Resource Co-Allocation based on Application Profiling: A Case Study in Multi-Objective Evolutionary Computations

Marco A. S. Netto[1], Christian Vecchiola[1], Michael Kirley[1], Carlos A. Varela[2], and Rajkumar Buyya[1]

[1]Grid Computing and Distributed Systems Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
{netto, csve, mkirley, raj}@csse.unimelb.edu.au

[2]Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
cvarela@cs.rpi.edu

Abstract

Several scientific experiments require coordinated resource allocation in multi-cluster environments. Rescheduling these experiments can reduce applications' response time and increase system utilization. However, existing allocation models, which are based on user run time estimates, limit rescheduling due to environment heterogeneity. This paper introduces a resource co-allocation model based on application profiling to enable automatic rescheduling in multi-cluster environments. Using a multi-objective evolutionary application as a case study, we demonstrate how to generate run time predictions and their impact on rescheduling. The evaluation is on Grid'5000; a large-scale platform comprising clusters with heterogeneous capabilities. From the evolutionary research field, this paper presents and compares the synchronous and asynchronous models for the target application. Our main findings are: it is possible to generate run time predictions to enable rescheduling by using a simple and practical approach with 7% error; and the asynchronous model for multi-objective evolutionary computations produces better optimization results and is faster than its synchronous counterpart.

1 Introduction

The use of multiple clusters for scientific experiments is fundamental to reduce application's response time. Not only large-scale experiments require multiple clusters, but also small and medium size experiments enjoy better response times by grouping resources that would be otherwise wasted due to fragmentation in scheduling queues [8]. Message passing parallel applications and workflows require coordinated access to these resources; a problem known as *resource co-allocation* [3]. Bag-of-Tasks applications may also require co-allocation as scientists need the completion of all tasks to post-process or analyze the results [13].

Applications waiting for resources may need to be rescheduled due to inaccurate usage estimations, request cancellations and modifications, and resource failures. Rescheduling of both single and multi-cluster applications reduces their response time and increases system utilization [12]. However, current allocation models limit rescheduling of multi-cluster applications due to environment heterogeneity. As these models are based on users specifying the estimated usage time for each cluster, applications may be aborted when rescheduled to slower resources; unless users provide high run time overestimations. When applications are rescheduled to faster resources, backfilling [11] may not be explored if estimated run times are not reduced. Therefore, system-generated predictions [16] can play an important role for automatic rescheduling.

This paper introduces a resource co-allocation model based on application profiling. Schedulers use application profiling to predict execution times for a given a set of resources. These predictions assist schedulers to check whether the execution of an application can fit into the clusters' scheduling queues during the rescheduling phase. We use a Parallel Multi-Objective Evolutionary Application as a case study and perform experiments on the Grid'5000 platform, which comprises clusters with heterogeneous capabilities. This paper contributes to two research fields:

- **Resource Co-allocation**: We show a simple and practical approach to generate run time predictions for a multi-cluster application and evaluate their impact on rescheduling. Predictions rely on observing the application's behavior with a short partial execution in each cluster. This is particularly possible for iterative applications with regular execution steps such as genetic algorithms and simulated annealing.

- **Multi-Objective Evolutionary Computations:** We introduce a framework to enable the execution of both

