Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers

Marco A. S. Netto, *Member, IEEE,* Rajkumar Buyya, *Senior Member, IEEE*

Abstract—Metaschedulers can distribute parts of a Bag-of-Tasks (BoT) application among various resource providers in order to speed up its execution. The expected completion time of the user application is then calculated based on the run time estimates of all applications running and waiting for resources. However, due to inaccurate run time estimates, initial schedules are not those that provide users with the earliest completion time. These estimates increase the time distance between the first and last tasks of a BoT application, which increases average user response time, especially in multi-provider environments. This paper proposes a coordinated rescheduling algorithm to handle inaccurate run time estimates when executing BoT applications in multi-provider environments. The coordinated rescheduling defines which tasks can have start time updated based on the expected completion time of the entire BoT application. We have also evaluated the impact of system-generated run time estimates to schedule BoT applications on multiple providers. We performed experiments using simulations and a real distributed platform, Grid'5000. From our experiments, we obtained reductions of up to 5% and 10% for response time and slowdown metrics respectively by using coordinated rescheduling over a traditional rescheduling solution. Moreover, coordinated rescheduling requires little modification of existing scheduling systems. System-generated predictions, on the other hand, are more complex to be deployed and may not reduce response times as much as when using coordinated rescheduling.

Index Terms—Rescheduling, bag-of-tasks, resource allocation, grid computing, metascheduling, parallel computing, performance prediction, run time estimates, quality-of-service.

1 Introduction

Bags-of-Tasks (BoTs) are parallel applications with no inter-task communication. A variety of problems in several fields, including computational biology [1], image processing [2], and massive searches [3], have been modeled as BoT applications. In comparison to the message passing model, BoT applications can be easily executed on multiple resource providers to meet a user deadline or reduce the user response time. Although BoT applications comprise independent tasks, the results produced by all tasks constitute the solution of a single problem. In most cases, users need the entire set of tasks completed to be able to post-process or analyse the results. Therefore, the optimization of the aggregate set of results is important, and not the optimization of a particular task or group of tasks [4], [5].

Metaschedulers can distribute parts of a Bag-of-Tasks (BoT) application among various resource providers in order to speed up its execution. The expected completion time of the user application is then calculated based on the run time estimates of all applications running and waiting for resources. A common practice is to overestimate execution times in order to avoid user applications to be aborted [6], [7]. Therefore, initial completion time promises are usually not accurate. In addition, when a BoT application is executed across multiple clusters, inaccurate estimates increase the time difference between the completion of its first and last task, which increases average user response time. This time difference, which we call *stretch factor*, increases mainly because rescheduling is performed independently by each provider.

System-generated predictions can reduce inaccurate run time estimates and avoid users having to specify these values. Several techniques have been proposed to predict application run times and queue wait times. One common approach is to analyse scheduling traces; i.e. historical data [8], [9], [10]. Techniques based on trace analyses have the benefit of being application independent, however may have limitations when workloads are highly heterogeneous. Application profiling has also been vastly studied to predict execution times [11], [12], [13], [14]. Application profiling can generate run time predictions for multiple environments, but usually requires application source code access.

This paper proposes a coordinated rescheduling strategy for BoT applications running across multiple resource providers. Rather than providers performing independent rescheduling of tasks of a BoT application, the metascheduler tracks of the expected completion time of the last BoT task. This strategy minimizes the stretch factor and reduces user response time. We also show that on-line system generated predictions, even though require time to be obtained, can reduce user response time when compared to user estimations. Moreover, with more accurate predictions, providers can offer tighter expected completion times, thus increasing system utilization by attracting more users. We performed experiments using simulations and a real

• *The authors are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.*
E-mail: {netto, raj}@csse.unimelb.edu.au

distributed platform, Grid'5000, on homogeneous and heterogeneous resources. We also provide an example of system-generated predictions using POV-Ray, a ray-tracer tool to generate three dimensional images to compose animations. The experiments consider three main variables: time to generate run times, accuracy of run time predictions, and time users are willing to wait for scheduling decisions.

2 Related Work

Extensive research has been done on performance prediction techniques, use of predictions for scheduling applications, the impact of inaccurate run time estimates for scheduling, and scheduling algorithms for BoT applications, which are the main related areas to our work. Due to space constraints, we present a few projects on these areas.

Several researchers have developed run time prediction techniques based on historical data of previous application executions [8], [9], [10], [15], [16]. These techniques vary in complexity and quality of predictions. Tsafrir et al. [8] proposed the use of average time of the previous two executions of jobs with same characteristics to calculate new estimation time, whereas Smith et al. [10] use search for similar jobs of the entire historical data base of previous executions. Run time predictions also assist management systems to provide users with expected waiting times [10] and to produce schedules that increase system utilization [15]. Most of existing techniques based on historical data are for homogeneous resources hosted by a single provider.

Performance predictions for multi-cluster environments have also been developed. Sadjadi et al. [17] proposed a modeling approach for estimating execution times of single-cluster long-running scientific applications for multi-cluster environments. Their approach relies on modeling resources, execution parallelism, and application input parameters, along with a set of previous application executions. The applications can access multiple resources, but those have to be homogeneous. Sanjay and Vadhiyar [12] developed a set of performance modeling strategies to predict execution times of parallel applications for single-cluster applications on Grid Computing environments. Yang et al. [11] introduced a performance translation method based on relative performance between two platforms for single-cluster applications. Predictions are generated based on a short execution of applications in each target platform and require source code access. Romanazzi and Jimack [13] proposed a prediction performance model for parallel numerical software systems on multi-cluster environments. Predictions for large-scale experiments are generated based on executing applications with fewer processors for short time periods, and require source code modification.

He et al. [18] addressed the problem of dynamic scheduling for parallel jobs in multi-cluster systems using performance predictions. In order to obtain run time predictions, they relied on the PACE (Performance Analysis and Characterization Environment) tool-kit [14], which requires application source code access. Berman et al. [19] have also proposed the use predictions for scheduling applications, and users have also to modify the application source code. Sonmez et al. [16] studied the use of time series prediction methods for job run times and queue wait times in Grid environments. They have also evaluated the impact of predictions when scheduling jobs in Grids. Their experiments consider schedulers using FIFO without backfilling and tasks of a BoT application are submitted to a single cluster.

Existing work on BoT applications mostly focuses on the initial scheduling [20], [21], [22], [23], [24], [25]; tasks are scheduled without considering the dynamic behaviour of resources and applications. Researchers have also developed task replication techniques to reduce user response time and handle the lack of information from resources and tasks [26]. Task replication is a particular type of rescheduling, but with the drawback of wasting resources. Therefore, our contribution is a coordinated rescheduling algorithm for BoT applications and an evaluation of impact of run time estimates when scheduling these applications across multiple providers.

3 Scheduling Architecture

A metascheduler receives user requests to schedule BoTs on multiple autonomous resource providers in on-line mode. Users provide the number of required resources along with either a run time estimation or an application profiler. The resources considered are space-shared machines such as clusters and massively parallel processing machines.

As illustrated in Figure 1, the scheduling of a BoT application consists of 6 steps. In step 1, the metascheduler exposes the application profiler or user estimations to the resource providers. In step 2, the resource providers execute the profiler (if available) and generate a list of offers that can serve the entire BoT or only part of it. An offer consists of a number of tasks, and the their expected completion time. In this work, resource providers generate offers to do not violate the expected completion time of already scheduled tasks and to consider the tasks' runtime estimation errors. Once the resource providers generate the offers, they send them to the metascheduler (step 3), which composes them according to the user requirements (step 4), and submits the tasks to resource providers (step 5). After the tasks of a BoT are scheduled, resource providers contact the metascheduler for rescheduling purposes (step 6).

