting time of job does not depend on the number of submitted jobs. However, when a system is saturated, the input load exceeds its treatment capacities, therefore, the average waiting time increases with the number of submitted job.

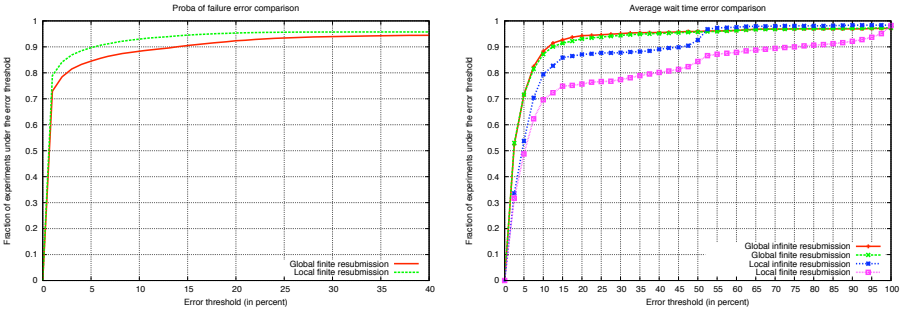
For each tuple (probability factor, heuristic, platform, number of copies/ maximum resubmission, $\lambda$) we have measured how the average queue size varies when the number of job increases from 1000 to 10000. This increase is measured by the slope of a linear interpolation. When this slope is smaller than a threshold, there is no saturation and when this slope is greater than a threshold there is saturation. By binary search we have found that the best discriminating

threshold is about 0.001 (for instance the average waiting time is 10 for a 1000 jobs and 19 for 10000 jobs). A good news is that this value does not depend on the considered heuristic (local resubmission or global resubmission). More precisely, for global resubmission, only 0.14% of the cases have a slope greater than 0.001 while considered by the model as non saturated cases and 0.7% have a slope lower than 0.001 while considered by the model as saturated cases. For local resubmission the percentage of error are respectively 0.6% and 3.9%.

From the above results, we see that models for resubmission are very accurate (local resubmission being a little bit less accurate due to some simplification hypothesis made).

**Probability of failure.** The accuracy of our model for determining the probability of failure is shown in Figure 1(a). For each case and each heuristic, we have compared the probability of failure given by the model ($pf1$) and the fraction of failure we have measured during our simulation ($pf2$). The error $e$ of the model is computed as follows $e = 100\frac{|pf2-pf1|}{pf2}$. In the graph of Fig. 1(a), we plot an error threshold on the x-axis and the fraction of cases that have an error lower than this threshold on the y-axis. Hence, such graph is similar to a cumulative distribution function (CDF) as for any value on the x-axis, we plot the fraction of experiments that have an error lower than a given value.

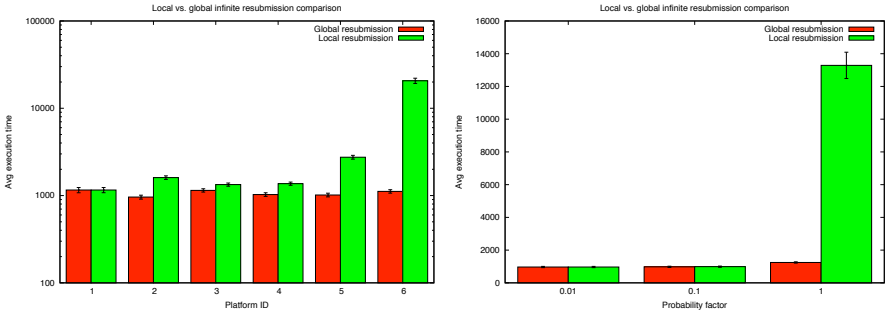
Note that unlimited resubmission is not shown here as the failure probability is 0. From this figure we see that more than 80% of the cases have an error lower than 5%. We see that the model for global resubmission is the least accurate though still very good (more than 90% of the cases have an error lower than 15%).




