

A probabilistic model to analyse workflow performance on production grids

Tristan Glatard¹
UNSA / CNRS / I3S, INRIA / Asclepios
EPU, 930 route des Colles
06903 Sophia Antipolis, France

Johan Montagnat
UNSA / CNRS / I3S
EPU, 930 route des Colles
06903 Sophia Antipolis, France

Xavier Pennec
INRIA / Asclepios
2004 route des Lucioles
06903 Sophia Antipolis, France

Abstract—Production grids are complex and highly variable systems whose behavior is not well understood and difficult to anticipate. The goal of this study is to estimate the impact of the variability of those infrastructures on the performance of workflow-based applications. A probabilistic model of workflows execution time is proposed and evaluated. Results show that the variability of the EGEE grid infrastructure impacts the execution time of a particular medical image analysis application by a factor 2. The model gives interesting insights on the grid behavior for different application parallelization modes.

I. PERFORMANCE ANALYSIS ON PRODUCTION GRIDS

In many scientific areas, applications with stringent requirements for high performance computing, large data sets analysis and complex computation flows have emerged. Pushed by these new computational challenges very large scale production grids infrastructures have been deployed world-wide. Such widely distributed systems have been operating 24/7 over several years now, providing a sustained high end computing facility that many applications exploit routinely. The experience gained exploiting these systems shows that they can hardly be compared to traditional clusters performing on local area networks. For instance, we showed in a previous work that setting a timeout value to the jobs is mandatory on production grids whereas it is useless on most clusters [1]. Such differences may come from various factors. First, the reliability and homogeneity of clusters and local networks cannot be assumed on grids. Second, grids face very variable load patterns and race conditions originating from the shared exploitation by large user communities. Finally, the heterogeneity and the volatility of grid resources further increases the variability.

Consequently, production grids exhibit hard to predict behaviors that result in variable overheads imposed to the computations from the users point of view. For instance, we observed that over thousands of computation tasks submitted to the EGEE production grid¹ in the same experimental conditions during months, an average delay of approximately 5 minutes with a standard deviation of the same order of magnitude (5 minutes) is experienced. For grid applications requiring the submission of a very large number of short (less than 1 hour long) jobs in parallel, such overheads are far from being negligible. As a result, applications computation time (makespans)

are hardly forecastable, which makes performance analysis on production grids very difficult. In particular, the impact of the *variability* of the platform on the application should be quantified, as some works already suggested that it may have a strong negative impact on the applications [2].

The objective of this paper is to propose a grid application makespan model that (i) aims at explaining the performance of applications on production grids, (ii) allows to study the impact of grid variability on applications and (iii) can be used in the future for optimization. Workflow-based loosely coupled applications that performs well on production grid infrastructures are considered. The model is validated on the EGEE grid in real conditions and results are illustrated using a scientific application dedicated to medical image analysis.

A. Probabilistic modeling

Due to their scale, heterogeneity and overall middleware complexity, production grids are evolving systems difficult to model or even to simulate using fine grain approaches. In production conditions, the middleware characteristics are not well known: scheduling policies are let to the responsibility of local system administrators, middleware parameters and versions may differ from one site to another one. Furthermore, grids typically experience variations in the number of resources available (new resources provision, system failures or network interruptions...). Thus, to tackle this complexity, we introduced a probabilistic model of the latency imposed to single jobs in [1]. A probabilistic approach provides a black-box model which has been successfully applied in many scientific areas to model complex systems [3], [4], [5], [6]. In this paper, we propose to model the makespan of workflow-based applications which are representative of a large class of scientific grid applications. The complexity of the system will be considered as an alea: R will denote the random variable associated to the grid overhead and Σ will be the makespan of the application workflow.

B. Parallel execution of workflows

Workflow-based applications are loosely coupled applications composed by computation tasks with limited dependencies and typically expressed using a graph ordering a group of computing services. Workflows have raised a lot of attention in the grid community over the last years as they provide a

¹Now affiliated to the University of Amsterdam

²Enabling Grids for E-science, <http://www.eu-egee.org>

simple and flexible framework for reusing existing codes and expressing parallel applications [7], [8].

In the remainder of this paper we consider without restriction that the target application is represented as a graph of functions (or services) whose input/output parameters (or ports) are linked with dependencies (see figure 1 for an example). Such a flow of services is defined independently from the data sets to process. It should be noted that this model differs from Directed Acyclic Graphs (DAGs) of tasks that are commonly used on grids. Still, our model remains valid as DAGs can be viewed as the instantiation of a flow of services over the data.

Each service in the workflow is executed on the grid and its execution time is impacted by the grid latency. Moreover, each service is potentially invoked many times depending on the number of data segments that need to be processed for the application. The goal of a grid workflow engine is to optimize the use of grid resources to execute the application in a minimal time. Typically, services with no dependencies in the workflow graph are executed concurrently on different grid resources: this execution mode is our baseline experimental set up in the following as it is implemented in all services workflow managers for grids (*e.g.* [9], [10], [11], [12]). To further optimize, different data segments processed by a single service can be executed concurrently: in this case, we refer to *Data Parallel* (DP) execution mode. In addition to DP, sequentially linked services can be pipelined (different data segments can be processed by sequentially linked services concurrently): we refer to *Data and Service Parallel* (DSP) execution mode in this case.