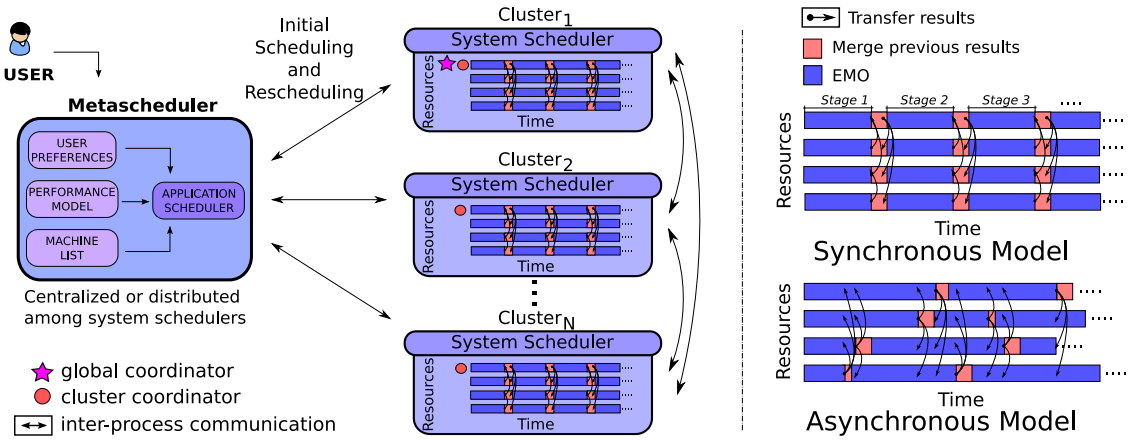


Figure 1. Deployment of EMO in multiple clusters using synchronous and asynchronous models.

synchronous and asynchronous models for the target application, and compare these models using execution time and quality of the optimization results as metrics. Results show that the asynchronous model for multi-objective evolutionary computations produces better optimization results and is faster than its counterpart synchronous model. The investigation of techniques to speed up these computations is important for several fields in science and engineering.

This work is a step towards more dynamic scheduling approaches for scientific experiments requiring multiple clusters with guaranteed response times. The use of application profiling presented here can also be applied when rescheduling single-cluster applications among multiple clusters.

2 Application Description

The case study application is EMO (Evolutionary Multi-objective Optimizer), which uses Genetic Algorithms [4] and introduces the concept of topology to drive the evolutionary process [10]. A **topology** is a graph interconnecting individuals of a population and is characterized by: (i) the node degree representing the average number of connections for each individual; and (ii) the path line defining the number of hops to be crossed on average to connect individuals. Individuals are chosen to exchange their information according to the topology links; process that determines how the current solutions of the approximation sets are selected to produce a new generation of solutions. The use of topologies provides better solutions in general but requires a large number of elements in the approximation sets, which becomes compute intensive for non-trivial problems. Moreover, topologies have impact on the execution time: sparsely connected topologies imply faster execution times than fully connected ones, but propagate more slowly the updates in the approximation set.

An initial implementation of EMO was provided for a single machine [10], whereas a distributed deployment was developed to reduce the computation time for non-trivial problems [17]. This work introduces two execution models for heterogeneous multi-cluster environments: synchronous and asynchronous (Figure 1).

EMO is an application that runs from the operating system shell and can be controlled by a set of 25 command line parameters. The main input parameters for our case study are: the topology used to produce the new solutions, the number of iterations, the size of the approximation set, and the optimization multi-objective function. As execution outcome, EMO produces: the final solution set (*Pareto Front*) and the approximation set that generated this solution. For the distributed version of EMO, we introduced a coordination layer that reiterates EMO processes by feeding them with updated information on the approximation set. An additional component, called EMOMerge, was used to merge and partition the approximation sets generated at each iteration of EMO processes.

In a multi-cluster environment, the clusters' computing power controls the upper bound execution time of each iteration and introduces gaps into the schedule of the EMO processes. In order to address this issue, we developed two new deployment models for EMO: synchronous and asynchronous (Figure 1). Note that as processes execute in multiple clusters, the impact of wide-area communication has to be minimized as much as possible [1]. Indeed, asynchrony masks communication and computation, and therefore minimizes this impact. **Resource co-allocation** is important for both models since: for the synchronous model, it prevents processes from being idle and thus completes the execution faster; whereas for the asynchronous model, it increases interaction among results of EMO processes executing with different topologies.

Epsilon Indicator. Multi-objective functions identify a multi-dimensional space whose properties are difficult to visualize effectively. It is then necessary to adopt synthetic measures that generally aggregate information about the quality of a solution into one number called indicator. In our case study, which has been configured with a 10-objective optimization function, we use a quality indicator called *Epsilon* [19]. This indicator is based on the distance between a reference solution and the *pareto front*. The indicator is calculated after the execution of the application. As future work we will also use the indicator to control the merge and split operation performed by the EMOMerge component.

