Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 301–311. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

SENSITIVITY ANALYSIS OF WORKFLOW SCHEDULING ON GRID SYSTEMS[*]

MARíA M. LÓPEZ, ELISA HEYMANN, MIQUEL A. SENAR[†]

Abstract. Scheduling is an important factor for the efficient execution of computational workflows on Grid environments. A large number of static scheduling heuristics has been presented in the literature. These algorithms allocate tasks before job execution starts and assume a precise knowledge of timing information, which may be difficult to obtain in general. To overcome this limitation of static strategies, dynamic scheduling strategies may be needed for a changing environment such as the Grid. While they incur run-time overheads, they may better adapt to timing changes during job execution. In this work, we analyse five well-known heuristics (min-min, max-min, sufferage, HEFT and random) when used as static and dynamic scheduling strategies in a grid environment in which computing resources exhibit congruent performance differences. The analysis shows that non-list based heuristics are more sensitive than list-based heuristics to inaccuracies in timing information. Static list-based heuristics perform well in the presence of low or moderate inaccuracies. Dynamic versions of these heuristics may be needed only in environments where high inaccuracies are observed. Our analysis also shows that list-based heuristics significantly outperform non-list based heuristics in all cases and, therefore, constitute the most suitable strategies by which to schedule workflows either statically or dynamically.

Key words. workflows, scheduling, grid

1. Introduction.

Grid environments connect distributed and heterogeneous resources in a seamless system that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations [1]. Grid systems offer high computing capabilities that are used in many scientific research fields. Biologists, chemists, physicists and other researchers have therefore become intensive users of applications with high performance computing characteristics. Several projects such as GriPhyn [2], DataGrid [3], GridLab [4], GrADS [5] or CrossGrid [6], whose middleware infrastructure is intended to simplify application deployment on computational grids, have also developed special middleware for defining and managing scientific workflows.

Scientific workflows are concerned with the automation of scientific processes that consist of inter-dependent jobs, where tasks are structured on the basis of their control and data dependencies. Workflow applications are an important class of programs that can take advantage of Grid-system power, as has been shown in the LIGO [7] pulsar search, several image processing applications [8] or the ATLAS physics experiment [9].

Any workflow management system requires certain key elements:

1. Definition and composition of workflow components.
2. Task mapping and scheduling during execution.
3. Data movement between dependent tasks.
4. Handling of execution failures.

From the above list of issues, we focus on the second, which is an ongoing research effort covered by much recent work. In general, workflow scheduling strategies cover different trade-offs between dynamicity and look-ahead. On the one hand, decisions about the resource on which to run a particular task can be made as soon as possible (early) or just before the task is executed (in-time). On the other hand, decisions can be made with reference to information about the whole workflow or simply the task at hand. Obviously, the complexity and run-time overhead incurred in the decision-making process increases as global information and in-time strategies are used. The design of a scheduling strategy also depends on what objective function the user wants to minimize or maximize (e.g. minimizing overall job completion time or maximizing resource utilization). Our objective function is to minimize overall job completion time or the application makespan. The makespan of a workflow application is defined as the time at which the last component of the workflow finishes execution.

Traditionally, scheduling strategies assume the existence of deterministic information about processing times for all workflow tasks and for their corresponding communications. These times may be provided by the user or obtained by profiling techniques, and are generated before the execution. Current information on the execution environment can also be obtained by on-line tools, such as NWS [10] and is needed to make scheduling decisions at run-time. Unfortunately, accurate models of processing and communication times are very hard

[*]This work was made in the frame of the "int-eu.grid" project (sponsored by the European Union), and supported by the MEyC-Spain under contract TIN 2004-03388, and partially supported by the NATO under contract EST.EAP.CLG 981032

[†]Departament d' Arquitectura d' Ordinadors i Sistemes Operatius Universitat Autònoma de Barcelona, Barcelona, Spain mmar@aomail.uab.es, {elisa.heymann, miquelangel.senar}@uab.es

to obtain in practice and, therefore, schedules based on computation and communication estimates may suffer from significant timing changes at run-time.