Due to system heterogeneity and the different loads in each resource provider, offers arrive at different time to the metascheduler. Once the metascheduler receives all offers, some of them may no longer be valid since other users submitted applications to the providers. To overcome this problem, we use a similar approach developed by Haji et al. [27], who introduced a Three-Phase

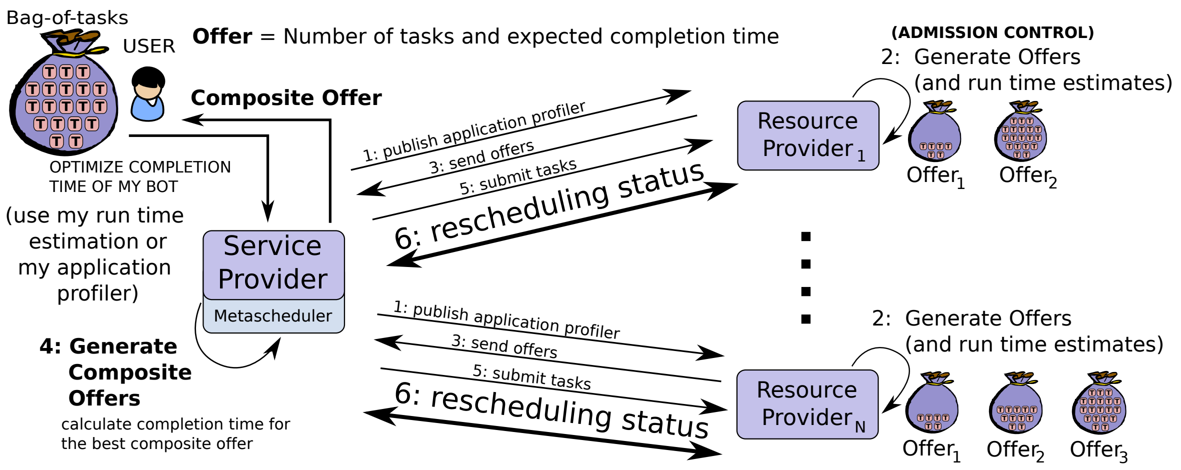

Bag-of-tasks
USER
Offer = Number of tasks and expected completion time
(ADMISSION CONTROL)
2: Generate Offers (and run time estimates)
Composite Offer
1: publish application profiler
3: send offers
5: submit tasks
6: rescheduling status
Resource Provider 1
Offer1
Offer2
OPTIMIZE COMPLETION TIME OF MY BOT
(use my run time estimation or my application profiler)
Service Provider
Metascheduler
4: Generate Composite Offers
calculate completion time for the best composite offer
Resource Provider N
Offer1
Offer2
Offer3

Fig. 1. Components' interaction to schedule a Bag-of-Tasks using application profiling and execution offers.

commit protocol for SNAP-based brokers. Our protocol uses *probes*, which are signals sent from the providers to the metaschedulers interested in the same resources to be aware of resource status' changes.

The **schedulers' goal** is to provide users with expected completion time and reduce such a time as much as possible during rescheduling phases.

4 Coordinated Rescheduling

Once the metascheduler provides the user with an expected completion time of his/her BoT application, tasks can start execution either immediately or after resources become available. For the second case, it means tasks are placed in a waiting queue and can be rescheduled to start before expected when tasks from other applications have inaccurate run time estimates.

Figure 2 illustrates the difference between the traditional rescheduling strategy, which considers all tasks independently, and the coordinated rescheduling, which considers the tasks of a BoT as being part of a single application. In this example, by using uncoordinated rescheduling the BoT application tasks in Site 1 are rescheduled without considering the tasks from Site 2; which increases stretch factor of this BoT, and delays the completion time of application A. By using the coordinated rescheduling approach, application A is rescheduled first since it earlier start time does not increase the *overall* completion time of the BoT application.

Whenever a job completes before the expected time, local schedulers execute Algorithm 1 to reschedule the waiting queue. The first step is to sort the jobs in the waiting queue by increasing order of their expected completion times. Jobs from the same BoT are sorted by increasing order of expected start time individually. Later, jobs are rescheduled one by one. For each job j_i being part of a BoT, the scheduler verifies whether j_i holds the expected completion time of the entire BoT. Both BoT jobs and other type of jobs are then rescheduled using FIFO with conservative backfilling. If a BoT job holds the expected completion time of the entire job and received a new completion time due to rescheduling, the algorithm keeps this job in a structure called *newCompletionTimes*, which contains the job id and the new completion time. After all jobs are rescheduled, the algorithm analyses the last completion time for each BoT in the *newCompletionTimes* structure. The local scheduler sends this structure to the metaschedulers holding the respective BoTs.

From the metascheduler side, each time it receives the $newCompletionTimes$ structure, it verifies whether the new completion times are local or global. If they are global, the metascheduler sends the new completion times to the resource providers holding the respective BoT tasks.

5 On-Line System Generated Run Time Estimations

In environments where resource providers work with precise run time estimates, metaschedulers can better distribute the load, thus reducing applications' response time, and resource providers can publish offers with tighter response times, thus increasing system utilization by attracting more users. However, it is difficult for users to provide accurate estimates, especially when they have various resource set options. One approach to generate run time estimations is through the analysis from previous execution of applications with similar characteristics. The main limitation of this approach is that if applications are submitted by several users with different requirements, it is difficult to find patterns to estimate execution times. Another approach to generate run time estimates is through application profiling via sampling execution.

Usually, tasks in a BoT application have similar nature, and therefore by executing a few tasks it is possible to estimate the overall application execution time. In addition, depending on the application, it is possible to reduce the problem size in order to speed up the

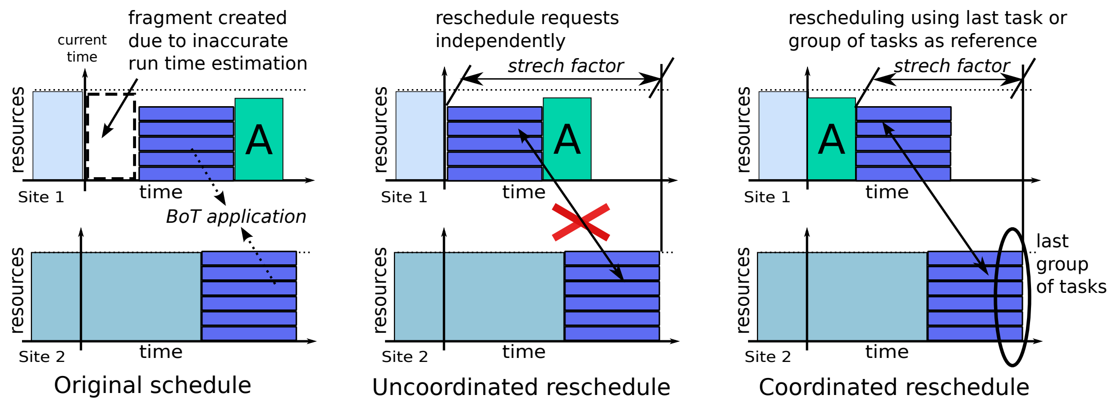


Fig. 2. Example of schedule using coordinated rescheduling.

Algorithm 1: Pseudo-code for rescheduling jobs, which is executed on the local schedulers when a job completes before the expected time.

```
1  Sort jobs by expected completion time. BoT tasks
   are sorted by expected completion of the entire BoT.
   Tasks from the same BoT are sorted by expected
   start time
2  for $\forall j_i \in$ waiting queue do
3      $isLastTask \leftarrow false$
4      $previousCompletionTime \leftarrow j_i$'s completion time
5      if $j_i$ is part of a BoT then
6          $isLastTask \leftarrow$ holds the last expected
           completion time
7      Reschedule $j_i$ using FIFO with conservative
       backfilling
8      if $previousCompletionTime \neq$
       $newCompletionTime$ and $isLastTask = true$
       then
9          add task to possible new completion time list
10 Check new completion times
11 Send new completion times
```

prediction phase. For example, applications in image manipulation can have the problem size reduced by modifying image resolutions. The following sections describe an example of run time generator and discuss when and how to execute the generator.