2.1 Synchronous model

In this model, EMO processes are distributed to machines and merged once they have completed the number of iterations specified by the user (Figure 1). We call *stage* a set of processes followed by a merging process. When a stage finishes, its results are redistributed to the machines, which execute the next processes. The execution completes when all processes achieved the total number of iterations specified by the user. As topologies and machines are heterogeneous, processes finish at different times. In order to avoid idle processor time, we keep iterating all processes in a stage until they reach the minimum number of required iterations. Therefore, EMO processes running on faster machines and/or using sparsely connected topologies iterate more than the others.

Each cluster involved in the execution has a master node, named **cluster coordinator**, responsible for merging the results of nodes in its cluster. This coordinator sends the results to the cluster coordinator with better aggregate CPU, called **global coordinator**, which merges all the results and sends the merged result to all clusters. After that, EMO processes start the new stage (Figure 1).

2.2 Asynchronous model

For this model, when an EMO process finishes, it distributes its results to other EMO processes asynchronously, merges its results with the last results from other processes, and continues its execution (Figure 1). This prevents any idle time, and provides better support for heterogeneous machines and processor fault tolerance.

3 Profiling-based Resource Co-allocation

Cluster management systems rely on system schedulers responsible for scheduling requests. A co-allocation request is therefore a set of single cluster requests, called **sub requests**, which start in a coordinated fashion; in our case, they start at the same time. These sub requests mainly comprise the number of resources, such as nodes or cores, and the usage time for each cluster. As users can access heterogeneous resources, it is difficult for them to know the usage time that needs to be allocated. From the system schedulers' side, it is difficult to reschedule sub requests since all of them should have the same start and completion time in all clusters, and they have to fit into the scheduling queues. Therefore, in order to simplify the scheduling and rescheduling processes, we introduce a resource co-allocation model based on application profiling.

A resource co-allocation request based on application profiling can be re-targeted to different resource sets in an automatic fashion. This is particularly important when rescheduling applications on multiple clusters. Note that rescheduling is allowed when requests are still in the waiting queues, and hence is different from migrating processes at run time. The profiling-based co-allocation request relies on four components, which can be located in a **metascheduler** responsible for allocating resources from multiple clusters (Figure 1):

- **Machine list:** contains the resources to be used by the application and is provided by system schedulers to the user or to the metascheduler.
- **Performance model:** users define the performance of the machines in terms of execution time, and the cost of inter-process communication. Processes can be on the same node, different nodes, or different clusters. Section 4 details how the model can be built.
- **User preferences:** users can specify application parameters such as the number of processes, number of objectives to be minimized, and other parameters as described in Section 2.
- **Application Scheduler:** uses the machine list, performance model, and user preferences to generate a schedule of the application' processes. This component generates a set of scripts used to deploy the application. It is outside the application's code; different from the application-level scheduler presented by Berman et al. [2].

The **system scheduler** uses the application scheduler to obtain the application's estimated run time. This estimation can be used for both the initial scheduling and the rescheduling.

3.1 System and application's scheduler interaction

The interaction between system and application's schedulers takes place during the initial scheduling or rescheduling of a request. The **initial scheduling** is triggered by a user or a metascheduler requesting for machines, whereas the **rescheduling** is triggered by the system scheduler or a meta-scheduler that is coordinating a co-allocation request.

For both cases, interaction between system and application's scheduler comprises three steps:

1. The application scheduler asks the system scheduler for the earliest n machines available;
2. The application scheduler generates a schedule containing the application's estimated execution time;
3. The metascheduler, or a system scheduler, verifies with the other system scheduler(s) whether it is possible to commit requests;
4. Step 1 is repeated if it is not possible to commit requests. A maximum number of trials can be specified.

Note that by using this algorithm, resource providers can keep their schedules private. Alternatively, in Step 1, the application scheduler could ask system schedulers for all free time slots (which are available time intervals for resources) and then minimize interactions between the metascheduler and the application scheduler. Elmroth and Tordsson discuss the advantages and drawbacks of keeping schedules private or asking for free time slots [7].

