Scheduling Workflows in Multi-Cluster Environments

Silvio Luiz Stanzani[1], Líria Matsumoto Sato[1], Marco A. S. Netto[2]

[1]Computer and Digital Systems Engineering (PCS)
Polytechnic School of the University of São Paulo
São Paulo – Brazil
silvio.stanzani@usp.br, liria.sato@poli.usp.br

[2]IBM Research
São Paulo – Brazil
mstelmar@br.ibm.com

Abstract—**Scientific applications modeled as workflows can exhibit both task and data parallelism. Scheduling these workflows in a multi-cluster environment is challenging due to the large number of task mapping possibilities. Therefore, several heuristics have been proposed over the last years to address such a problem. A key limitation of existing heuristics for multi-cluster environments is that individual tasks are mapped onto single resources, which limits the resource options to reduce the time to the complete workflow executions. This paper introduces the Multi-Cluster Allocation-Heterogeneous Earliest Finish Time (MCA-HEFT) heuristic, which deploys single parallel tasks of a workflow into multiple clusters and schedules them accordingly. We evaluated MCA-HEFT against the Mixed-parallel Heterogeneous Earliest Finish Time (M-HEFT) heuristic, which is one of the most well-known workflow scheduling heuristics in literature. MCA-HEFT was able to produce makespans that were up to 42% shorter than those produced by M-HEFT, having only approximately 10% of tasks distributed on multiple clusters. Our experiments considered several metrics and parameters including critical path size, makespan, number of clusters used to execute tasks, and the network impact when deploying the tasks in multiple clusters.**

Keywords–Workflow scheduling; resource co-allocation; multi-cluster executions; HEFT; M-HEFT; Mixed parallel applications.

I. INTRODUCTION

Distributed infra-structures such as grids and clouds have been developed in order to enable the execution of scientific applications across multiple sites [1]. Over the last years, grid computing infra-structure composed of multiple clusters have been recognized as an infra-structure for efficient execution of parallel tasks based on the message passing paradigm.

Scientific applications are composed of parallel and sequential tasks which can be computationally intensive. Such applications can be organized as workflows in order to be scheduled, executed, and managed in a distributed infra-structure such as multi-clusters. In this sense, the task parallelism and data parallelism [2], [3] can be explored to improve the application's performance. The task parallelism is present at level of tasks of a workflow, and the data parallelism can be present in a single task of a workflow.

The scheduling of workflows is essential for an efficient execution of scientific applications. Such scheduling poses two challenges: the heterogeneous nature of resources and the task malleability [4], which imply that tasks can be executed on any number of resources with different execution time. Therefore, a number of heuristics have been developed [5] for scheduling workflows on heterogeneous infra-structures, most part of such heuristics is based on Heterogeneous Earliest Finish Time (HEFT) [6] which is one of the most popular and efficient heuristics.

One key aspect of scheduling workflows containing parallel tasks is deciding whether a single or multiple clusters should be used for their execution. This aspect has been investigated over the last years, in the context of two research areas namely resource co-allocation and workflow scheduling. Resource co-allocation [7] investigates the use of multiple clusters to execute a single parallel task. Research in this area has not considered such parallel tasks as part of a workflow containing task dependencies. For workflow scheduling, existing solutions restrict the execution of parallel tasks to a single cluster [8] [3]. This is due to the support of high interconnection speed, high bandwidth and low latency networks such as Infiniband [9], and the support of Network File System (NFS) which provides easy and transparent access to data from any of the parallel process.

The goal of this work is to leverage existing knowledge on resource co-allocation and workflow scheduling in order to allow the scheduling of workflow parallel tasks into multiple clusters. This brings the advantage of enabling the access to more resources, thus reducing overall user response time, even considering the low network communication performance across multiple clusters. Therefore, the contributions of this work are:

- A workflow scheduling heuristic, named Multi-Cluster Allocation-HEFT (MCA-HEFT), that considers the execution of single parallel tasks into multiple clusters.
- A detailed analysis on the impact of utilizing multiple clusters for the execution of single parallel tasks inside workflows, considering the number of tasks per workload level, critical path, and the inter-cluster network overhead.

II. Problem Description and Definitions

This section presents a definition of the multi-cluster computing environment, the workflow application model, and the problem our proposed workflow scheduling heuristic tackles.

A. Computational Environment and Application Model

The computational environment considered here is a multi-cluster grid environment. Each cluster is composed of a set of identical nodes managed by a batch scheduler. Such clusters can be deployed in the same administrative domain, or can be deployed in geographically dispersed administrative domains using a wide-area network.