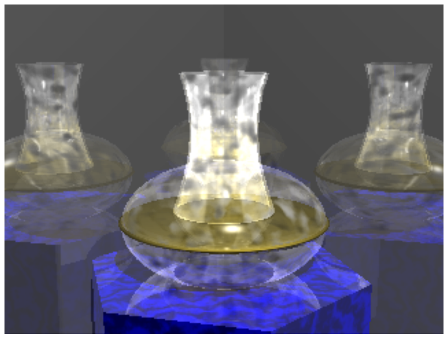
5.1 Example of run time estimator for POV-Ray

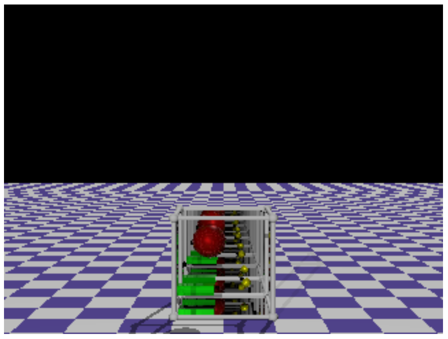
This section describes a run time generator for POV-Ray, a ray-tracer tool in order to generate three dimensional images to compose animations. Ray tracing is a CPU intensive process that depends on several factors such as image size, rendering quality, and objects and materials in a image. We consider users with animation specifications to generate sets of frames. The execution times are unknown since they depend on several factors in each frame and on the properties of the machine processing the frames. Therefore, application profiling can give an insight on the time cost to generate all the frames, which has an impact on the application scheduling. We use POV-Ray to create three short animations containing 200 frames each, with a resolution of 2048x1536 pixels (Quad eXtended Graphics Array).

In order to create the animations, we used three examples of images that come with the POV-Ray package, namely Sky Vase, Box, and Fish. Sky Vase consists of a Vase with a sky texture on a table with two mirrors next to it (we replaced the texture *BrightBlueSky* to *Clouds* in order to increase the workload). To generate the animation for Sky Vase, we rotate the vase 360 degrees. Box consists of a chess floor with a box containing a few objects with mirrors inside. We therefore included a camera the gets closer to the box and crosses it on the other side. Fish consists of a fish over water that rotates 360 degrees. Different from Sky Vase, Fish has a more heterogeneous animation due to the fish's shape (a vase shape is symmetric vertically).

Regarding execution time behaviour, Sky Vase has a steady execution time since the vase is the only object the rotates and its texture has similar work to be processed on each frame. For the Box animation, at the beginning of the animation the box is still far, and hence small, consuming little processing time. However, as the camera approaches the box, more work has to be processed, getting to its maximum when the camera is inside the box. After the camera cross the box, only the floor has to be rendered. The Fish animation has a very heterogeneous execution time due to the fish's shape, which impacts on the amount of work that needs to be processed when rendering the reflex of the fish on the water. An example of images for each of the three animations is illustrated in Figure 3.

When predicting the execution time, one must consider a trade-off between the time spent to predict and its accuracy. The prediction should be fast enough allow prompt scheduling decisions to be made and accurate enough to be meaningful for the schedulers. Apart from that, it should be easy to be deployed in practice. One possibility is to render the animation in a much lower resolution and render a *base frame* of the actual ani-


(a) Sky Vase.


(b) Box.

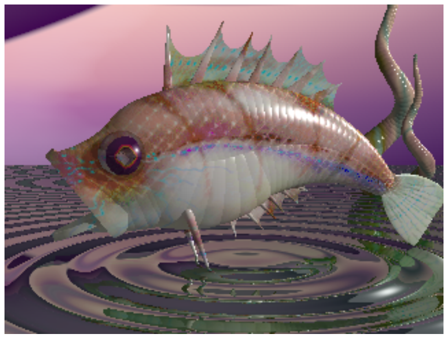
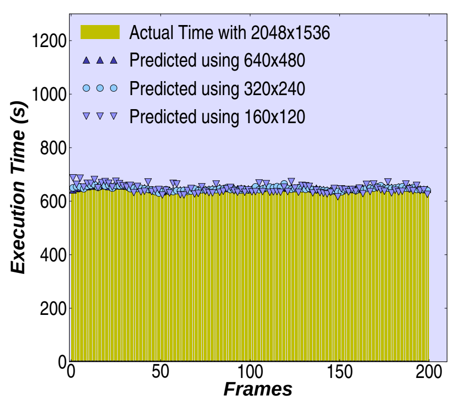
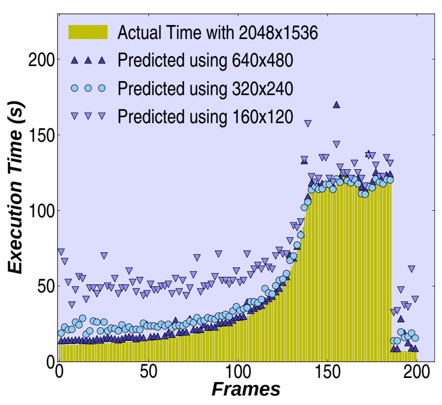

(c) Fish.

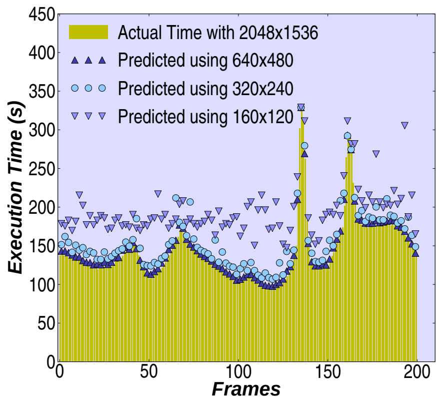
Fig. 3. Example of images for each of the three animations.



(a) Sky Vase.



(b) Box.



(c) Fish.

Fig. 4. Predicted execution time using most CPU consuming frame as base for estimations.

TABLE 1
Time to generate execution time estimates.

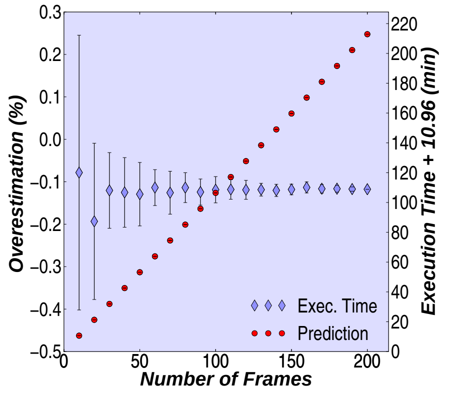
Animation	Resolution	Exec. Time (min)	Perc. of total time	Accuracy
Sky Vase	640x480	223	10.3	0.1% underest.
	320x240	64	2.9	1.3% underest.
	160x120	27	1.2	0.2% underest.
Box	640x480	18.4	12.3	7.30% overest.
	320x240	6.8	4.5	14.00% overest.
	160x120	4.0	2.6	64.90% overest.
Fish	640x480	53.1	11.0	3.03% overest.
	320x240	18.57	3.9	10.47% overest.
	160x120	9.90	2.0	37.64% overest.

mation. Using the execution time of the base frame of the actual and the reduced animation, it is possible to generate a factor to be multiplied on the lower resolution animation to predict the execution actual time of each frame. Figure 4 presents predictions using the maximum and execution time frame as base frame. For this experiment, we used 640x480, 320x240, and 160x120 as lower resolutions for generating predictions. We also observe that both 640x480 and 320x240 resolutions provided much better predictions than 160x120. For the Box animation using the base frame with minimum execution time, all resolutions provided inaccurate predictions for long execution time frames. This happens because the base frame is to small to capture the differences between the resolutions. Table 1 summarizes the execution times to generate the predictions and their accuracies using the base frame with maximum execution time. The results show that good predictions are time consuming since we are using the entire animation.