(a) Failure probability (b) Average wait time

**Fig. 1.** Accuracy of the model

**Average waiting time.** To show the accuracy of the model concerning average waiting time, we show the same kind of graph as above: in Fig. 1(b), we plot the error threshold on the x-axis and, on the y-axis, the fraction of experiments that have an average waiting time error lower than this fraction. The error being computed as $e = 100\frac{|wt2-wt1|}{wt2}$, where $wt1$ is the model prediction and $wt2$ is the measured value.




(a) Platform view (b) Probability factor view

**Fig. 2.** Comparison of the unlimited local vs. global resubmission

We see that the global resubmission model (both for the unlimited or limited case) is very accurate, almost 90% of the experiments have an error lower than 10%. The local resubmission model is less accurate but the unlimited case is still very good. The model for the local limited resubmission is acceptable with half of the cases having an error lower than 5% and 80% of the cases having an error lower than 40%.

### 5.3 Comparison of the Heuristics

**Local vs. Global Unlimited Resubmission.** Here we compare the unlimited resubmission heuristics (jobs are resubmitted to the system until they are successfully executed). Therefore, the probability of success is 1.

In Fig. 2(a) we show the average run-time of both heuristic for the 6 different considered platforms. We see that the local strategy never outperforms the global one. The two strategies provide similar results when no computing element is highly unreliable (platform 1 and to a less extent platforms 3 and 4). When the highest unreliable computing element has a probability of failure close to 0.9 (platform 2 and 5) the global strategy is better than the local one. When a platform has a highly unreliable computing element (such as in platform 6) the global heuristic greatly outperforms the local one.

In Fig. 2(b) we show the average run-time of both strategies when varying the *probability factor*, which is multiplicative factor of the given probability of failure of each computing element of the platform. From that figure, we see that when the probability factor is low (*i.e.* platforms are reliable), the local strategy matches the global one. However, when the probability of failure increases (probability factor of 1), the local strategy is outperformed by an average factor of 10.

We see that the local resubmission outperforms the global resubmission in 2.6% of the cases when the probability factor is 1 (platform are highly unreliable), in 36.6% of the cases when the probability factor is 0.1 and 49.1% of the cases when the probability factor is 0.01 (the platform is fairly reliable). This confirms

that the less reliable the platform the less efficient the local resubmission. In any cases, the local resubmission never outperforms the global resubmission with a ratio better than 1.08. This means that when local resubmission is better than global resubmission this is only by a very small margin.

Global resubmission is better than the local resubmission is explained by the fact that, if a job fails, this means that the processor that has executed this job is not reliable and therefore it is better to not reuse it for further execution and hence, it is better to resubmit the job globally to the system.

**Local vs. Global limited Resubmission.** Here, we compare both approaches when only limited resubmission is possible. When the maximum number of resubmission is allowed, the probability of success is lower than 1 and is different if one use the local or global submission. We already know that when probability of success is one (unlimited resubmission), the global strategy is better than the local strategy. Hence, here, we fix the maximum probability of failure and see if the local strategy can, sometimes outperform the global one. To do so, for each possible combination (platform, input load, and probability factor), we have fixed the success threshold to the one obtained by the global strategy when *one resubmission* is allowed (*i.e.* $R = 2$ and the exact success probability is $\alpha^2$). From Table 2, we see that, in more than 90% of the cases the local resubmission needs more than 2 resubmissions to exceed this success threshold of the global resubmission when $R = 2$. Hence, the local resubmission needs more resubmission to achieve the same reliability. Moreover, when the success threshold is exceeded by the local strategy, it leads to a better throughput in only 31% of the cases. But, in these cases, when we compare the throughput of the local strategy to the one of the global strategy we see that the improvement is only marginal (at most 4.3%) (detailed results for all platforms are given in table 2). This means that the local strategy sometimes requires a lot of resubmissions to match the reliability of the global strategy. When it does so, it is at the cost of a lower throughput in general and in any case, the local strategy is almost never able to outperform both the reliability and the throughput of the global strategy. Our explanation comes from the fact that, when a job has failed, it requires, on the average, a lot more local resubmissions than global ones to be successfully executed.