The workflow applications we considered in this paper are modeled as a Direct Acyclic Graph (DAG), which is specified as $G = (N, E)$, where $N = \{n_i \mid i = 1, \dots, N\}$ is the set of vertices representing the graph tasks, and $E = \{e_{ij} \mid (i,j) \in \{1, \dots, N\} \times \{1, \dots, N\}\}$ is the set of edges representing the precedence constraints among tasks.

The tasks can be either sequential or parallel, and the parallel tasks can be either rigid or malleable. The sequential tasks require one resource, rigid tasks require a fixed number of resources, and malleable tasks can be executed with any number of resources, which are reserved to the task until its completion [4].

Each task t is defined by the following values $[comp_t, comm_t, mincores_t, Type_t, pf_t]$, where, $comp_t$ is the task t computational size in instructions, $comm_t$ is the amount of data to be transferred along with the execution of task t in Mega Bytes (MBs), $mincores_t$ represents the minimum number of cores required to execute the task t, pf_t is the parallel fraction of task t, and the value $type_t$ is used to identify the task t as sequential, rigid or malleable, $type_t = 1$ indicates sequential task, $type_t = mincores_t \neq 1$ indicates rigid task, and $type_t > mincores_t$ indicates malleable task.

The parallel tasks are composed of a percentage of sequential code and a percentage of parallelized code. The pf_t is the parallel portion of code. The execution time of pf_t varies according to the set of resources used.

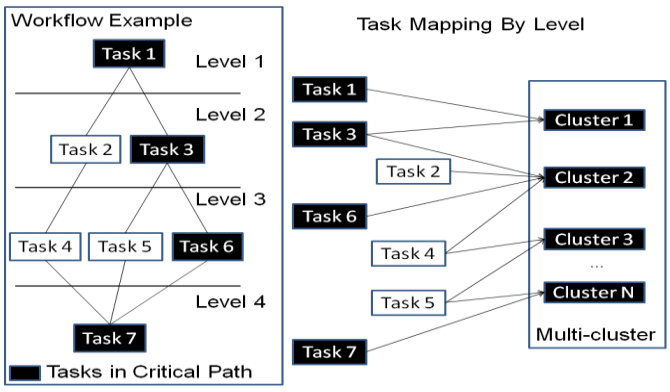

Figure 1. Scheduling Problem.

B. Problem Statement

The problem tackled by the proposed heuristic is the scheduling of workflows in multi-cluster environments with the objective of minimizing makespan. We consider the possibility of mapping single parallel tasks to multiple clusters as illustrated in Figure 1. We optimize makespan as it is the most popular workflow scheduling optimization metric, which is defined as the time between the submission and the completion of the workflow.

III. MCA-HEFT Scheduling Heuristic

The objective of Multi-Cluster Allocation-HEFT (MCA-HEFT) is to schedule workflows in multi-cluster environments. MCA-HEFT is based on the HEFT list scheduling heuristic [6], which has two phases, namely the task prioritization and processor selection.

The priority function in HEFT defines the order in which the task will be scheduled. Such order is based on the upward rank, which is the critical path from a given task to the exit task. In the processor selection phase, HEFT maps each task to the resource that presents the lower Estimated Finish Time (EFT). The priority function in MCA-HEFT is also based on the upward rank, but the processor selection phase maps tasks to a set of resources with the lowest EFT.

We present the MCA-HEFT heuristic by describing its major components: the rank definition and the processor selection.

A. Rank Definition

The rank of a task t is based on its computational weight, which is defined as the sum of the task's computational and communication cost($comp_t$ *and* $comm_t$), with the $comp_t$ and $comm_t$ of successive tasks with the highest rank until the last task of the workflow, according to Equation (1).

$$rank(t) = (comp_t + comm_t) + \max(rank(succ(t))). \quad (1)$$

Where, succ(t) represents the successive tasks of task t to the last task of the workflow.

B. Processor Selection

The processor selection is performed in three steps: cluster allocation, resource allocation, and task mapping.