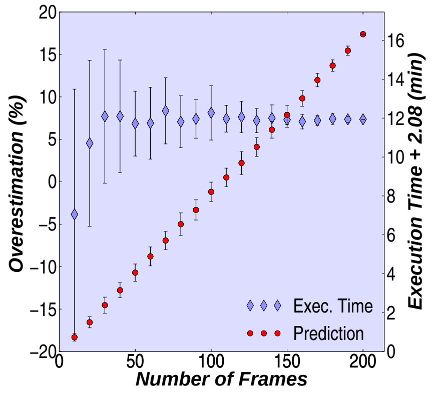
It is possible to reduce the profiling time by sampling a set of frames with a lower resolution rather than using the entire animation. Figures 5 and 6 show the execution time and the prediction accuracy as a function of the number of frames sampled using resolutions 640x480 and 320x240 respectively. For this experiment we used the maximum execution time as base frame. The results show that it is possible to considerably reduce the profiling time keeping a good prediction accuracy level. This happens because in an animation, neighbour frames have similar content and depending on the case the variation is minimum during the entire animation, such as for Sky Vase.

5.2 Where to generate the estimations

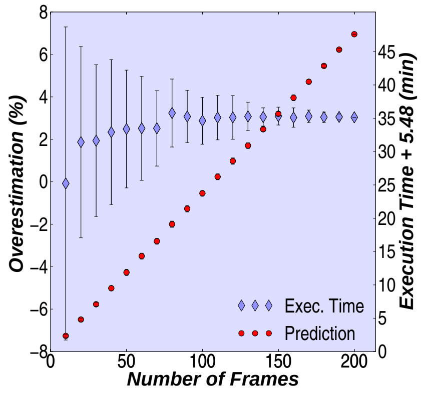
Either users or resource providers can generate run time estimates. If users know the exact resource configuration



(a) Sky Vase.

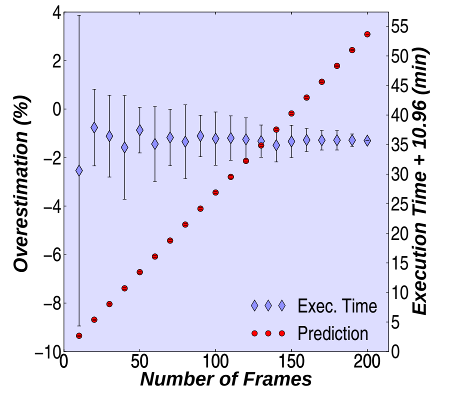


(b) Box.

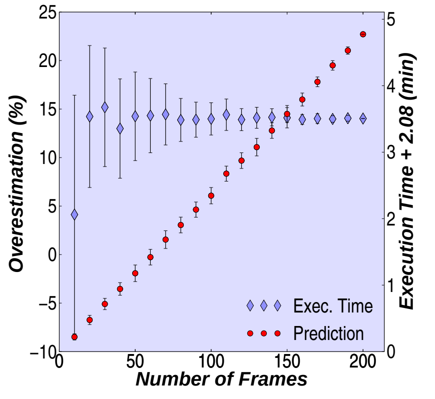


(c) Fish.

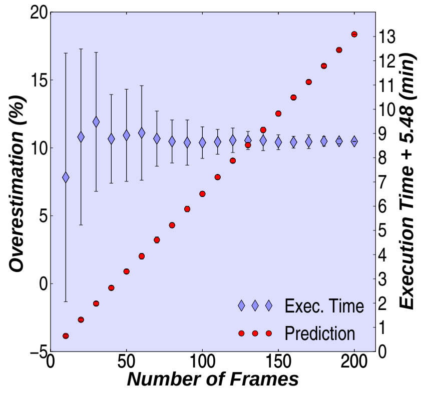
Fig. 5. Predicted execution time using partial sampling of 640x480 frames.



(a) Sky Vase.



(b) Box.



(c) Fish.

Fig. 6. Predicted execution time using partial sampling of 320x240 frames.

for each provider, users can execute the run time estimation generator. Also, even if users do not know the configuration, providers that work with virtual machines can make them available to users. In this case, the users can run the generator through virtual machines in their local machines. Another option is to allow providers to generate run time estimates at the moment users submit their application requirements to the metascheduler. In this case, providers can have a dedicated set of resources to generate run time estimates, or providers can generate them by placing estimator jobs in their shared resources for actual executions. All these options depend on the application, environment settings, and user needs.

6 Evaluation

This section evaluates the coordinated rescheduling algorithm and the impact of inaccurate run time estimates when scheduling BoT applications on multiple resource providers. We performed experiments using both a simulator and a real testbed. Simulations have allowed us to perform repeatable and controllable experiments using various parameters. The experiments in a real testbed allow us to verify how the scheduler architecture can be used in practice. We have used our event-driven simulator, named PaJFit (Parallel Job Fit), and workloads produced by the Lublin-Feitelson model. For the real experiments, we used an extended version of PaJFit, which works with sockets for communication between modules, on Grid'5000. Following we describe the experiment configuration and the analysis of the results.

6.1 Experimental configuration

We have set up a computing environment with a metascheduler and four clusters, C_{1-4}, with 300 processors each. From this environment, we have explored a set of scenarios as described in Table 2. The set of experiments with all clusters with the same configuration helps us to analyse the differences between system and user generated estimations and the rescheduling algorithm. The experiment with all clusters with the same configuration but using different policies for run time estimates helps us to understand which resource provider would benefit more from each approach. We also analyse the impact of heterogeneity for scheduling and rescheduling of bag-of-tasks across multiple providers.

We used the workload model proposed by Lublin and

TABLE 2
Main scenarios for the experiments.

Hardware	Estimation Type and rescheduling
C_1=C_2=C_3=C_4	UE with independent rescheduling
C_1=C_2=C_3=C_4	SE 50% and 80% more accurate than UEs with uncoordinated rescheduling and 5-30min generation time
C_1=C_2=C_3=C_4	UE with coordinated rescheduling
C_1=C_2=C_3=C_4	C_1 and C_2 with UEs and C_3 and C_4 with SEs
C_1=C_2 20 % and 50% faster than C_3=C_4	UEs
C_1=C_2 20 % and 50% faster than C_3=C_4	SEs
C_1=C_2 20 % and 50% faster than C_3=C_4	UEs with coordinated rescheduling

Feitelson [28] to generate traces for both the simulations and the experiments in Grid'5000. We simulated 15 days of the workload and used 10 workloads for each experiment. We also considered 10 run time estimation values. Therefore, for each scenario described in Table 2, we have a total of 100 simulations. For all experiments we set up the system load as 70% by changing the arrival times of the external jobs. To achieve this load we used a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [29], but we fixed the time interval and included more jobs from the trace.

We performed our experiments in Grid'5000 by placing a local scheduler in four clusters with access to 300 processors. Table 3 presents an overview of the node configurations in which we deployed the local schedulers and the metascheduler.[1]

TABLE 3
Overview of the node configurations for the experiments in Grid'5000. Sites are interconnected inside the same VLAN at 10Gbps.