Many static heuristics have been proposed in the literature. Heuristics based on list scheduling are among those that provide good quality schedules at a reasonable cost. These strategies [11, 12, 13] schedule the tasks in the order of their previously computed priority. Thus, at each scheduling step, a task is first selected and its processor is then calculated. HEFT [11] is considered to be one of the best strategies within the group of list scheduling heuristics. However, in [13] it was observed that its performance is affected by the approach followed to assign weights to the nodes and edges of the graph. A recent study [14] has also analyzed the performance of a low-cost rescheduling policy, which attempts to reschedule tasks selectively during workflow execution (hence, without incurring a high overhead).

Other studies have analyzed another set of heuristics that do not take into account the whole structure of workflow dependencies during the scheduling process. The graph is analyzed according to the dependencies but each task does not have a predefined priority. So, at each scheduling step, each ready task is tentatively assigned to every processor and the best (task, processor) pair is selected. Thus, at each step, both a task and its corresponding processor are selected at the same time. Different strategies use different criteria to select the best pair (for instance, the pair that guarantees the minimal completion time for all ready tasks, the pair that provides a minimum starting time for all ready tasks, etc). Among others, strategies such as min-min [15], max-min [16] and sufferage [17] have been used and analyzed in recent work [18, 19].

In general, list-scheduling strategies exhibit a higher computational complexity than strategies from the other group. They perform well for irregular and communication-intensive graphs, where the scheduling of crucial tasks first seems to be very important. Strategies that are not based on list-scheduling perform well on computation-intensive graphs with a high degree of parallelism, in which a maximum utilization of processors seems to be one of the most important factors to be considered.

However, to the best of our knowledge, no comparative studies have been carried out between list-scheduling strategies and non-list scheduling strategies. In our work, we have compared strategies from both groups and we study their behavior both when they are applied statically and dynamically.

The remainder of the paper is organized as follows. Section 2 provides some background and describes the assumptions that we consider within our problem setting. Section 3 illustrates the problem of timing inaccuracies on static scheduling heuristics. In section 4, we analyze the sensitivity of several heuristics to timing changes during graph execution and also study the benefits of dynamic rescheduling applied to all the strategies. Finally, section 5 concludes the paper.

2. Background. Many complex applications consist of inter-dependent jobs that cooperate in order to solve a particular problem. The completion of a particular job is the condition needed to start the execution of those jobs that depend upon it. This kind of application workflow may be represented in the form of a DAG a directed acyclic graph. A DAG is a graph with one-way edges that does not contains cycles. DAGs are frequently used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. The programs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies of these programs. Fig. 2.1 represents an example DAG corresponding to the MB application [20].

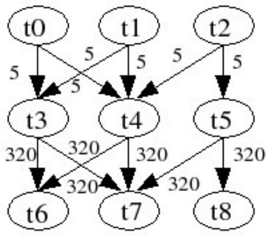


Fig. 2.1. *DAG corresponding to the MB application.*

Associated to each node n and machine m, there is the real execution time, which represents the exact amount of time that takes to execute node n on machine m. In practice, in a real grid environment, real execution times (corresponding to both task computation and intertask communications) are not available;

however we use them for comparative purposes. Fig. 2.2(b) shows the real execution times for the MB nodes when executed on 3 different machines (M1, M2 and M3).

Machines	Transfer Rate real (Mbits per sec)	Transfer Rate estimated (Mbits per sec)
M1-M2	11	15
M1-M3	11	13
M2-M3	8	5

(a)

Real Execution Time

	M1	M2	M3
t0	227	232	309
t1	168	172	230
t2	150	153	205
t3	299	306	408
t4	311	318	424
t5	299	306	408
t6	281	288	384
t7	362	370	494
t8	193	197	264

(b)

Estimated Exec. Time

M1	M2	M3
265	269	339
197	168	202
174	136	173
338	272	433
275	343	437
341	278	439
255	246	335
361	333	476
184	222	294

(c)

FIG. 2.2. *(a) Communication time between machines (real and estimated) . (b) Real execution times in seconds. (c) Estimated execution times (variation=50%).*

As real execution times are in practice not available at submission time, in our work we assume that estimates of the execution times of the nodes are known in advance. We calculate these estimated by varying the real value randomly by a certain percentage. Fig. 2.2 (c) shows an example of varying the execution times of Fig. 2.2 (b) up to ±50%.