1) Cluster Allocation

In the cluster allocation step the MCA-HEFT heuristic defines whether a given task can be scheduled to multiple clusters or to a single one, based on a metric called $MCACCR$,

which is defined with basis on the Computation to Communication Ratio (CCR) and parallel fraction pf_t. The CCR is an estimate of the amount of computation performed along with communication, and the pf_t is used to estimate the CCR of the parallel portion of parallel task only. The $MCACCR$ of a task t is defined according to Equation (2):

$$MCACCR(t) = \left(\frac{comp_t}{comm_t}\right) * pf_t. \quad (2)$$

The threshold variable named MCA is based on the $MCACCR$ of a task. If the $MCACCR$ of a task is lower than MCA the task is executed in one cluster, if the task is greater than MCA the task can be executed in multiple clusters. MCA is defined empirically based on the network infra-structure.

2) Resource Allocation

In the resource allocation phase a certain amount of resources is allocated to task t. For each task, the allocation is performed as follows: first, $mincores$ resources are allocated to the task, and then new resources are allocated to the task according to threshold variables MCAMAX and MCAIMP.

The MCAMAX variable defines the maximum amount of resources that can be allocated to a single task. The resources considered are the total number of nodes in all the clusters.

The allocation process begins allocating $mincores$ to each task. Then, for each level, the total number of resources is divided among the tasks. Each task receives a number of resources proportional to the rank value according to Equation (3) where $totalnodes_c$ represents the total number of resources of cluster c. However, $MCAMC(t)$ of a task has to be lower than the MCAMAX.

$$MCAMC(t) = \left(\frac{rank(t)}{\sum_{j=1}^{totalTaskbyLevel} rank(j)}\right) * \sum_1^{clusters} totalnodes_c. \quad (3)$$

The MCAIMP threshold variable is a percentage which is used to define that in order to allocate a new resource to the task, the computational cost must be decreased by at least MCAIMP percentage. The computational cost of a task is defined in Equation (4) in terms of amount of instructions, and the execution time is defined in Equation (5) in seconds according to the estimated amount of instructions.

$$compCost(t, nodes) = \left(\frac{(comp_t * pf_t)}{nodes}\right) + comp_t * (1 - pf_t). \quad (4)$$

$$MCAET\ (t, nodes) = \frac{compCost(t,nodes)}{Min(\ avgcomp_c\ )}. \quad (5)$$

The $avgcomp_c$ is the computational power of the cluster c in MIPS and is calculated according to Equation (6), where $avgcomp_r$ is computational power of resource r.

$$avgcomp_c = \sum_1^{totalnodes_c} avgcomp_r\ /totalnodes_c. \quad (6)$$

The computational power considered $avgcomp_c$ is the lowest computational power among all clusters. The allocation of nodes is increased by one if $MCAET\ (nodes + 1)$ is lower than $MCAET\ (nodes)$ by at least MCAIMP percentage.

3) Task Mapping

The mapping is performed following the rank order. The tasks are mapped to the cluster which has an amount of free resources that is greater than or equal to the amount of resources allocated to the task. The clusters are verified in decreasing load order. The cluster load is defined by Equation (7), where, $usednodes_c$ is the amount of resources of cluster c which has already been allocated.

$$load(c) = \frac{avgcomp_c * usednodes_c}{totalnodes_c}. \quad (7)$$

In case no cluster has more free resources than the ones allocated to the task, MCA-HEFT can change the amount of resources allocated to the task or map the task to multiple clusters. If the MCACCR of a task is greater than MCA the task is mapped to multiple clusters, otherwise MCA-HEFT allocates the task to the least loaded cluster. After the task is mapped to the cluster the least loaded resources in the cluster are chosen.

IV. EVALUATION

We evaluate MCA-HEFT by comparing its produced makespans against the ones produced by M-HEFT. We chose M-HEFT because it is an efficient heuristic for scheduling workflows and is based on HEFT. To perform the experiments, we developed a simulator based on SimGrid toolkit version 3.6.2 [10] using its MSG framework. We used simulations, as they are a mean to study the scheduling heuristics in a repetitive and controllable environment.

A. Experiment Setup

We modeled the multi-cluster environment called Distributed ASCI Supercomputer 3 DAS-3[1], located in the Netherlands. Such environment is composed of 277 processors located in 5 clusters the number of processors varies from 30 to 90 and the average processor speed is 2.4 GHz. The interconnection speed between nodes in the same cluster is an average of 1 GB/s and the interconnection speed between nodes of different clusters is an average of 100 MB/s.