Scheduler	Cluster	Location	CPUs' Configuration
Metascheduler	sol	Sophia	AMD Opteron 2.0 GHz
Provider 1	paradent	Rennes	Intel Xeon 2.5 Ghz
Provider 2	bordemer	Bordeaux	AMD Opteron 2.2 GHz
Provider 3	grelon	Lille	AMD Opteron 2.6 GHz
Provider 4	chicon	Nancy	Intel Xeon 1.6 GHz

6.2 Results and analysis

There are two factors related to the reduction of user response times: load balancing and backfilling. Load balancing can be improved by having better run time estimates, since the metascheduler can decide the right amount of work to send to each provide, whereas backfilling can fill queue fragments generated by earlier completion times of user requests. These fragments can be filled as long as estimations are smaller or the same size as the fragments. By increasing user estimations, more fragments are created and therefore more jobs can be backfilled. However, there is a limit in which backfilling can be explored. Figures 8 and 9 show the requested run times, fragment lengths, and number of jobs that would fit into the fragments for run time estimates with accuracy of 85% and 50%, respectively. We observe that the higher the accuracy the smaller is the number of jobs that have changes of being backfilled. In this example, we are not considering the submission time of the jobs. However, by plotting the total number of jobs that would have chances to be backfilling as a function of run time accuracy, we notice that there is a limit on the backfilling chances. Figure 10 shows that after an overestimation of 200%, the chances of backfilling become steady.

1. More details about the machines in Grid'5000 can be found at https://www.grid5000.fr

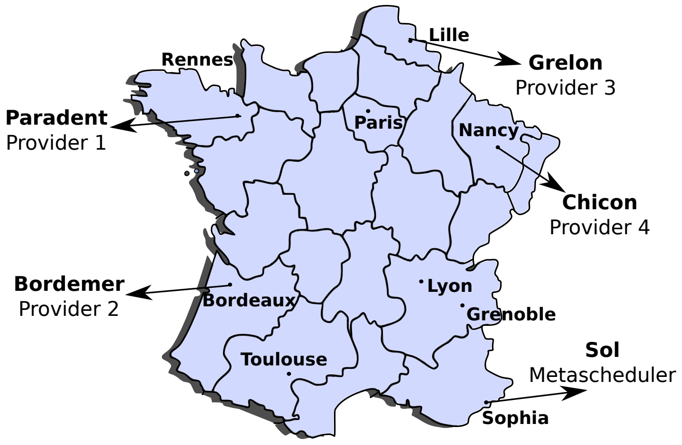
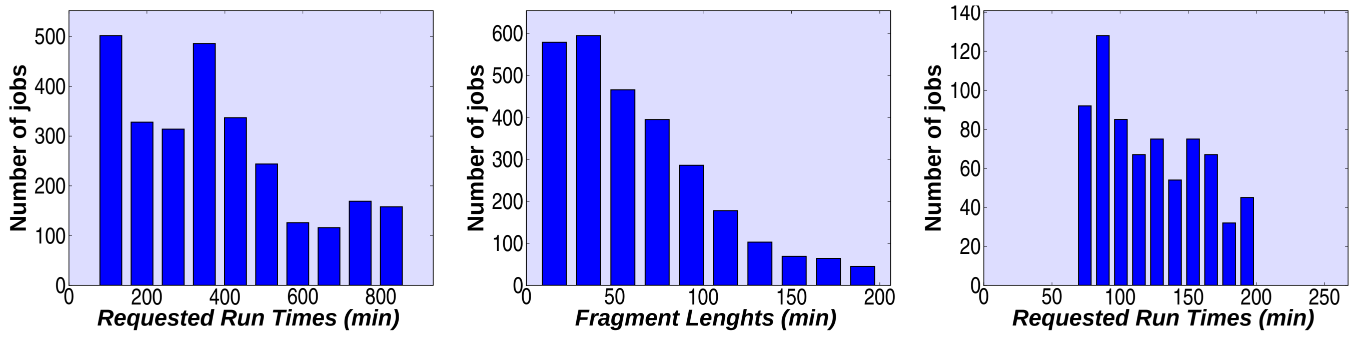


Fig. 7. Location of resources in Grid'5000.

The main motivation for developing the coordinated rescheduling for BoT applications is the observation that stretch factor increases with the run time overestimations. Figure 11 presents the stretch factor for applications scheduled in multiple clusters as a function of run time overestimation for homogeneous and heterogeneous environments. Until 30% of overestimation, there is no difference between the rescheduling strategies. This happens because by this value, just a few jobs have chances of backfilling. However, after 30%, tasks of BoT applications spread over the scheduling queues due to the backfilling, thus increasing the stretch factor. The coordinated rescheduling minimises this effect in approximately 20% and 10% for homogeneous and heterogeneous environments respectively.

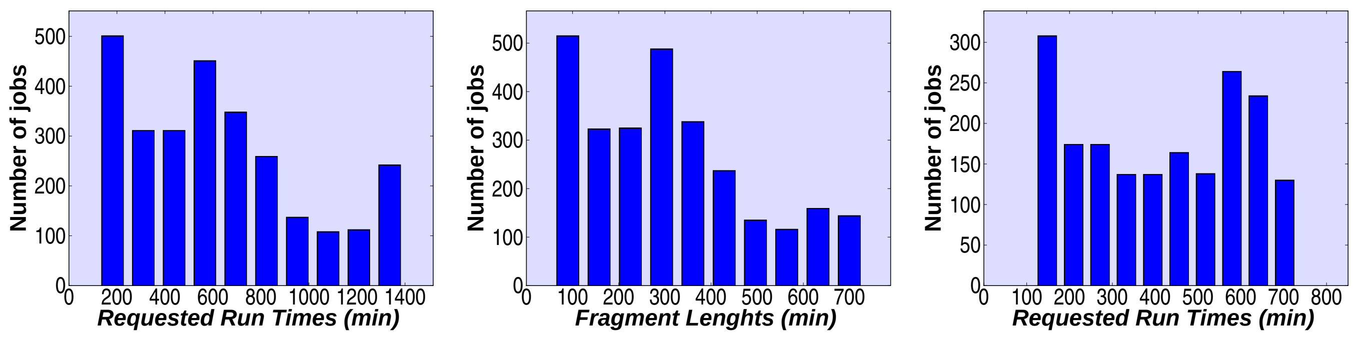


(a) Requested run times.

(b) Fragment lengths.

(c) Chances to be backfilled.

Fig. 8. Requested run times and fragment lengths of the workloads for accuracy of 85%.



(a) Requested run times.

(b) Fragment lengths.

(c) Chances to be backfilled.

Fig. 9. Requested run times and fragment lengths of the workloads for accuracy of 50%.

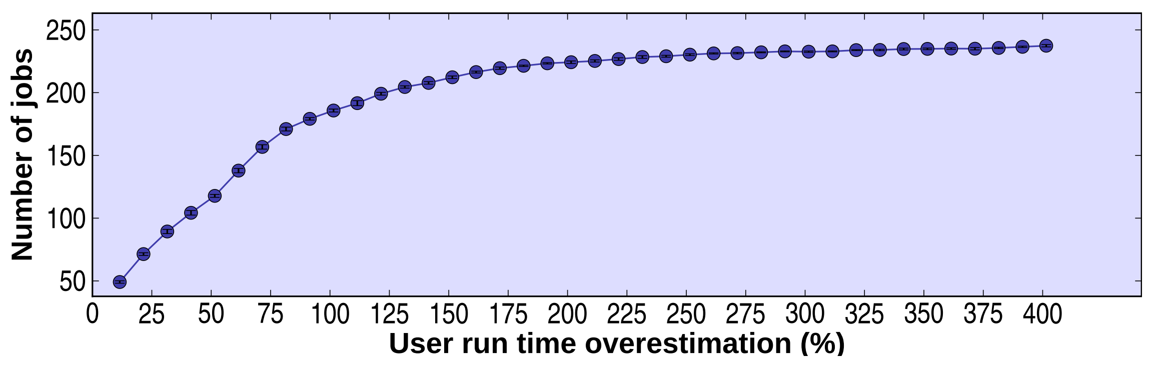
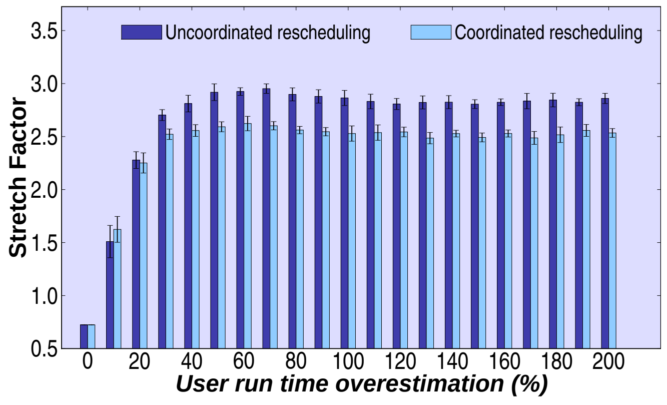