II. WORKFLOW MAKESPAN MODELING

In this section, a probabilistic model of a workflow of services is presented. It yields an estimation of the expectation and standard-deviation of the makespan of the workflow, given that the distribution of the grid latency is known.

A. Critical path of the workflow

In a flow of services, a *path* denotes a set of services linking an input of the workflow to an output. A path is defined independently from the data to process: it will be instantiated at runtime on a set of data segments. The *critical path* of the workflow denotes the longest path in terms of execution time.

B. Notations

Let n_W denote the number of services on the critical path of the workflow and n_D denote the number of data segments to be processed (n_D corresponds to the degree of DP that will be achieved). Let $i \in [0, n_W - 1]$ denote the index of the i^{th} service of the critical path of the workflow. Similarly, let $j \in [0, n_D - 1]$ denote the index of the j^{th} data segment to be processed by the workflow. $T_{i,j}$ denotes the duration in seconds of the processing of the data set j by the service i . It corresponds to the total time from the job submission to its completion. $T_{i,j} = r_{i,j} + R_{i,j}$ is made of an application-dependent part $r_{i,j}$ and the grid latency part $R_{i,j}$. $r_{i,j}$ corresponds to the computation time of service i

on the data segment j . It is supposed to be a fixed value (predictable execution time) by opposition to $R_{i,j}$ which is a random variable. $R_{i,j}$ will model all the sources of variability coming from the infrastructure. For instance, the variability coming from the performance of the grid nodes or the network connection of the execution site will be included in this variable. To study the impact of the variability of the grid on the performance of the application, the case where $R_{i,j}$ is a fixed value will also be considered in the following.

The goal of the next sections is to express the application makespan Σ with respect to n_D , n_W , $r_{i,j}$ and $R_{i,j}$ and to the parallelism configuration.

C. Hypotheses

The n_D data segments on which the application is iterated are assumed to be of equal size, which is realistic for applications such as parameter sweeps. Consequently, the execution times $r_{i,j}$ of the jobs can be assumed to be independent from the data: $\forall j, r_{i,j} = r_i$. Performance differences between the CPUs will be included in $R_{i,j}$. If the variability of the execution times of the services has to be taken into account, one should also consider the execution times of the services as random variables. Then, in the following, $T_{i,j}$ notations should not be expanded (into $r_i + R_{i,j}$) and the distribution of this random variable could be determined with respect to the distributions of the execution times and of the grid latency $R_{i,j}$. Yet, in this work, we concentrate on the variability introduced by the grid infrastructure itself rather than on the intrinsic variability of the algorithms, which is completely application-specific.

$R_{i,j}$ are assumed to be independent random variables: the dependencies among the job latencies are neglected. Given the scale of production grid infrastructures, this hypothesis can be considered as realistic. What is assumed here is that the application itself does not impact the grid latency significantly. Bottlenecks may challenge this hypothesis. For instance, the submission time of several jobs from the same machine is very likely to depend on the number of submitted jobs. Taking this phenomenon into account may not be easy from a general perspective: understanding how jobs interact with each other in the whole system seems difficult. Still, for specific steps such as the submission, some models could be integrated to take into account the interactions between jobs.

The grid latency is assumed not to depend on the nature of the submitted jobs. It is true that the queuing time of the job in the batch of a computing center is highly dependent on the expected duration of the task. However, as it is done by the huge majority of grid end-users, the expected wall-clock time of the job is assumed to be set to its default value, which is supposed to be much higher to the effective duration of the submitted jobs. Consequently, the distribution of the grid latency is assumed to be independent from i . Similarly, the distribution of the grid latency is supposed to be independent from the data (*i.e.* the distribution of $R_{i,j}$ is independent from j). Assuming that the distribution of the latency is independent from the service and from the data is not so critical. Considering applications handling large volumes of

data (*i.e.* applications for which data transfer times would be of several minutes), one could simply include it into the r_i value. Problems may only arise for applications for which the data impacts the job life cycle inside the system, *i.e.* disturbs its submission, scheduling or queuing time. If they ever exist, such interactions should be of limited importance and still negligible with respect to the average grid latency. Thus, $R_{i,j}$ are assumed to be independent and identically distributed (iid) random variables.

D. Workflow makespan expression

Under those hypotheses, the expression of the makespan of the workflow during a DP or DSP execution can be derived.