[1] DAS-3 website: http://www.cs.vu.nl/das3/index.shtml

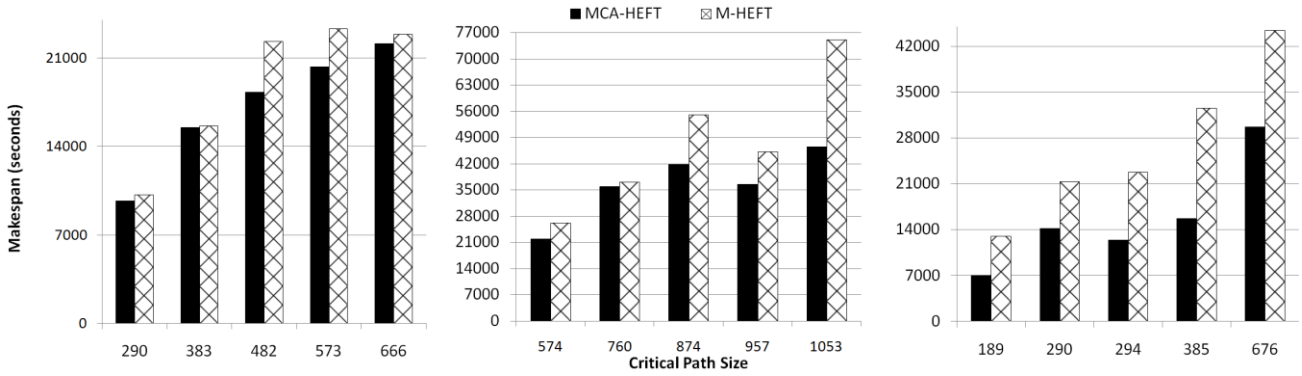

Figure 2. Makespan of the Few-tasks-by-level Set, Various-tasks-by-level Set, Many-tasks-by-level Set.

Table 1. Evaluation Workload.

Tasks	Levels	MCACCR > MCA (%)	Critical Path Size
Few-tasks-by-level			
21	3	33	290
29	4	34	383
37	5	38	482
46	6	41	573
55	7	33	666
Various-tasks-by-level			
55	6	33	574
89	8	43	760
93	9	31	874
113	10	45	957
117	11	44	1053
Many-tasks-by-level			
39	2	31	189
56	3	30	290
56	3	36	294
92	4	40	385
133	7	37	676

We modeled a set of workflows using a script that follows the approach of automatic DAG generators [2]. The workflow generation is defined by the following parameters: number of levels, minimum and maximum number of tasks per level. Based on the following parameters the generator defines values for computational cost, communication cost, and parallel fraction of each parallel task of the workflow. The workflows' characteristics that we used for the evaluation are presented in the three sets of Table 1.We evaluated the allocation process and the performance efficiency of MCA-HEFT over a variety of scenarios using such workflows.

In the three sets an average of 37% of tasks can potentially be scheduled in multiple clusters. The workflow tasks in the *Few-tasks-by-level* set have a high probability of being executed in multiple clusters, because few tasks can have large allocations. The workflow tasks in the *Various-tasks-by-level* set have lower probability of being executed in multiple clusters, because the workflows have more tasks by level than the *Few-tasks-by-level* set. The workflow tasks in the *Many-tasks-by-level* set have a very low probability of being executed in multiple clusters, because each level has a great number of tasks which will make tasks to be allocated on a few resources.

B. Result Analysis

Table 2 shows the number of allocated resources per heuristic and the percentage of tasks mapped to multiple clusters by using MCA-HEFT. We observe that, by comparing the average number of resources allocated for each workflow in all the sets, MCA-HEFT produces larger allocations than M-HEFT.

The makespan produced by the three sets are shown and compared with the critical path size. Figure 2 shows the makespan for the sets *Few-tasks-by-level*, *Various-tasks-by-level*, and *Many-tasks-by-level* respectively. The results show that MCA-HEFT has a better performance than M-HEFT. The average performance improvement is 8%, 20%, and 42% for *Few-tasks-by-level*, *Various-tasks-by-level*, and *Many-tasks-by-level*, respectively.

Table 2. Resource Distribution.

Avg. Number of Resources MCA-HEFT	Avg. Number of Resources M-HEFT	Tasks Mapped to Multiple Clusters MCA-HEFT (%)
Few-tasks-by-level		
24	13	10
26	10	10
26	9	11
25	8	13
25	7	9
Various-tasks-by-level		
22	7	7
18	5	11
21	5	5
19	5	7
20	5	9
Many-tasks-by-level		
7	8	0
9	7	2
9	7	4
8	5	1
11	4	3

In order to understand the performance improvement of the proposed heuristic, we analyze three aspects: (i) how resources are allocated; (ii) the impact of network infra-structure on the execution of workflows; and (iii) the relation between critical path size and performance improvement.