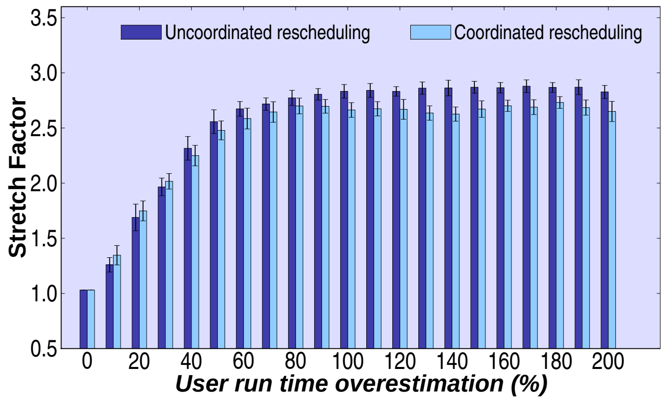
Fig. 10. Backfilling limit as a function of run time overestimations.

For the heterogeneous environment, although stretch factor is reduced using coordinated rescheduling over the uncoordinated one, this improvement is slightly lower (Figure 11). The reason is that applications tend to execute in fewer clusters (the fastest ones), and therefore the importance for coordinated rescheduling among providers is reduced. As showed in Figure 12, the number of clusters per job is reduced in the heterogeneous environment. Most of the applications are scheduled to one or two clusters, whereas for the homogeneous environment similar number of applications access two, three, and four clusters.

Reducing the stretch factor has a direct impact on the user response time. Figure 13 presents the response time reduction of coordinated rescheduling and system-generated predictions in comparison to user run time estimates with uncoordinated rescheduling. We observe that the differences between the policies is higher for the homogeneous environment, since jobs are more distributed to multiple providers than in the heterogeneous environment. In addition, system-generated predictions have better improvements in the heterogeneous environment than in the homogeneous one. The reason is that incorrect balancing the load in a heterogeneous environment causes more negative effects than in a homogeneous one. We also observe that system-generated

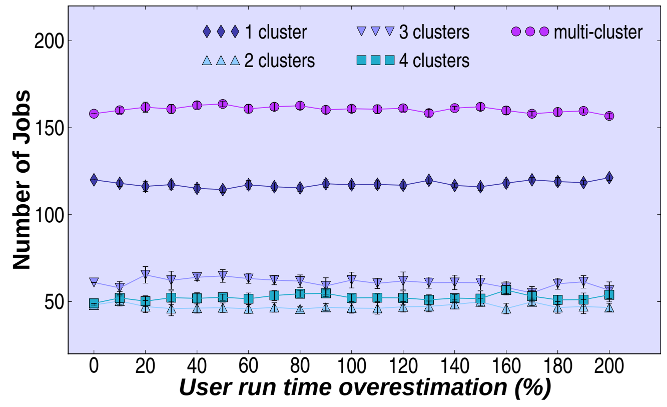


(a) Homogeneous environment.

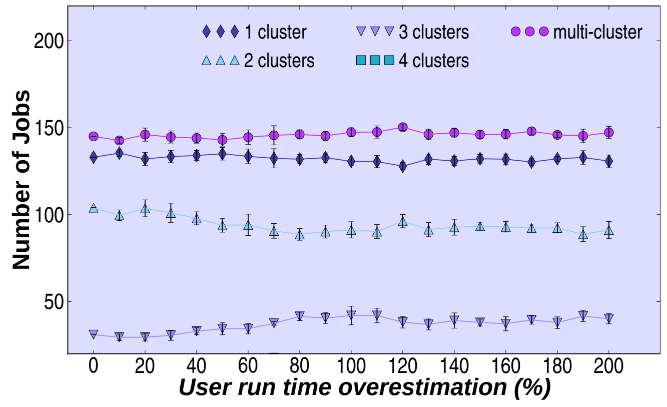


(b) Heterogeneous environment.

Fig. 11. Stretch factor variation as a function of the run time estimation accuracy and rescheduling schema.



(a) Homogeneous environment.



(b) Heterogeneous environment.

Fig. 12. Number of clusters per job.

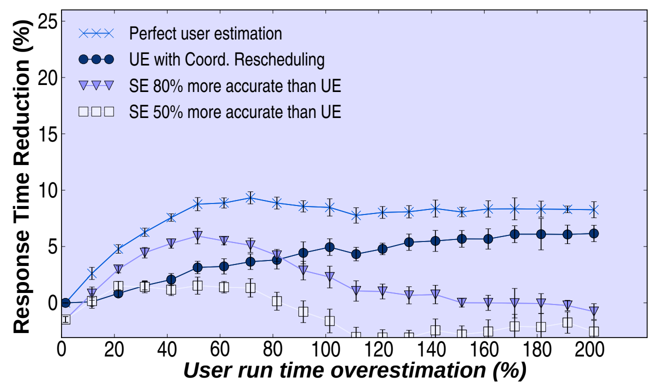
prediction policies have a similar curve shape that perfect user estimation until a certain threshold (60% for homogeneous and 70% for heterogeneous environment). After this threshold the advantage of using system-generated predictions is reduced. This happens because there is a benefit limit in backfilling (as illustrated in Figure 10) and it becomes lower than the cost paid to obtain better estimations.

We also analysed the user response time separately for multi- and single-provider jobs. Figure 15 presents the results for single-provider jobs. We observed that the increase of user overestimations actually reduces user response time for these jobs, which corroborates with previous studies on effects of run time estimates for job scheduling [30]. User response time for coordinated rescheduling produces an improvement of up to 5% in relation to uncoordinated rescheduling for these jobs. The main benefits of higher run time accuracy and coordinated rescheduling come from multi-provider jobs, as illustrated in Figure 16.

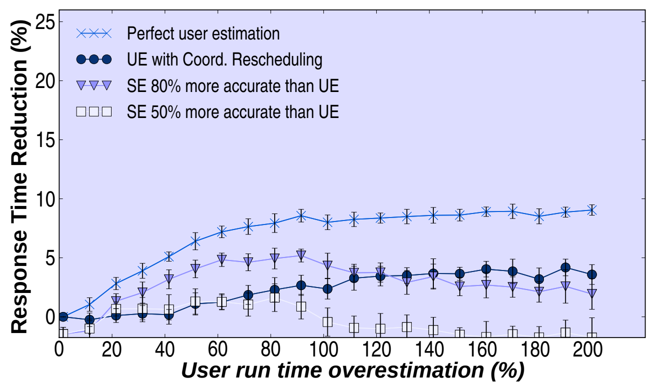
We have also analysed the slowdown (with 10 minutes bound), which is the response time divided by the application run time. Figure 14 presents the slowdown for homogeneous and heterogeneous environments. We observe that for this metric, coordinated rescheduling presents even better results than using perfect run time estimations. This happens because this metric highlights the improvements of smaller jobs in relation to big ones—smaller jobs have more chances of backfilling than the big ones.

We have also calculated system utilization for user and system-generated estimations and uncoordinated/coordinated rescheduling algorithms. The results are similar with a difference of less than 1%. This difference may increase if we consider a competition scenario with providers offering different levels of completion time guarantees. In this scenario, users will tend to execute their applications on providers with more optimized completion time guarantees. Figure 17 illustrates the average utilization level of providers with different run time estimation schemas; the higher the accuracy of run time predictions the higher the chances of attracting more users.