DP case. All the data segments are processed concurrently and the execution is synchronized after each service invocation.

$$\begin{aligned}\Sigma_{DP} &= \sum_{i < n_W} \max_{j < n_D} \{T_{i,j}\} = \sum_{i < n_W} \max_{j < n_D} \{r_i + R_{i,j}\} \\ &= \sum_{i < n_W} r_i + \sum_{i < n_W} \max_{j < n_D} \{R_{i,j}\}\end{aligned}\quad (1)$$

DSP case. All the data segments are processed concurrently and the services are pipelined.

$$\begin{aligned}\Sigma_{DSP} &= \max_{j < n_D} \left\{ \sum_{i < n_W} T_{i,j} \right\} = \max_{j < n_D} \left\{ \sum_{i < n_W} (r_i + R_{i,j}) \right\} \\ &= \sum_{i < n_W} r_i + \max_{j < n_D} \left\{ \sum_{i < n_W} R_{i,j} \right\}\end{aligned}\quad (2)$$

E. Deterministic case

If the latencies $R_{i,j}$ were fixed values, then for every i and every j , $R_{i,j} = \bar{R}$ and the above expressions simplify:

$$\Sigma_{DP} = \Sigma_{DSP} = \sum_{i < n_W} r_i + n_W \cdot \bar{R}\quad (3)$$

In this case, there is no difference between the DP and DSP cases. This deterministic model will be used to forecast the performance of the application in absence of variability of the latency. It corresponds to a theoretical non-variable system which has the same average latency as the production grid.

F. Probabilistic case

The goal is to determine the expectation of the makespan of the workflow $E(\Sigma)$ and its standard deviation $\sigma(\Sigma)$ as a measure of its variability. In the following, given a random variable X , f_X will denote the probabilistic density function (pdf) of X and F_X its cumulative density function (cdf).

DP case: thanks to the linearity of the expectation operator and to the fact that r_i is a fixed value, equation 1 gives:

$$E(\Sigma_{DP}) = \sum_{i < n_W} r_i + n_W E\left(\max_{j < n_D} \{R_{i,j}\}\right)$$

Given that the cumulative density function of the random variable $K = \max_{j < n_D} \{R_{i,j}\}$ is $F_K = F_{R_{i,j}}^{n_D}$, we finally have:

$$E\left(\max_{j < n_D} \{R_{i,j}\}\right) = n_D \int_{-\infty}^{\infty} t f_{R_{i,j}}(t) F_{R_{i,j}}(t)^{n_D-1} dt,$$

Thus:

$$E(\Sigma_{DP}) = \sum_{i < n_W} r_i + n_W n_D \int_{-\infty}^{\infty} t f_{R_{i,j}}(t) F_{R_{i,j}}(t)^{n_D-1} dt \quad (4)$$

Moreover, given that two jobs are independent, equation 1 gives:

$$\sigma(\Sigma_{DP})^2 = n_W \sigma\left(\max_{j < n_D} \{r_i + R_{i,j}\}\right)^2$$

And thus, because r_i are fixed values:

$$\sigma(\Sigma_{DP})^2 = n_W \sigma\left(\max_{j < n_D} \{R_{i,j}\}\right)^2$$

Given that $\sigma(\max_{j < n_D} \{R_{i,j}\})^2 = E(\max_{j < n_D} \{R_{i,j}\}^2) - E(\max_{j < n_D} \{R_{i,j}\})^2$ and that $E(X^2) = \int_{-\infty}^{\infty} t^2 f_X(t) dt$, we have:

$$\begin{aligned}\sigma(\Sigma_{DP})^2 &= n_W \sigma\left(\max_{j < n_D} \{R_{i,j}\}\right)^2 \\ &= n_W \left[n_D \int_{-\infty}^{\infty} t^2 f_R(t) F_R(t)^{n_D-1} dt \right. \\ &\quad \left. - n_D^2 \left(\int_{-\infty}^{\infty} t f_R(t) F_R(t)^{n_D-1} dt \right)^2 \right]\end{aligned}\quad (5)$$

DSP case: The max operator prevents from simplifying the expressions of the expectation and standard-deviation of the makespan. Yet, those values can still be computed numerically, as it will be done in section III.

$$E(\Sigma_{DSP}) = \sum_{i < n_W} r_i + E\left(\max_{j < n_D} \left\{ \sum_{i < n_W} R_{i,j} \right\}\right)\quad (6)$$

$$\sigma(\Sigma_{DSP}) = \sigma\left(\max_{j < n_D} \left\{ \sum_{i < n_W} R_{i,j} \right\}\right)\quad (7)$$

III. EXPERIMENTAL RESULTS

The goal of this section is to present experimental results that will be used to:

- 1) evaluate the relevance of the model presented above to explain the makespan of the application; and
- 2) study the impact of the latency variability on the execution of a workflow on a production grid.

An application to medical image analysis algorithms assessment is first introduced. The results obtained running this application on the EGEE grid are then presented.

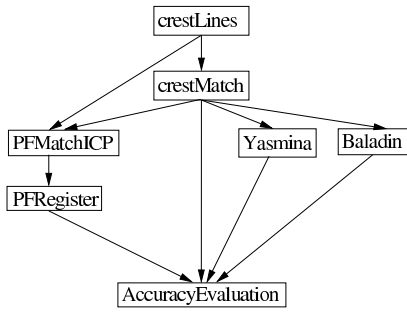


Fig. 1. Bronze standard production application workflow: six services are executed on the grid for each image pair to process. The AccuracyEvaluation service is a lightweight process combining all former results that is not taken into account here.