1) Resource Allocation

MCA-HEFT allocates more resources to single parallel tasks than M-HEFT because the number of available resources considered by MCA-HEFT is the sum of all resources in all the clusters. MCA-HEFT can therefore map a single task to multiple clusters to guarantee that it receives the resources specified by the first phase of the heuristic. Such allocation process, combined with the prioritization of resource allocation to the tasks in the critical path, contributed to the performance improvement compared with M-HEFT scheduling the same workflows.

The number of tasks scheduled in multiple clusters is inversely proportional to the number of tasks by level. For *Few-tasks-by-level,* an average of 11% of tasks were scheduled in multiple clusters, for *Various-tasks-by-level,* an average of 8% of tasks were scheduled in multiple clusters, and for *Many-tasks-by-level,* an average of only 2% of tasks were scheduled in multiple clusters. These values are encouraging as they show that makespans can be reduced using up to 11% of tasks on multiple clusters.

2) Network Impact

In order to evaluate the network impact we measured the time spent performing computation and communication for each workflow task. Then, we calculate the average and standard deviation of time spent performing communication operations in all workflows for the three sets, as shown in Table 3.

The average time spent performing communication is short in M-HEFT and long in MCA-HEFT. Such results were expected since a number of tasks was executed using a low speed communication infra-structure, also the standard deviation is high indicating a great difference of communication costs among tasks.

Such results show that a good estimation of tasks that need to be mapped to multiple clusters is important to improve makespans. This is mainly necessary to minimize the execution time considering the communication overhead.

3) Critical Path Size x Performance Improvement

The results show a relation between makespan and critical path size. In Figure 2 one can notice that the longer the makespan the higher the performance improvement between makespans produced by MCA-HEFT against makespans produced by M-HEFT. Such relation can be explained considering two aspects: the MCA-HEFT scheduling process is performed by levels and MCA-HEFT prioritizes the scheduling of tasks on critical path.

Table 3. Network Execution Time.

	MCA-HEFT		M-HEFT	
	AVG	SD	AVG	SD
Few-tasks-by-level	770	455	0	1
Various-tasks-by-level	14821	12401	1	4
Many-tasks-by-level	205	440	0	1

The scheduling plan of a workflow is performed by levels. In each level the makespans produced by MCA-HEFT may have a better performance or a worse performance than that produced by M-HEFT. The MCA-HEFT heuristic has a worse performance in the levels that has many tasks that has a high communication cost and low computational cost, due to the fact that MCA-HEFT does not use the network bandwidth or latency as scheduling parameters.

The scheduling of workflows with few levels containing many tasks with high communication cost and low computational cost using MCA-HEFT will have a better performance than M-HEFT. In the case of workflows with many levels, when the critical path is low the MCA-HEFT scheduling plan of each level will probably have a worse performance than M-HEFT because tasks on critical path will have a sharply improvement on execution time, thus, the MCA-HEFT performance will be similar to M-HEFT. When the critical path is high the tasks on critical path on each level have the execution time heavily improved, consequently the scheduling plan of each level has a better performance than M-HEFT, thus, MCA-HEFT performance is better than M-HEFT.

In cases where the number of tasks per level is high, MCA-HEFT allocates a minimum number of resources to almost all tasks, and only the tasks on the critical path are allocated to a high amount of resources. All these factors make the performance of MCA-HEFT better than M-HEFT as seen in Figure 2.

V. Related Work

The MCA-HEFT heuristic is a list scheduling heuristic based on HEFT [11] [6] with the objective of scheduling workflows with sequential and parallel tasks on multi-cluster environments. A number of strategies for scheduling workflows with sequential tasks in grid environment have been developed [12], some of these strategies have been extended in order to consider the task parallelism, such as, Parallel HEFT (P-HEFT) [13] and M-HEFT.

M-HEFT [3] extends HEFT to the case of data-parallel tasks and a platform that consists of heterogeneous clusters. M-HEFT considers each task as a parallel task [14]. In this sense, the weight of tasks is computed on all possible subset of resources of each cluster and the weight of data transfer between tasks is computed by the sum of the average network latency and data size divided by bandwidth of each pair of resources in each cluster.