3.2 Application schedulers

We developed two application schedulers for EMO, one for synchronous and the other for the asynchronous model. The input arguments for them are: list of resources, number of iterations per process, number of processes, number of topologies, performance model for computation and interprocess communication. For the synchronous model (Algorithm 1), the scheduler sorts resources by their computing power, which is determined during the application profiling phase (Lines 1) and assigns EMO processes according to their topologies (more CPU consuming ones first) for each stage (Lines 6-12). The number of stages is determined by the total number of resources and total number of EMOs specified by the user (Line 3). As machines and topologies make the process execution times vary, more iterations are added to processes that would be waiting for slower processes (Lines 13-14). For the asynchronous model (Algorithm 2), the scheduler assigns one process of each topology to the resource with the earliest completion time (Lines 3-6). The scheduler gives priority to topologies that require more CPU (Line 1), however each topology receives one resource per round (Line 4).

4 Evaluation

The experiments evaluate the run time predictions and their impact on rescheduling co-allocation requests. In addition, we compare the synchronous and asynchronous models of EMO and the importance of using multiple topologies to improve optimization results.

Algorithm 1: Pseudo-code for generating the schedule using the synchronous model.

```
1  Sort resources by decreasing order of computing power
2  Sort EMO processes by their topologies. Decreasing order
   of CPU demand
3  Stages ← nProcs / nResources
4  ProcsPerTopology ← nResources / nTopologies
5  MaxCompletionTime ← 0
6  for each stage do
7      r ← 0
8      for each topology do
9          for 0 to ProcsPerTopology do
10             Schedule EMO of this topology to resource r
11             Update MaxCompletionTime
12             r ← r + 1
13     for each resource do
14         Make last EMO process on this resource complete
           at MaxCompletionTime by increasing the
           number of iterations
```

Algorithm 2: Pseudo-code for generating the schedule using the asynchronous model.

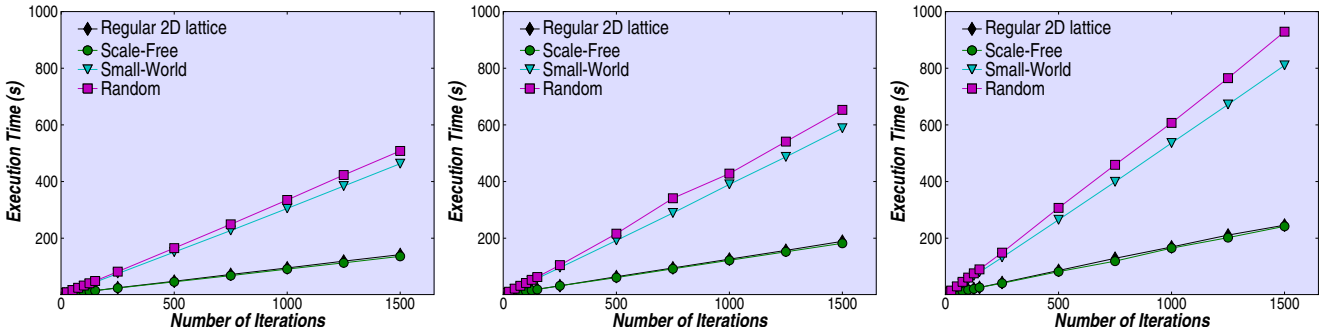
```
1  Sort topologies by decreasing order of CPU demand
2  n ← 0
3  while n < nProcs do
4      for each topology do
5          Select resource r with earliest completion time
           Schedule EMO process to resource r
6      n ← n + nTopologies
```

4.1 Experimental configuration

The evaluation has been performed in Grid'5000, which consists of a set of clusters across France dedicated to large-scale experiments. We used seven clusters located in three cities in France with different computing capabilities. Table 1 presents an overview of the node configurations, including cores per node, for these clusters[1]. Machines in the same location share the same file system, which simplifies file transfer between nodes of clusters in the same site.

Application configuration. We configured EMO for solving the DTLZ6 function with a setup of 10 objectives. This function is one of the most compute intensive functions in the benchmark proposed by Deb et al. [5]. We have used four topologies: Regular 2D, Scale-Free, Small-World, and Random [10]; and 1024 individuals for each EMO process with a minimum of 200 iterations each process. The application was deployed on 40 cores using 480 EMO processes, i.e. 120 processes for each topology in order to optimize 10 objective functions.

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



(a) Paramount. (b) Bordemer. (c) Bordeplage.