A. Medical image analysis application

The application used for these experiments is designed to evaluate a class of medical image analysis algorithms known as *rigid registration* algorithms. It is a compute intensive optimization procedure that evaluates a *bronze standard* from a statistically significant image set [13]. It is workflow-based and very scalable as the larger the data set to process, the more accurate the statistical estimate. The simplified application workflow is illustrated in figure 1. The inputs for this workflow are pairs of images corresponding to different acquisitions of a same patient. Up to 126 Magnetic Resonance Image pairs of the brain were available for the experiments reported here.

B. Experimental conditions

The medical application was executed on different input data sets sizes, ranging from 12 to 126 image pairs. Each one of the input image pairs led to 6 job submissions on the grid. Thus, the amount of tasks submitted ranged from 72 to 756. We used the MOTEUR workflow manager developed in our group to run the application [12]. It enables the execution of flows of services on the EGEE grid both in DP and DSP modes. With more than 30'000 CPUs distributed over 200 sites and more than 5'000 registered users, EGEE is a very large scale shared infrastructure. The workflow executions were not simultaneous. Submitting all the executions simultaneously would not have been possible without introducing strong biases in the results. Indeed, the submission middleware would have become a bottleneck and it is very likely that the executions would have disturbed each other. Changes in the grid status (number of available sites, average load...) may thus happen between those runs. They are captured by the fitting of the parameters of the latency distribution that is adapted to the execution conditions, as developed below.

On a production grid infrastructure, setting a timeout to tasks is mandatory because a small fraction of tasks are likely to remain blocked for hours in a waiting queue or even to get lost: the timeout value prevents the application from facing outliers. Because of that, and taking into account failures that are likely to occur, tasks need to be resubmitted if necessary. For example, on the EGEE grid, the tasks success

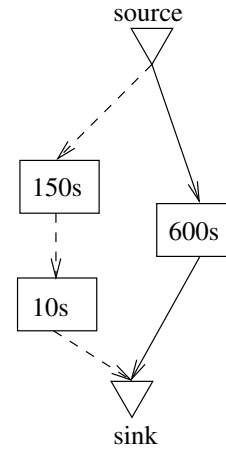


Fig. 2. Considering this workflow, if $R_{i,j}$ are assumed Gaussian with $\mu = 300$ s and $\sigma = 200$ s, and if a single data segment is processed, then the critical path of the workflow is the plain one which is expected to be 900 s (300 s for the expected latency + 600 s for the execution) whereas the expectation of the dashed one is only 760 s (2×300 s + 160 s). But as soon as the number of data segments is greater or equal to 3, then the critical path of the workflow becomes the dashed one: for 3 data segments, the expectation of the dashed path is 1098 seconds whereas it is 1069 seconds for the plain path (using equation 4).

rate was around 84% at the time of those experiments. In those experiments, the timeout value was arbitrarily set to 1 hour (which is far greater than the services wall-time r_i) and no retry was performed in order to prevent the makespan to be influenced by resubmissions that are not modeled. Thus, timed-out jobs are neglected. A strategy to optimize the timeout value is described in [1].

C. Model computation

To compute the probabilistic model presented in section II-F, the required parameters are (i) the deterministic part of the running time of each service on a single data set r_i and (ii) the mean μ and standard deviation σ of the grid latency.

The r_i values were obtained by benchmarking the workflow services during various run and averaging the results. In the experiment presented here, μ and σ are estimated *a posteriori*, from the logs of the execution, in order to keep off the problem of estimating up-to-date parameters. Indeed, the goal of this experiment is not to obtain an up-to-date model of the distribution of the grid latency but rather to validate a model of the *application*, assuming that the distribution of the latency is known. μ and σ were thus evaluated from the execution trace. Estimating them *a priori* requires a dedicated grid monitoring system, which is out of this paper scope.

The first step required for the computation of the model is to determine the critical path of the workflow. Because of the variability of the latency, the expected critical path depends on n_D , as suggested by figure 2. Thus, we determined the critical path of the workflow separately for each workflow run.

The value of the makespan obtained from the deterministic model is an estimate of the performance that could be obtained in absence of variability. As suggested by equation 3, it is