We have also performed experiments in Grid'5000 to evaluate possible technical difficulties to deploy the coordinated rescheduling. We selected a few workloads and

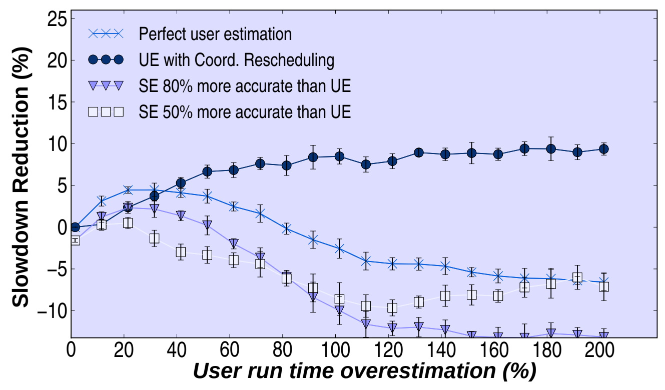


(a) Homogeneous environment.

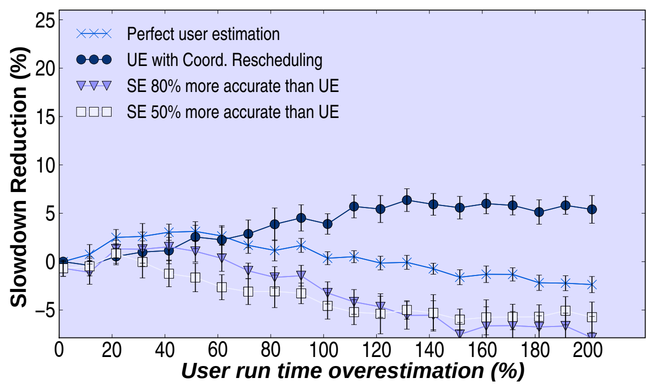


(b) Heterogeneous environment.

Fig. 13. Global user response time reduction in comparison to uncoordinated rescheduling.

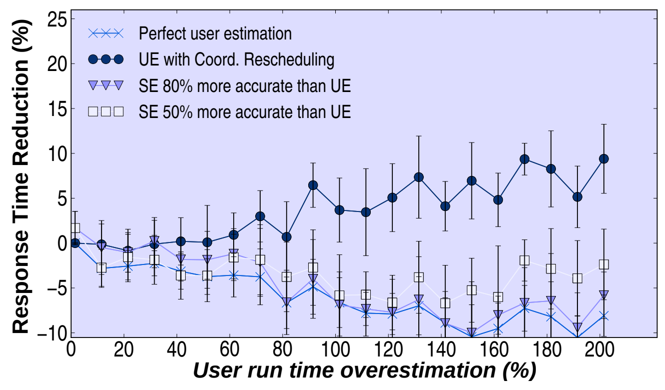


(a) Homogeneous environment.

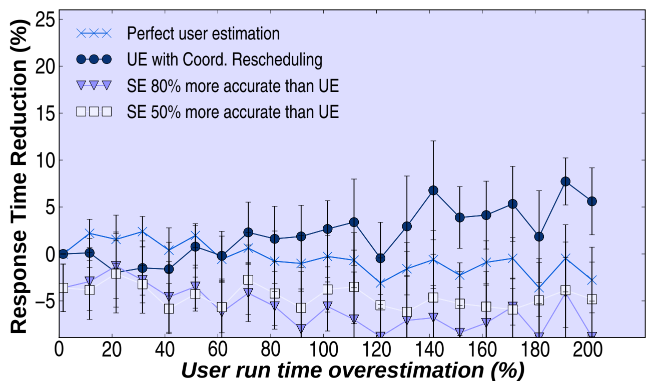


(b) Heterogeneous environment.

Fig. 14. Global slowdown reduction in comparison to uncoordinated rescheduling.



(a) Homogeneous environment.



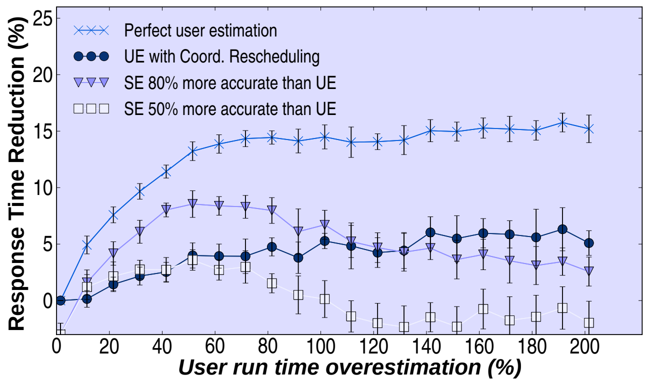
(b) Heterogeneous environment.

Fig. 15. User response time reduction in comparison to uncoordinated rescheduling for single-cluster jobs.

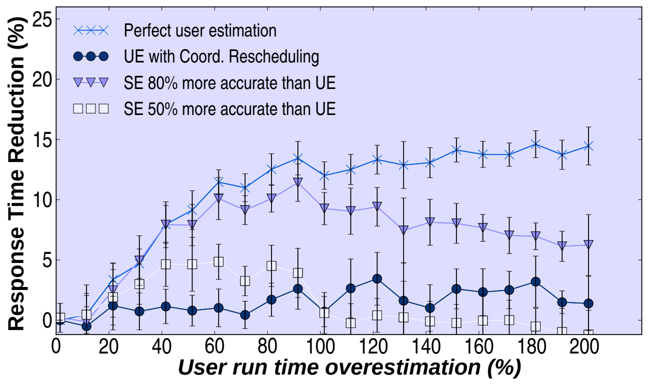
compared the results of simulation and the execution in the real system. Table 4 presents the scenarios and obtained results. We observed that for these experiments, both simulations and executions in the real environment provided similar results, showing the practical benefits of coordinated rescheduling. As we described in Section 3, the required modification in an existing scheduling architecture is minimal.

7 Conclusions

This paper presented a coordinated rescheduling algorithm for BoT applications executing across multiple

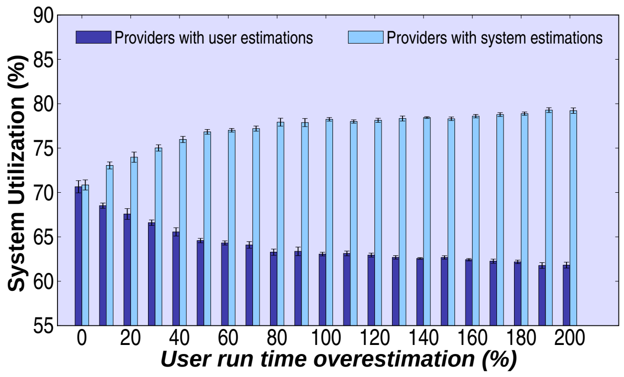


(a) Homogeneous environment.

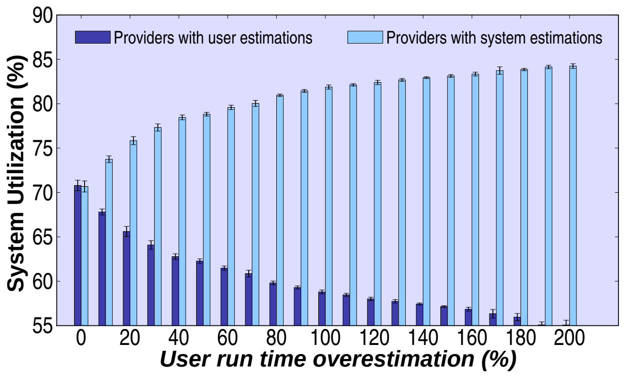


(b) Heterogeneous environment.