Figure 2. Execution times as a function of the topologies for three machine types in Grid'5000.

Table 1. Overview of the node configurations.

Cluster	Location	CPUs' Configuration	Cores per Node
azur	Sophia	AMD Opteron 246 2.0GHz	2
sol	Sophia	AMD Opteron 2218 2.6GHz	4
bordemer	Bordeaux	AMD Opteron 248 2.2 GHz	2
bordeplage	Bordeaux	Intel Xeon EM64T 3GHz	2
paraquad	Rennes	Intel Xeon 5148 2.33 Ghz	4
paramount	Rennes	Intel Xeon 5148 2.33 Ghz	4
paradent	Rennes	Intel Xeon L5420 2.5 Ghz	8

Application profiling. We executed the stand alone version of EMO on a single core of each cluster using four topologies varying the number of iterations. EMO has a predictable behavior as we can see in Figure 2, which shows the execution times as a function of the topologies and number of iterations for three machine types in Grid'5000. One important decision when profiling the application is to choose the number of iterations EMO needs to be executed in order to capture the application's behavior. Figure 3 represents the throughput (iterations/second) of an EMO execution using the Regular 2D topology on a single core of seven machine types as a function of number of iterations. After 250 iterations, the throughput becomes steady. Therefore, by executing 250 iterations, we can identify the application's behavior in each machine type. We obtained the same behavior for the other three topologies. Table 2 contains the performance model, in our case EMO's throughput, for each cluster and topology. We observe that sparsely connected networks, such as the Regular 2D and the Scale-Free networks, imply a faster iteration time than more connected networks, such as the Random topology. For the Random topology, the value of the path line was around five times smaller than the path line of the Regular 2D. This means that, on average, the selection of the individual to exchange information requires traversing a list five times smaller than for the Random topologies; and this reflects the execution time difference.

Inter-process communication profiling. Communication is based on file transfer between EMO processes during the merging phases. The files transferred among the sites are

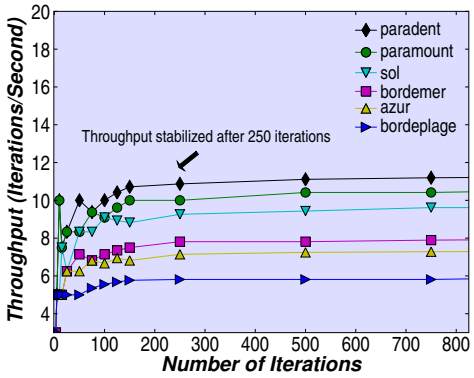


Figure 3. Throughput for Regular 2D topology on each machine type.

Table 2. Performance model (iterations/sec.) of each machine type and topology.

	Regular 2D	Scale-Free	Small-World	Random
paradent	10.87	11.36	3.57	3.29
paramount	10.00	10.42	3.33	3.05
paraquad	10.00	10.42	3.33	3.05
sol	9.26	9.62	3.09	2.81
bordemer	7.81	7.81	2.60	2.38
azur	7.14	7.35	2.34	2.14
bordeplage	5.81	6.10	1.89	1.68

500 Kbytes on average. Therefore, the cost of transferring the files is minimum compared to EMO's execution, which takes minutes. However, file transfer relies on secure copy (*scp*) command, which requires authentication. Therefore, inter-site file transfer is around 800 ms.

Resource sets. We executed the EMO application using synchronous and asynchronous models on seven clusters in Grid'5000. Table 3 presents the list of clusters and number of cores used in each resource set.

Metrics. To evaluate co-allocation based on application profiling and its importance on rescheduling, we analyze the *difference between actual and predicted execution times*. The prediction for each resource set assists schedulers to

know whether they can reschedule the new sub requests into the scheduling queues of other schedulers. For comparison between the synchronous and asynchronous models, we use *execution time* and the *Epsilon indicator* (Section 2).

Table 3. Resource sets on seven clusters.