In addition to the nodes computation time, there is a communication volume associated to each pair of communicating nodes. These times in Mbytes are shown beside each edge on Fig. 2.1.

Finally there is a communication transfer rate associated to each pair of machines, which are shown in Fig. 2.2(a). In this example, differences between real and estimated transfer rates have been computed by applying a 50% variation to real transfer rates.

Grid environments are composed of a large number of heterogeneous machines. Commonly, users have an estimated knowledge of their DAG timing when it is executed on their local machines, but we have no guarantees about the accuracy of this estimated execution time (as the DAGs jobs may be executed with different input data, on different machines and so on). In the following section, we illustrate the problems derived from this fact when scheduling workflows.

3. Scheduling Sensitivity to Timing Inaccuracies. We now show an example of the makespan obtained when scheduling the MB DAG described in the previous section.

Usually only estimated information about both the jobs' execution and communication time is available. By scheduling this DAG using a particular scheduling strategy (HEFT) applied to the estimated execution time of the jobs on 3 machines, the Gantt chart in Fig. 3.1 is obtained. Slashed boxes represent communication before the execution of a node. For the sake of simplicity in this example we asume that communication times are known. It is worth mentioning that in the rest of the paper we consider that exact communication times are not known, as it happens in reality. In the figure we see that tasks 0, 4 and 6 were assigned to machine M1, tasks 1, 5 and 8 to machine M2, and tasks 2, 3 and 7 to machine M3.

But after executing tasks on machines we get the real execution time, which is frequently different to the estimated one. Fig. 2.2 (b) shows a variation of the real execution time up to a 50% of the estimated time. If we consider these inaccurate estimated execution times and apply a machine normalizing factor for the machines M1, M2 and M3, the Gantt chart in Fig. 3.2 is obtained. In this case the tasks are executed on the same machines as in the previos Gantt chart, but we measure the obtained real makespan.

Having accurate information about the nodes execution time is not quite realistic when executing real applications on real environments, but it is useful to see how makespan is worsened when this information is inaccurate. Fig. 3.3 shows the Gantt chart obtained when real execution times are known. We see that the makespan experienced a worsening of 10.3% when the estimated times were considered (Fig. 3.2)

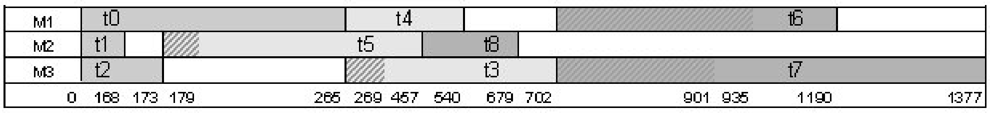


Fig. 3.1. *Gantt chart obtained using the HEFT policy when only estimated execution times are known.*

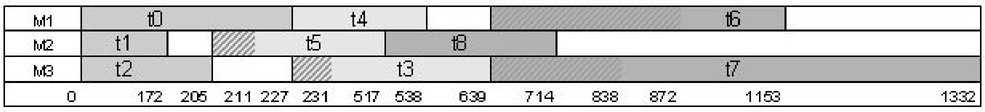


Fig. 3.2. *Gantt chart obtained using the HEFT strategy when estimated execution times are known and real execution times are obtained.*

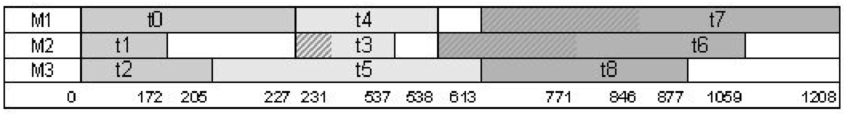


Fig. 3.3. *Gantt chart obtained using the HEFT strategy when real execution times are known.*

As the scheduling strategies for workflows (described in the following section) are sensitive to the accuracy of the estimated execution times, we studied how these strategies behave when performing Rescheduling. Rescheduling consists of updating the value of the estimated execution time for all the pending nodes of a specific machine (the one on which the task that has just finished was executed), according to the execution time value obtained by the recently completed task. Fig. 3.4 depicts the Gantt chart obtained when applying rescheduling to our example DAG. In this case, tasks 0, 5 and 7 were assigned to machine M1, tasks 1, 4, 6 to machine M2, and tasks 2, 3 and 8 to machine M3. It can be seen that the makespan worsening is reduced to 1.16% with respect to the ideal situation in which real execution times of the tasks are known in advance.