**Table 2.** Performance of the local strategy when asking to exceed the reliability obtained by the global strategy when $R = 2$

| Platform id | Total cases | Percentage of cases requiring more than 2 resubmissions to exceed the reliability of the global strategy | Percentage of cases with better throughput | Overall best throughput ratio |
|---|---|---|---|---|
| 1 | 87 | 48.3% | 48% | 0.5% |
| 2 | 147 | 93.9% | 35% | 4.3% |
| 3 | 147 | 91.2% | 54% | 2.1% |
| 4 | 147 | 93.2% | 11% | 0.7% |
| 5 | 147 | 99.3% | 16% | 1.2% |
| 6 | 147 | 100% | 22% | 0.5% |
| Overall | 822 | 90.5% | 31% | 4.3% |

## 6 Conclusion

Failure is an ordinary characteristic of large-scale distributed environments. In this paper we have studied the problem of random brokering on unreliable platforms directly inspired from existing grids such as EGEE. We propose a different approach from the usual trace-based (top-down) where the model is analytically computed from a general model of the environment and the submission strategy.

Here, our bottom-up approach is based on a simple model where incoming jobs are randomly dispatched to computational elements with a probability proportional to the accumulated speed of this element. As we assume that job execution can fail, we study two strategies to improve the reliability (namely local resubmission and global resubmission). For each heuristic we are able to model the saturation ratio (when the incoming load exceed the maximum throughput of the environment), the average waiting time of the jobs and the probability of success of each job. Our experiments show that the proposed models are very realistic. For each of the above metric the models usually predict a very precise value. Furthermore, experiments show that, on the average, the global resubmission is the best strategy as it outperforms, in almost every case, the local resubmission.

Future works are directed towards the evaluation of more metrics such as resource usage, load balance, etc. An other direction of future research is to improve the submission model by adding new laws for inter-arrival or duration or by mixing laws (*i.e.*. some job durations follow an exponential law while others follow a Weibull). The ultimate goal is to be able to come-up with analytical model that would match real traces using this approach.

## References

1. Angskun, T., Fagg, G., Bosilca, G., Pjesivac-Grbovic, J., Dongarra, J.: Scalable Fault Tolerant Protocol for Parallel Runtime Environments. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 141–149. Springer, Heidelberg (2006)
2. Berten, V., Goossens, J., Jeannot, E.: On the Distribution of Sequential Jobs in Random Brokering For Heterogeneous Computational Grids. IEEE Transactions on Parallel and Distributed Systems 17(2), 113–124 (2006)
3. Bouteiller, A., Herault, T., Krawezik, G., Lemarinier, P., Cappello, F.: Mpich-v: a multiprotocol fault tolerant mpi. International Journal of High Performance Computing and Applications (2005)
4. 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 (March 2008)
5. Costa, G.D., Dikaiakos, M., Orlando, S.: Analyzing the workload of the south-east federation of the egee grid infrastructure. Tech. Rep. TR-0063, Institute on Knowledge and Data Management, CoreGRID - Network of Excellence (February 2007),
`http://www.coregrid.net/mambo/images/stories/TechnicalReports tr-0063.pdf`

6. Enabling Grids for E-sciencE (EGEE), http://www.eu-egee.org/
7. Iosup, A., Dumitrescu, C., Dick, H.J., Epema, H.L., Wolters, L.: How are real grids used? the analysis of four grid traces and its implications. In: GRID 2006, pp. 262–269 (2006)
8. Jensen, H.T., Leth, J.R.: Automatic Job Resubmission in the Nordugrid Middleware. Tech. rep., Aalborg University (2004), http://www.nordugrid.org/documents/jensen_leth.pdf
9. Li, H., Heusdens, R., Muskulus, M., Wolters, L.: Analysis and synthesis of pseudo-periodic job arrivals in grids: A matching pursuit approach. In: CCGRID 2007, pp. 183–196 (2007)
10. Medernach, E.: Workload analysis of a cluster in a grid environment. In: Job scheduling strategies for parallel processing, pp. 36–61 (2005)
11. Rood, B., Lewis, M.J.: Multi-state grid resource availability characterization. In: GRID 2007, pp. 42–49 (2007)
12. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. In: DSN 2006, pp. 249–258 (2006)
13. TeraGrid, http://www.teragrid.org/