Clusters/Resource Sets	1	2	3	4	5	6	7
paradent	32						
paramount	04		20			04	
paraquad	04	20					
sol		12	12	20	20	12	
bordemer		02	08	20	08	06	20
azur		06			06	10	
bordeplage					06	08	20

4.2 Results and Analysis

Execution time predictions. Figure 4 presents predicted and actual execution times for synchronous and asynchronous models. Actual execution times are averages of five executions for each resource set. We observe that execution time for the asynchronous model is shorter than the synchronous model for all resource sets. For the asynchronous model, all EMO processes execute the minimum required number of iterations, whereas for the synchronous model, EMO processes may execute more iterations in order to wait for processes that take longer. In addition, the difference between actual and predicted execution is on average 8.5% for synchronous and 7.3% for the asynchronous model. These results highlight that (i) it is possible to reschedule processes on multiple clusters since schedulers can predict the execution time for different resource sets; and (ii) the prediction for the asynchronous model is slightly better than for the synchronous model since the merging phases take less time for the first model.

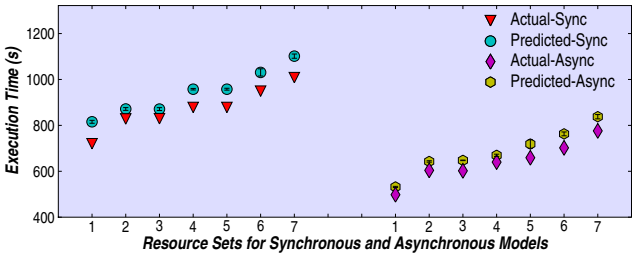


Figure 4. Predicted execution times.

For the quality of the predictions, resource sets 2 and 3 present better results compared to the other sets for the synchronous model. This happens because the merging phase is split by sites (locations in France). For these sets, three sites are used, and therefore the load for merging results is well balanced. Resource set 6 also comprises three sites, but only four resources in one of the sites. For the asynchronous model, the worst prediction is for resource set 7 since 20 resources from the worst cluster (*bordeplage*) are used, which makes the merging process slower.

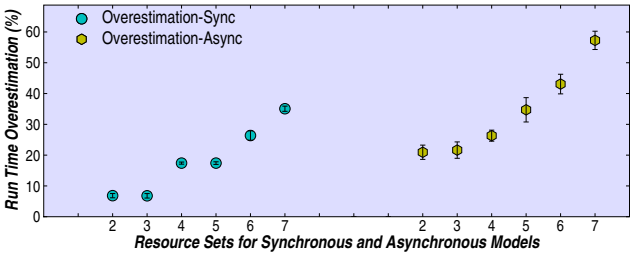
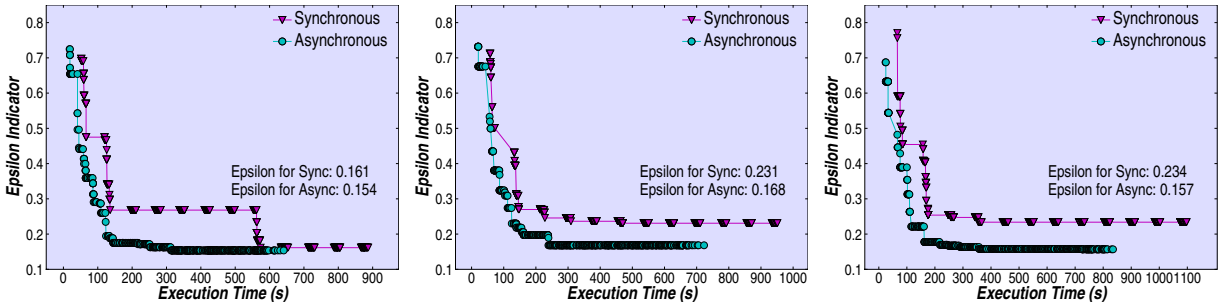


Figure 5. Overestimation required to avoid application being aborted due to rescheduling from resource set 1 to other sets without co-allocation based on application profiling.