In general, rescheduling will not provide the makespan obtained with the real (and therefore, accurate) execution time values, but it provides an improvement on the makespan obtained when only estimated execution times are known. We performed a systematic and exhaustive study with the aim of evaluating the behavior and benefits obtained when rescheduling. This study is presented in the following section.

4. Experimental Study. In this section, we evaluate the performance of several scheduling strategies with respect to the makespan obtained when these are applied to scheduling both fork-join and random workflows on heterogeneous systems. We test various scheduling strategies under different scenarios:

1. Knowing in advance the real execution time of the applications nodes. Clearly this case does not correspond to a real situation, but it is used for comparative purposes.

2. Knowing only an estimate of the nodes execution time and the intertask communication times. This scenario pertains when there is an estimate of task execution times and communication rates, possible obtained from previous executions, which may (or may not) be accurate.

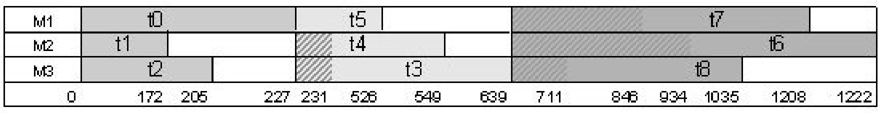


FIG. 3.4. *Gantt chart obtained using the HEFT policy when rescheduling.*

3. Knowing only an estimate of the nodes execution time and the intertask communication times, but being able to dynamically *reschedule* the remaining tasks after each task finishes. In this case, accurate information obtained for a just-executed task is used to update the estimated execution times of the remaining tasks.

As the main result from these simulation experiments, we are interested in obtaining information relating to how rescheduling strategies perform on average, and which scheduling policy is most suitable for performing such rescheduling.

4.1. Heuristics Description. The set of scheduling strategies studied were the following:

1. **Min-min:** For each task Ti, the machine Mi having the minimum estimated completion time for Ti is found, and the tuple (Ti, Mi, ET) is formed, ET being the execution time of Ti on Mi. Of all the tuples, that with minimum ET is scheduled. These steps are repeated iteratively until all the tasks have been scheduled.
2. **Max-min:** For each task Ti, the machine Mi having the minimum estimated completion time for Ti is found, and the tuple (Ti, Mi, ET) is formed, ET being the execution time of Ti on Mi. Of all the tuples, that with maximum ET is scheduled. These steps are repeated iteratively until all the tasks have been scheduled.
3. **Random:** Tasks ready to be executed are assigned randomly to the machines. This strategy represents the case of a pure dynamic method that has no information on the application. In principle, this should obtain the worst performance of all the strategies presented; therefore it will be used as a lower bound in the performance achievable by the other strategies.
4. **Sufferage:** In this strategy, both the minimum and second best minimum ET values are found for each task. The difference between these two values is the sufferage value. The task having the maximum sufferage value is scheduled next. These steps are repeated iteratively until all the tasks have been scheduled.
5. **HEFT:** This strategy selects the task with the highest upward rank value at each step, and assigns the selected task to the processor which minimizes its earliest finish time.