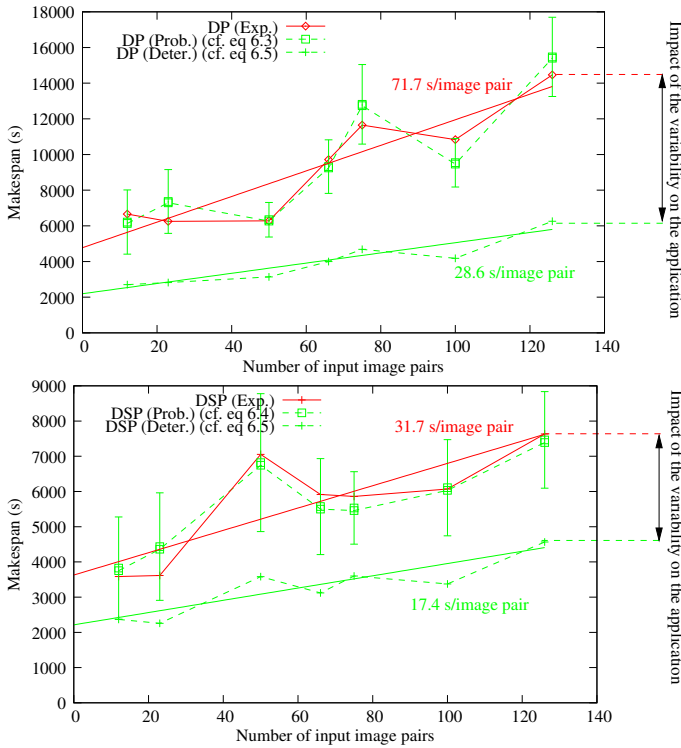


Fig. 3. Comparison of the makespan of the application in the experimental and model cases. Top: DP case. Bottom: DSP case. The DP case is less robust to the latency distribution tail, which explains its weaker performance. The impact of the variability of the latency can be noticed by comparing the deterministic (bottom dashed green curves) case with the experimental (plain red) one. Variability leads to a factor 2 performance drop on this application.

computed by considering that the latency is a fixed value (the average value of the observed latency).

D. Results

Figure 3 displays the experimental results for DP and DSP executions. On both graphs, the experimental data is depicted in plain red. Probabilistic models are represented with squares and deterministic ones with crosses. For the experimental and deterministic cases, a linear regression is superimposed. For the probabilistic cases, intervals corresponding to $[\mu - 3\sigma, \mu + 3\sigma]$ are also drawn.

IV. MODEL EVALUATION AND PERFORMANCE ANALYSIS

A. Metrics for the analysis

To analyze performances, the first relevant metric from the user point of view is the speed-up, measured as the ratio of the makespan over the sequential execution time. The most interesting speed-up value is the maximal one obtained on the application, which in this case is the one obtained for the largest input data set.

To have a finer interpretation of the results, the global behavior of the application makespan with respect to the number of input data sets can be approximated with straight lines estimated through a linear regression. Those fitted straight lines are also plotted on figure 3. The relative error of this

approximation with respect to the experimental data is 7.8% for the DSP case and 11.6% for the DP one.

The y-intercept and slope of the fitted lines can then be considered as metrics. The y-intercept value, expressed in seconds, measures the latency of the application on this infrastructure. This value corresponds to the nominal latency of the grid added to the execution time of a single data set by the application workflow: it is the incompressible amount of time required to access the infrastructure. The slope of the fitted line, expressed in seconds by jobs, is related to the throughput of the application. This value measures the data scalability of the infrastructure, that is to say its ability to process huge data sets with the same level of performance.

The values of those metrics are reported in table I. The two first columns of this table correspond to the experimental values for the DP and DSP cases. The two next ones correspond to the values computed with the probabilistic model of section II-F, from measured mean and standard-deviation of the latency. The two following columns correspond to the values computed with the deterministic model of section II-E. Those values correspond to the ones that would have been obtained if the infrastructure were not variable.

B. Relevance of the probabilistic model

First, from a qualitative point of view, the results shown on figure 3 exhibit some singular behaviors. For instance, even if the global trend of the curves is to increase with the number of input image pairs, one can notice some local decreases, as between 50 and 75 input images for the DSP case and between 75 and 100 input images for the DP one. It is correctly explained by the model, thanks to the fitting of the parameters (mean and standard-deviation of the latency) to the experimental data. Actually, those local decreases can be explained by a diminution of the latency mean and standard-deviation between those values which do not correspond to simultaneous executions, as already mentioned.

Another singular behavior are the measures done for 50 input images pairs. Indeed, the DP case is there faster than the DSP one. Here again, this behavior can be explained by changes of the grid status between those two runs: it would not have happened if the execution were simultaneously submitted. However, the probabilistic model is again able to explain this behavior thanks to the *a posteriori* fitting of the Gaussian distribution to the observed one.

From a quantitative point of view, and as figure 3 shows, the probabilistic model is quite relevant and able to explain the experimental results (on this figure, experimental results are displayed in plain red and values from the probabilistic model are depicted with squares). The mean relative error of the probabilistic model with respect to the experimental data is 6.7% for the DSP case and 8.4% for the DP one. The fact that this error is greater in the DP case than in the DSP one is consistent because the makespan of the application is more affected by distribution tails in the DP case than in the DSP one. Indeed, in the former case, the processing of every data segment is depending on the processing of all the

	Experiment		Probabilistic Model		Deterministic Model	
	DP	DSP	DP(eq 4)	DSP(eq 6)	DP(eq 3)	DSP(eq 3)
y-intercept (seconds)	4778.0	3628.2	4921.6	4002.4	2195.2	2214.5
Slope (s/data sets)	71.7	31.7	72.2	26.0	28.6	17.4
Max speed-up	7.0	13.2	6.5	13.5	15.9	21.7