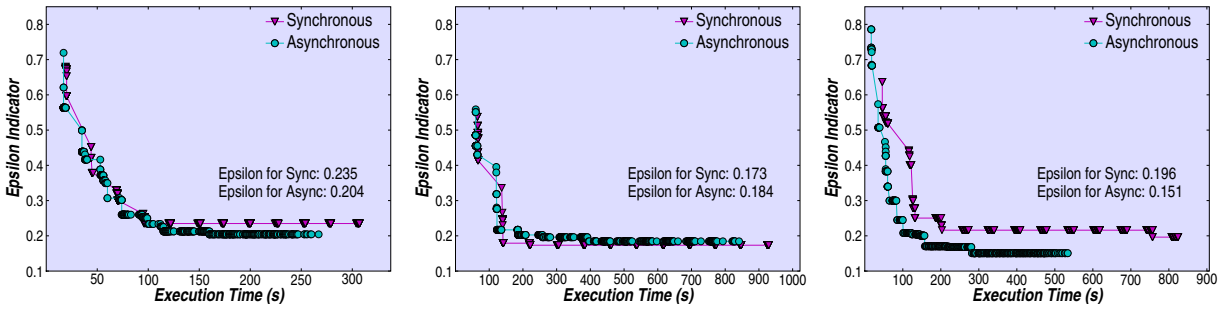
Importance of predictions to rescheduling. When remapping processes from one resource set to another, the application run time may remain the same, increase or decrease. When it remains the same, schedulers just have to redefine the number of resources in each cluster; which can be performed by the metascheduler or by the system schedulers themselves. This is the case for remapping processes from, for example, resource set 1 to 2 and 2 to 3 or 4 for synchronous and asynchronous model respectively. When the run time increases, the prediction generated by the application-scheduler may avoid the application to be aborted due to underestimations. A rescheduling from a shorter to longer execution time is desired when the application can start earlier than expected. This is the case for remapping processes from resource set 1 to 7. Figure 5 shows that overestimation is required to avoid the application to be killed when rescheduling from resource set 1 to the other resource sets. For the synchronous model, 35% of overestimation is required, whereas for the asynchronous model 57%. When rescheduling a request from a longer to shorter run time, predictions assist schedulers to increase the chances of backfilling sub requests [11]. This is the case when rescheduling processes from resource set 7 to 1 for synchronous and asynchronous model.

Synchronous versus asynchronous models. In order to understand the output produced by the application, we have also compared the quality of the optimization results between synchronous and asynchronous models. Figure 6 shows the Epsilon indicator (the lower the better) for synchronous and asynchronous models under three resource sets. The asynchronous model reaches faster and produces better results than the synchronous model. This happens because the asynchronous model is able to mix more results from different EMO processes, which might have different topologies, in relation to the synchronous model. For resource set 3, the synchronous model produces similar result for the Epsilon indicator as the asynchronous model, but the Epsilon values get closer after a considerable execution time. Figure 7 illustrates the importance of mixing results



(a) Resource set 3. (b) Resource set 5. (c) Resource set 6.

Figure 6. Epsilon indicator for three resource sets using synchronous and asynchronous models.



(a) Only Scale-Free topology. (b) Only Random topology. (c) Four topologies together.

Figure 7. Epsilon indicator for resource set 1 showing the importance of mixing topologies.

from different topologies. The results show that although Random, which is the most CPU consuming topology, has the greatest impact on the Epsilon indicator, the less CPU consuming Scale-Free topology contributes to the function optimization. Moreover, even for one-topology executions, asynchronous produces better optimization results and it converges faster than its synchronous counterpart. Similar results were obtained for the other resource sets but are not included due to space constraints.

5 Related Work

Yang et al. [18] introduced a performance translation based on relative performance between two platforms. Similar to our work, their approach relies on a short execution of the application in each target platform in order to predict its execution time. The main difference is that their work is for applications running in a single cluster.

Sanjay and Vadhiyar [15] developed a set of performance modeling strategies to predict execution times of parallel applications for single-cluster executions. As their target platform comprises non-dedicated clusters, their strategies require more and longer executions of the application and rely on more parameters. For our application, we just need a short execution for each topology on each target cluster.

Jarvis et al. [9] studied performance prediction of applications using their PACE (Performance Analysis and Characterization Environment) toolkit. Their model relies on source code analysis, which differs from our approach. Tsafrir et al. [16] introduced system-generated predictions for single-cluster applications. Their technique relies on previous execution of applications, whereas ours relies on partial short executions.

Closer to our work, Romanazzi and Jimack [14] 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. The main difference of their work is that the application has a different structure in terms of computation and communication model and the predictions are used only for scheduling, and not for rescheduling.