The execution of parallel tasks on multiple clusters has been investigated by several authors [7], [15], [16], [17], [18], [19], [20]. The main reasons for executing a parallel application on multiple clusters are: (i) applications may require certain computing power that is not available in a single cluster; (ii) users may want to reduce the response time of their applications by using resources from multiple clusters; or (iii) user can have lower response time by merging fragments of multiple scheduling queues. One key issue is the possible bottleneck of the inter-cluster network overhead. Most of the

studies considering this issue showed that 25% of the total application execution is a tolerable overhead value. The main limitation of the projects that worked with resource co-allocation is that they considered parallel applications in an isolated fashion, *i.e.* having no influence on what needs to be executed before or after them, which is the case explored in this paper. The proposed heuristic can split tasks onto multiple clusters considering workflow task dependency.

VI. Conclusion and Further Work

We introduced a heuristic called MCA-HEFT for scheduling workflows composed of sequential and parallel tasks. The key novelty of MCA-HEFT is that it explores the possibility of mapping parallel tasks in multiple clusters. We performed a detailed analysis of the benefits of MCA-HEFT compared to M-HEFT, which is one of the most efficient heuristics in the literature.

A metric called MCACCR was created in order to define whether the task can have the performance improved by being executed in multiple clusters. Such metric takes into account the parallel task characteristics and the infra-structure characteristics. One interesting finding is that, based on our workloads, scheduling approximately only 10% of the tasks to multiple clusters, it is possible to reduce drastically the makespan of the workflow executions.

As future work, we will explore data transfer between tasks and dynamic task mapping inside MCA-HEFT. Moreover, we are planning on performing experiments on a multi-cluster production environment.

References

[1] I. T. Foster, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," 2001, pp. 1–4.

[2] H. Casanova, F. Desprez, and F. Suter, "On cluster resource allocation for multiple parallel task graphs," *Journal of Parallel and Distributed Computing*, vol. 70, no. 12, pp. 1193–1203, Dec. 2010.

[3] T. N'Takpe, F. Suter, and H. Casanova, "A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms," in proceedings of the Parallel and Distributed Computing, 2007. ISPDC '07. Sixth International Symposium on, 2007, p. 35.

[4] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," *Proceedings of the Job Scheduling Strategies for Parallel Processing*, pp. 1–34, 1997.

[5] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*, 2008.

[6] H. Topcuouglu, S. Hariri, and M. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[7] M. A. S. Netto and R. Buyya, "Resource Co-Allocation in Grid Computing Environments," in *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*, IGI Global, 2010, pp. 476–494.

[8] P. F. Dutot, T. N'takpé, F. Suter, and H. Casanova, "Scheduling Parallel Task Graphs on (Almost) Homogeneous Multicluster Platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 7, pp. 940–952, 2009.

[9] T. Shanley, *Infiniband*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[10] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A Generic Framework for Large-Scale Distributed Experiments," in proceedings of the Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on, 2008, pp. 126–131.

[11] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer, 2008.

[12] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*, 2008.

[13] J. G. Barbosa and B. Moreira, "Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters," *Parallel Computing*, vol. 37, no. 8, pp. 428–438, Aug. 2011.

[14] F. Suter, F. Desprez, and H. Casanova, "From Heterogeneous Task Scheduling to Heterogeneous Mixed Parallel Scheduling," in *Proceedings of the 10th International Euro-Par Conference (Euro-Par'04)*, Pisa, Italy, 2004, vol. 3149, pp. 230–237.

[15] M. A. S. Netto and R. Buyya, "Rescheduling co-allocation requests based on flexible advance reservations and processor remapping," in *2008 9th IEEE/ACM International Conference on Grid Computing*, 2008, pp. 144 –151.

[16] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit, "On Advantages of Grid Computing for Parallel Job Scheduling," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Washington, DC, USA, 2002, p. 39–.

[17] W. M. Jones, W. B. Ligon,III, L. W. Pang, and D. Stanzione, "Characterization of Bandwidth-Aware Meta-Schedulers for Co-Allocating Jobs Across Multiple Clusters," *J. Supercomput.*, vol. 34, no. 2, pp. 135–163, Nov. 2005.

[18] Q. Snell, M. J. Clement, D. B. Jackson, and C. Gregory, "The Performance Impact of Advance Reservation Meta-scheduling," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, London, UK, UK, 2000, pp. 137–153.

[19] H. H. Mohamed and D. H. J. Epema, "Experiences with the KOALA co-allocating scheduler in multiclusters," in *IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005*, 2005, vol. 2, pp. 784 – 791 Vol. 2.

[20] A. I. D. Bucur and D. H. J. Epema, "Scheduling Policies for Processor Coallocation in Multicluster Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 958–972, 2007.