TABLE I

METRICS VALUES. THE RELEVANCE OF THE PROBABILISTIC MODEL CAN BE NOTICED BY COMPARING COLUMNS 4 AND 5 TO COLUMNS 2 AND 3. THE IMPACT OF THE VARIABILITY OF THE LATENCY ON THE APPLICATION CAN BE QUANTIFIED BY COMPARING COLUMNS 6 AND 7 TO COLUMNS 2 AND 3.

others because the execution is synchronized after each service invocation. It is also worth noticing that all the experimental values stay inside the $[\mu - 3\sigma, \mu + 3\sigma]$ interval. It shows that the model is able to provide bounds for the error it makes with respect to the experimental case. The observed error basically indicates how realistic our assumptions are. In particular, assuming that job latencies are i.i.d variables seems reasonable here.

The speed-up figures measured and displayed in table I (7.0 and 13.2 in the DP and DSP cases respectively) are very close to the probabilistic model estimates (6.5 and 13.5 respectively), showing that MOTEUR efficiently enables the workflow, data and service parallelism without introducing a significant performance loss.

C. Impact of the service parallelism

It has been explained in section II-F that in a deterministic system, the DP and DSP cases lead to identical performance. Considering the maximal experimental speed-up values, the DSP case was 1.8 times faster than the DP one. The y-intercept metric is 1.3 times higher in the DP case than in the DSP one. The slope ratio comparing those two cases is 2.3.

The fact that service parallelism does speed the execution up can be explained by the service parallelism making the application less sensitive to distribution tails. If no variability was possible (deterministic model), the impact of service parallelism would indeed be lower: the maximal speed-up ratio would be 1.4, the y-intercept ratio would be 1.0 and the slope ratio would be 1.6. It confirms the behavior described above: the more variable the infrastructure, the more interesting the service parallelism.

The impact of service parallelism is higher on the slope than on the y-intercept value: for the experimental case, table I shows that service parallelism reduces the slope with a factor 2.3, whereas it only leads to a factor 1.3 on the y-intercept. It is consistent that the benefit yielded by service parallelism mainly affects the data scalability of the application: the higher the number of submitted jobs, the higher the probability to lie in the distribution tail.

However, even in case of a non variable platform, there is still an impact of service parallelism on the slope of the straight lines and thus on the maximal speed-up, whereas there is no more on the y-intercept value. This can be explained by the fact that service parallelism reduces the mean grid latency due to sequential procedures such as the submission time. Indeed,

if service parallelism is not present, waves of simultaneous job submissions occur, whereas submissions are more spread over time in case of service parallelism. This explains the impact of service parallelism on the scalability of the application.

D. Impact of variability

The impact of the variability of the grid latency on the makespan of the application is represented by the distance between the dashed green and plain red curves on figure 3. Considering the values of table I, variability led to a maximum speed-up reduction factor of 2.4 for the DP case and 1.6 for the DSP one. If the infrastructure were deterministic, we would obtain a maximal speed-up of 21.7 in the DSP case, whereas it is only 13.2 there. Considering the y-intercept metric, variability leads to an increased factor of 2.24 for the DP case and this factor is 1.8 for the DSP one. Variability also introduces a 2.5 increase factor on the slope metric for the DP case and a 1.5 one for the DSP case. Variability has more impact on the DP case than on the DSP one. Indeed, as already mentioned before, the DP case is far less robust than the DSP one.

The estimates made for a deterministic system show that an additional speed-up in the order of 2 can be expected by adopting strategies to reduce the system variability.

E. Analysis of the grid's latency

The total mean latency introduced by the grid is slightly growing with the number of input data sets, as displayed on figure 4. This figure plots the mean latency obtained for the DP and DSP cases and identifies the different sources of latency, namely submission, scheduling and queuing times and the overhead added to the wall-time. These values were obtained by subtracting the average benchmarked wall-time to the average actual wall-time of the tasks. The slow latency increase shows that the large infrastructure is far from saturation.

Table II displays the mean values obtained for each entity of the infrastructure. The most important source of latency is the queuing time, as it is easily understandable on a multi-users platform. Then comes the overhead on the wall-time, that includes data transfers and performance of the running hosts. Submission and scheduling times are the less important sources of overhead. The latency coming from the load of the infrastructure is distributed among those four entities. Yet, most of it may be included in the queuing latency. The latency coming from the wall-time of the jobs covers the