Regarding our application, the closest work is the asynchronous model for genetic search developed by Desell et al. [6]. The comparison results between synchronous and asynchronous model showed in this paper corroborate the results for their application in the astronomy field; i.e. asynchronous model has better convergence rates, especially when heterogeneous resources are in place.

6 Conclusions

The coordinated use of multiple clusters for scientific experiments speeds up executions and reduces resource access time. However, current allocation models limit rescheduling of these experiments, which is important due to necessary updates in clusters' scheduling queues. In this work we have introduced a resource co-allocation model based on application profiling. Predictions enable automatic rescheduling of parallel applications; in particular they pre-

vent applications from being aborted due to run time underestimations and increase backfilling chances when executing in faster resources.

From the application's side, we have developed a framework to execute synchronous and asynchronous models, and compared them using execution time, prediction feasibility, and quality of optimization results as metrics. The asynchronous model performs better for all these metrics in relation to the synchronous model. This is a result of faster interactions between multiple EMO processes, in particular those with different topologies. We have also shown the impact of merging results from multiple topologies on optimization results.

Thus, our main findings are: it is possible to generate run time predictions to enable rescheduling of a multi-cluster application using a simple and practical approach with 7% error; and the asynchronous model for multi-objective evolutionary computations not only executes faster but also produces better optimization solutions than its synchronous counterpart. As future work, we will investigate rescheduling for other multi-cluster applications.

Acknowledgments

We thank Marcos Dias de Assunção and Mukaddim Pathan 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). This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Dept. of Innovation, Industry, Science and Research (DIISR). It is also partially supported by U.S.A. NSF CAREER CNS Grant No: 0448407.

References

[1] H. E. Bal, A. Plaat, M. G. Bakker, P. Dozy, and R. F. H. Hofman. Optimizing parallel applications for wide-area clusters. In *Proc. of the 12th International Parallel Processing Symposium / 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98)*, 1998.

[2] 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*, 14(4):369–382, 2003.

[3] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *Proc. of the 8th HPDC'99*, pages 219–228, Redondo Beach, USA, 1999.

[4] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[5] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multiobjective optimization. *Evolutonary Multiobjective Optmization*, 2005.

[6] T. J. Desell, B. K. Szymanski, and C. A. Varela. Asynchronous genetic search for scientific modeling on large-scale heterogeneous environments. In *Proceedings of the 17th Heterogeneity in Computing Workshop (HCW'08), in conjuction with 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*, 2008.

[7] E. Elmroth and J. Tordsson. A standards-based grid resource brokering service supporting advance reservations, coallocation and cross-grid interoperability. *Concurrency and Computation: Practice and Experience (to appear)*, 2009.

[8] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *Proc. of the 2nd CCGrid*, pages 39–, Berlin, Germany, 2002.

[9] 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*, 22(7):745–754, 2006.

[10] M. Kirley and R. Stewart. Multiobjective evolutionary algorithms on complex networks. In *Proceedings of 4th International Conference Evolutionary Multi-Criterion Optimization, Lecture Notes Computer Science 4403*, 2007.

[11] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[12] M. A. S. Netto and R. Buyya. Rescheduling co-allocation requests based on flexible advance reservations and processor remapping. In *Proc. of 9th IEEE/ACM International Conference on Grid Computing (GRID'08)*, Tsukuba, Japan, 2008.

[13] M. A. S. Netto and R. Buyya. Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. In *Proc. of the 18th International Heterogeneity in Computing Workshop (HCW'09), in conj. with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, Rome, Italy, 2009.

[14] 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.

[15] H. A. Sanjay and S. S. Vadhiyar. Performance modeling of parallel applications for grid scheduling. *Journal of Parallel and Distributed Computing*, 68(8):1135–1145, 2008.

[16] D. Tsafrir, Y. Etsion, and D. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.

[17] C. Vecchiola, M. Kirley, and R. Buyya. Multi-objective problem solving with offspring on enterprise clouds. In *Proc. of the 10th International Conf. on High-Performance Computing in Asia-Pacific Region (HPC Asia'09)*, 2009.

[18] 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.

[19] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.