Fig. 16. User response time reduction in comparison to uncoordinated rescheduling for multi-cluster jobs.



(a) SE = 0.5 UE



(b) SE = 0.2 UE

Fig. 17. Impact of estimations on the system utilization by attracting more users through more optimized completion time guarantees.

TABLE 4
Comparison of results from Grid'5000 and simulations.

Metric and Overestimation (%)	**From simulation (%)**	**From real system (%)**
SFactor uncoord - 50	2.92 ± 0.08	3.05
SFactor uncoord - 100	2.86 ± 0.07	2.81
SFactor uncoord - 150	2.81 ± 0.04	2.69
SFactor coord - 50	2.59 ± 0.05	2.65
SFactor coord - 100	2.53 ± 0.07	2.42
SFactor coord - 150	2.49 ± 0.04	2.56
Response time red. - 50	3.14 ± 0.55	3.84
Response time red. - 100	4.95 ± 0.74	6.94
Response time red. - 150	5.68 ± 0.74	5.99
Slowdown red. - 50	6.66 ± 0.79	6.74
Slowdown red. - 100	8.48 ± 0.92	10.4
Slowdown red. - 150	8.88 ± 1.29	10.07

providers and the impact of run time estimates for these applications. Due to inaccurate run time estimates, initial schedules have to be updated, and therefore, when each provider reschedules tasks of a BoT application independently, other applications may not have chances of reducing their response time.

The main finding is that tasks of the same BoT can be spread over time due to inaccurate run time estimates and environment heterogeneity. Coordinated rescheduling of these tasks can hence reduce user response time for both single- and multi-provider applications in approximately 5%; and slowdown reduction of up to 10%. This improvement comes from the observation that reducing completion time of tasks from the same BoT independently prevents backfilling of other tasks. Moreover, in order to deploy coordinated rescheduling, metaschedulers and resource providers only require to keep track of the expected completion time of the last task of each BoT applications. System-generated predictions can serve as an alternative to coordinated rescheduling but require more effort for deployment and may not reduce user response times as much as when using coordinated rescheduling. From the providers perspective, one of the main advantages of system-generated predictions is to be able to attract more users by providing tighter expected completion times.

Acknowledgments

We thank Adam Barker and Marcos Dias de Assunção for their comments on this paper. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). We also thank Dr. Dror Feitelson for maintaining the Parallel Workload Archive, and all organizations and researchers who made their workload logs available. This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Dept. of Innovation, Industry, Science and Research (DIISR).

References

[1] V. S. Pande, I. Baker, J. Chapman, S. Elmer, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic, "Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing," *Peter Kollman Memorial Issue, Biopolymers*, vol. 68, no. 1, pp. 91–109, 2003.

[2] S. Smallen, H. Casanova, and F. Berman, "Applying scheduling and tuning to on-line parallel tomography," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'01)*. Denver, USA: ACM, November 10-16 2001.

[3] S. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool." *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990. [Online]. Available: http://dx.doi.org/10.1006/jmbi.1990.9999

[4] A. Auyoung, L. Grit, J. Wiener, and J. Wilkes, "Service contracts and aggregate utility functions," in *Proceedings of the 15th International Symposium on High Performance Distributed Computing (HPDC'06)*. Paris, France: IEEE, June 19-23 2006.

[5] M. A. S. Netto and R. Buyya, "Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems," in *Proceedings of the International Heterogeneity in Computing Workshop (HCW), in conjunction with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, California: IEEE Computer Society, 2009.

[6] C. B. Lee and A. Snavely, "On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 495–506, 2006.

[7] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'04)*, ser. Lecture Notes in Computer Science, vol. 3277. New York, USA: Springer, 2004, pp. 253–263.

[8] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[9] D. Nurmi, J. Brevik, and R. Wolski, "Qbets: Queue bounds estimation from time series," in *Proceeding of the 13th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, ser. Lecture Notes in Computer Science, vol. 4942. Springer, 2007.

[10] W. Smith, V. E. Taylor, and I. T. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Proceeding of the International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, ser. Lecture Notes in Computer Science, vol. 1659. Springer, 1999.

[11] L. T. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *Proc. of the ACM/IEEE SC'05*, 2005.

[12] H. A. Sanjay and S. S. Vadhiyar, "Performance modeling of parallel applications for grid scheduling," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1135–1145, 2008.

[13] G. Romanazzi and P. K. Jimack, "Parallel performance prediction for numerical codes in a multi-cluster environment," in *Proc. of the 2008 International Multiconference on Comp. Science and Information Technology (IMCSIT'08)*, 2008.

[14] S. A. Jarvis, D. P. Spooner, H. N. L. C. Keung, J. Cao, S. Saini, and G. R. Nudd, "Performance prediction and its use in parallel and distributed computing systems," *Future Generation Computer Systems*, vol. 22, no. 7, pp. 745–754, 2006.

[15] A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*. Geneva, Switzerland: IEEE Computer Society, 1997, pp. 209–218.

[16] O. O. Sonmez, N. Yigitbasi, A. Iosup, and D. H. J. Epema, "Trace-based evaluation of job runtime and queue wait time predictions in grids," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*. Garching, Germany: ACM, 2009, pp. 111–120.

[17] S. M. Sadjadi, S. Shimizu, J. Figueroa, R. Rangaswami, J. Delgado, H. Duran, and X. J. Collazo-Mojica, "A modeling approach for estimating execution time of long-running scientific applications," in *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*, 2008.

[18] L. He, S. A. Jarvis, D. P. Spooner, X. Chen, and G. R. Nudd, "Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids," in *Proceedings of the International Conference on Grid Computing (GRID'04)*, 2004.

[19] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. M. Figueira, J. Hayes, G. Obertelli, J. M. Schopf, G. Shao, S. Smallen, N. T. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369–382, 2003.

[20] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien, "Offline and online master-worker scheduling of concurrent bags-of-tasks on heterogeneous platforms," in *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'00)*. Miami, USA: IEEE Computer Society, April 14-18 2008.

[21] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker," *Future Generation Computer Systems.*, vol. 18, no. 8, pp. 1061–1074, 2002.

[22] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, and R. Joshi, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 154–169, 2007.

[23] A. Iosup, O. O. Sonmez, S. Anoep, and D. H. J. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *Proceedings of the 17th International Symposium on High-Performance Distributed Computing (HPDC'08)*. Boston, USA: ACM, 23-27 June 2008, pp. 97–108.

[24] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized versus distributed schedulers for bag-of-tasks applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 698–709, 2008.

[25] W. Cirne, D. P. da Silva, L. Costa, E. Santos-Neto, F. V. Brasileiro, J. P. Sauvé, F. A. B. Silva, C. O. Barros, and C. Silveira, "Running bag-of-tasks applications on computational grids: The MyGrid approach," in *Proceedings of the 32nd International Conference on Parallel Processing (ICPP'03), Kaohsiung, Taiwan*. IEEE Computer Society, 2003, pp. 407–.

[26] W. Cirne, F. V. Brasileiro, D. P. da Silva, L. F. W. Góes, and W. Voorsluys, "On the efficacy, efficiency and emergent behavior of task replication in large distributed systems," *Parallel Computing*, vol. 33, no. 3, pp. 213–234, 2007.

[27] M. H. Haji, I. Gourlay, K. Djemame, and P. M. Dew, "A SNAP-based community resource broker using a three-phase commit protocol: A performance study," *The Computer Journal*, vol. 48, no. 3, pp. 333–346, 2005.

[28] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105–1122, 2003.

[29] E. Shmueli and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," *Journal of Parallel Distributed Computing*, vol. 65, no. 9, pp. 1090–1107, 2005.

[30] D. Tsafrir and D. G. Feitelson, "The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help," in *Proceedings of the 2006 IEEE International Symposium on Workload Characterization (IISWC'06)*. San Jose, California, USA: IEEE, 2006, pp. 131–141.