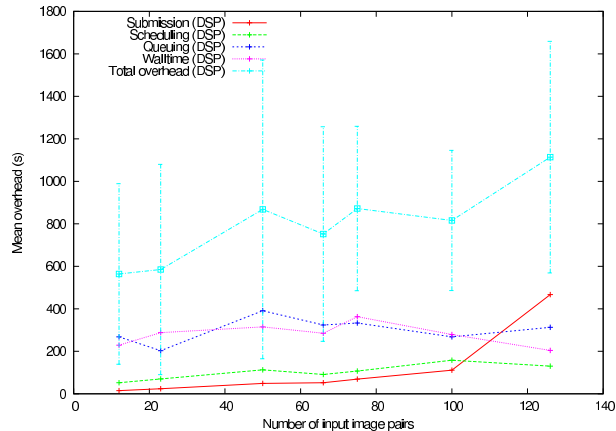
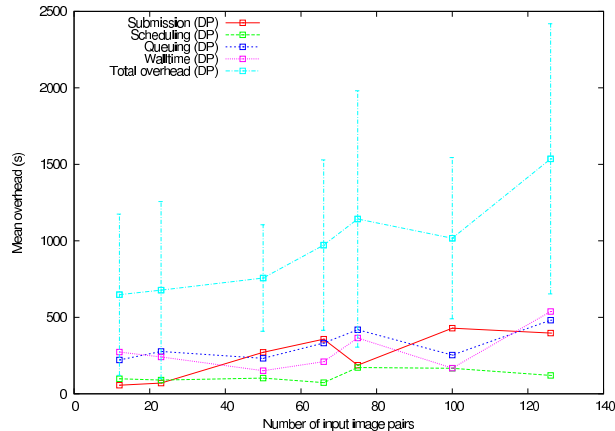


Fig. 4. Mean overhead for each grid's component. The standard deviation of the total latency is plotted on the corresponding curve. Left: DP. Right: DSP.

Entity	Mean latency (s)
Submission	182
Scheduling	110
Queuing	308
Walltime	279
Total	880

TABLE II
MEAN GRID OVERHEAD FOR EACH COMPONENT

heterogeneity of the machines of the grid. Indeed, the services have been benchmarked on a particular machine and the performance of the grid worker nodes is unknown. All those values have been measured with the grid information system. They are thus highly dependent on its accuracy. In particular, too small update frequencies may disturb those measures. Yet, applications also rely on this information system so that those values are representative of what could be measured from the applications.

The variability of the overhead is hardly interpretable. The standard deviation of the total overhead varies from 390 s to 890 s but it does not exhibit global trends.

V. CONCLUSION

The probabilistic model of workflows makespan presented in this paper captures the overall grid complexity through a simple random latency variable. The makespan expectation and standard deviation were derived for two different parallelization modes (DP and DSP) that are commonly used for scientific workflows. The model validity was demonstrated on a medical image analysis application, using prior estimation of the grid latency distribution function. Variability is a source of performance loss. Extrapolating the model to a theoretical deterministic infrastructure, it was shown that a speed-up factor of the order of 2 can be expected on our application by reducing the impact of the grid variability.

In the future, the model could be used to anticipate the expected makespan of applications and design optimization strategies. For this use, the grid latency has to be evaluated prior to the application runs. Due to variable workload conditions over time, an update mechanism for this evaluation has to be set up. A first step could be to study the grid over a significantly long period of time in order to determine whether latency observations for run $n+1$ could be induced from run n .

VI. RELATED WORK

The probabilistic modeling of applications has been investigated for quite a long time. However, the sources of variability were not the same and the application areas thus significantly differed from this paper. Statistical investigations on grid systems have only been introduced in the last years. A broad survey of such methods is reported in Feitelson's in progress book² which synthesizes many of his papers [4], [14]. Yet, as far as we know, such methods have only been introduced from the infrastructure's point of view so far. For instance, statistical attempts have been done to model the job inter arrival time of a cluster of the grid. The idea of considering the whole grid as a black box introducing a random latency on the jobs submitted by the user is original.

Probabilistic approaches to performance analysis have been used for quite a long time in parallel and distributed applications. Gelenbe *et al* [15] and Mussi and Nain [6] already considered the execution time of a task-graph as a random variable and determined its distribution from the graph parameters and topology. Even if the motivating problem of those works is very different to ours (in [15], the variability is related to the topology of the task graph and in [6], only task trees are considered), the probabilistic tools employed are very similar, reinforcing the idea that they are adequate to model this kind of problem.

²<http://www.cs.huji.ac.il/~feit/wlmod/>

Later on, Gautama *et al* [5] noticed that directly using the pdf to determine the execution time of the application leads to heavy computations preventing from any practical application. They thus proposed an approach based on the four first moments of the distribution. The authors also take into account more complex program patterns. However, parallel operators raise problems in this framework and density functions have to be approximated with generalized lambda distributions, characterized by four parameters only [16]. Assuming that, the moments of the execution time of the graph are expressed from the ones of the tasks. Results concerning normal distributions show that the error made by the approximation remains under 1% for 1000 parallel tasks. However, only low mean and standard deviation values are presented due to numerical instabilities.

Close to this approach, Schopf and Berman use stochastic values, defined by their mean and standard deviation to model the execution time of an application [17], [2]. They define arithmetic operations using the arithmetic on normal distributions. As in Gautama's work, the definition of the max operation, that is critical in a parallel execution is not obvious and has to be "supplied by the model builder, scheduler or user". The application model presented in this work seems to be quite specific whereas using a workflow representation allows us to describe any workflow-based application in a more generic way.

Works such as [18] and inside references propose performance analysis methods for task scheduling into embedded systems, considering probabilistic models of task execution times. In this work, the authors model task execution by a generalized continuous probability distribution and propose a method not restricted to any specific scheduling policy. They consider both execution time and memory aspects. Their method is based on the construction of an underlying stochastic process and its analysis. Even if this approach is entirely probabilistic and makes no assumption on the nature of the probability function of the execution time, which well suits with our hypotheses, they assume all the tasks to be executed concurrently on a single processor.

In practice, the probabilistic approaches mentioned in the previous paragraphs have never been applied to production grid infrastructures at the scale we are demonstrating here. Even the recent work of Schopf and Berman described above exhibits very different orders of magnitude to ours. Results are showed on a cluster environment whereas the EGEE grid on which we conducted our experiments is much wider. Consequently, variability in [2] is about 100 seconds whereas it can reach 900 seconds in our case. In our case, variability is related to the grid latency itself, which does not occur in such proportion on smaller platforms.

General considerations about features and architecture required for an efficient production grid (particularly focusing on data transfers) are discussed in [19] from the experience of the EU DataGrid project. This work focus on the large-scale multi-users grid that we are also targeting here. However, no model to explain the infrastructure's behavior is proposed.

ACKNOWLEDGMENT

This work is partially funded by the AGIR and the GWEN-DIA projects (contract number ANR-06-MDCA-009) from the French National Research Agency MDCA program. We are grateful to the EGEE European project for providing the grid infrastructure and user assistance.

REFERENCES

- [1] T. Glatard, J. Montagnat, and X. Pennec, "Optimizing jobs timeouts on clusters and production grids," in *International Symposium on Cluster Computing and the Grid (CCGrid)*. Rio de Janeiro: IEEE, May 2007, pp. 100–107.
- [2] J. Schopf and F. Berman, "Using Stochastic Information to Predict Application Behavior on Contended Resources," *International Journal of Foundations of Computer Science*, vol. 12, no. 3, pp. 341–364, 2001.
- [3] A. Kesselman and Y. Mansour, "Optimizing TCP Retransmission Timeout," in *International Conference on Networking*, ser. LNCS, vol. 3421, Saint-Denis de la Réunion, 2005.
- [4] D. Feitelson, *Workload modeling for performance evaluation*. Springer-Verlag - LNCS vol 2459, Sep. 2002, pp. 114–141.
- [5] H. Gautama, "A probabilistic approach to the analysis of program execution time," Master's thesis, Delft University of Technology, Information Technology and Systems, 1998.
- [6] P. Mussi and P. Nain, "Evaluation of parallel execution of program tree structures," in *Proceedings of the 1984 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '84)*. New York, NY, USA: ACM Press, 1984, pp. 78–87.
- [7] P. Kacsuk and G. Sipos, "Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal," *Journal of Grid Computing (JGC)*, vol. 3, no. 3-4, pp. 221 – 238, Sep. 2005.
- [8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbre, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing (JGC)*, vol. 1, no. 1, pp. 9–23, 2003.
- [9] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience*, 2005.
- [10] I. Taylor, I. Wand, M. Shields, and S. Majithia, "Distributed computing with Triana on the Grid," *Concurrency and Computation: Practice & Experience*, vol. 17, no. 1–18, 2005.
- [11] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pockock, A. Wipat, and P. Li, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinformatics journal*, vol. 17, no. 20, pp. 3045–3054, 2004.
- [12] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and efficient workflow deployment of data-intensive applications on grids with MOTEUR," *International Journal of High Performance Computing and Applications (IJHPCA)*, 2008, to appear in.
- [13] T. Glatard, X. Pennec, and J. Montagnat, "Performance evaluation of grid-enabled registration algorithms using bronze-standards," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI'06)*, ser. LNCS, Copenhagen, Denmark, Oct. 2006.
- [14] D. Feitelson, "Metric and workload effects on computer systems evaluation," *Computer*, vol. 36, no. 9, pp. 18–25, Sep. 2003.
- [15] E. Gelenbe, E. Montagne, R. Suros, and C. Woodside, "A performance model of block structured parallel programs," in *International Workshop on Parallel Algorithms and Architectures*. Luminy, France: Elsevier Science B.V (North Holland), Apr. 1986, pp. 127–138.
- [16] H. Gautama and A. J. C. van Gemund, "Symbolic Performance Estimation Of Speculative Parallel Programs," *Parallel Processing Letters*, vol. 13, no. 4, pp. 513–524, 2003.
- [17] J. Schopf and F. Berman, "Performance prediction in production environments," in *12th International Parallel Processing Symposium*, Orlando, Florida, USA, Apr. 1998, pp. 647–653.
- [18] S. Manolache, P. Eles, and Z. Peng, "Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Time," in *Euromicro Conf. on Real-Time Systems*, Delft, The Netherlands, 2001.
- [19] E. Laure, H. Stockinger, and K. Stockinger, "Performance Engineering in Data Grids," *Concurrency and Computation: Practice & Experience*, vol. 17, no. 2-4, 2005.