4.2. Simulation Framework. All described scheduling strategies have been systematically simulated for the three possible scenarios, in order to obtain the makespan, considering the following factors:

1. *Number of Machines:* This represents the number of processors on which each DAG is executed. The features of the machines used in the simulation were taken from executing real applications on real machines. We performed simulations using 5, 10, 15 and 20 machines. Machines are heterogeneous, and communications among them were characterized by the bandwidth values obtained experimentally.
2. *DAGs:* This represents the DAGs considered. We generated 100 completely random DAGs, and also 100 fork-join random graphs. The number of nodes ranged from 50 to 250. Thus we examine systems with a medium or large amount of tasks. The generated fork-join graphs contained a number of levels between 5 and 20, and the number of nodes per level ranged between 5 and 25. Task computing-time values ranged between 100 and 500 units, while communication times among tasks were also randomly generated and ranged between 1 and 20 Mbytes per second.
3. *Variation:* This represents the percentage of error that the estimated values may have with respect to real execution times and real communication times. We considered values of 10%, 30%, 50%, 70% and 90%. When a 10% variation was used, task execution times and communication times were fairly reliably estimated. When a 90% variation was used, tasks exhibited significant changes in their execution, corresponding to applications with highly erroneous estimations.

It should be observed that, in our simulations when two adjacent nodes were executed on the same machine, the associated communication cost is 0, while if these two nodes are executed on different machines, the communication cost is weighted up to the bandwidth of the link communicating those machines.

Time variations have been applied in a congruent way. By congruent we mean that in our simulations we assume a set of processors P1, P2, , Pi, with a fixed performance ratio between each pair. As a consequence, when task execution times and their variations were generated, execution times in the fast processor P1 were always smaller than in the slow processor P2, and their ratio was always the performance ratio between P1 and P2; i. e., if the execution time in processor P1 for tasks T1 and T2 is 10 and 20, respectively, and if processor P2 is twice as fast as P1, then the execution times for tasks T1 and T2 in processor P2 will be 5 and 10, respectively.

Given a number of machines, a DAG and a variation, our simulator computes the makespan obtained for the five strategies studied for the three scenarios previously described (considering exact task execution times, estimated execution times and rescheduling). The particular features of the simulated grid nodes were taken from a real grid environment using the Globus Information Index [21] for machine information, and the Network Weather Service tool [10] for information about the network bandwidth.

Each DAG was simulated using 10 different sets of grid machines (out of 20). In the next subsection we show the average of the makespan obtained considering the 100 DAGs for each scenario.

4.3. Simulation Results. Although we have conducted tests for all the values commented on, in this section we present only those results that are the most interesting. We will illustrate—with figures—the results for 5 and 20 machines since these prove to be sufficiently representative for the results obtained with an intermediate number of machines.

In the rest of the section, certain pertinent result figures for makespan time are presented. The Y-axis contains average makespan. We now review the most relevant results obtained from our simulations.

Fig. 4.1 and Fig. 4.2 show the negative effect of varying the estimated execution time for all the scheduling strategies in the case of 5 (Fig. 4.1) and 20 machines (Fig. 4.2) for an average of 100 random generated graphs. The average makespan is obtained for the 5 strategies (min-min, max-min, sufferage, heft and random) under the cases of real and estimated information on task execution times. The X-axis contains the variation, ranging from low-variation applications (10%) to highly inaccurate estimations (90%).

Simmilarly, Fig. 4.3 and Fig. 4.4 show the results obtained by the same heuristics when real and estimated execution and communication times. In this case, results are slightly worse than in the previous one because errors in communication times are added to errors on execution times.

Fig. 4.5 and Fig. 4.6 shows a reduction of the negative effect of the makespan when rescheduling, considering different variations, for all the scheduling strategies in case of 5 (Fig.4.5) and 20 machines (Fig. 4.6), for an average of 100 random generated graphs. The average makespan is obtained for the 5 strategies (min-min, max-min, sufferage, heft and random) for estimated (i. e., inaccurate) cases and when performing rescheduling. The X-axis contains the variation, ranging from low-variation applications (10%) to highly inaccurate estimations (90%).

Fig. 4.7 and Fig. 4.8 show simulation results obtained when rescheduling was applied to the cases that exhibited variations in execution and communication times.

It is observed that, regardless of the number of machines, if the real times for tasks and communications are not known (the case in which estimated times are used), higher makespans are obtained. As expected, the higher the variation, the higher the execution time worsening for the whole DAG. In particular, when the variation applied to the estimated execution times of the graph nodes is 10%, makespan worsening is close to 5%; in contrast, when the variation is 70%, this worsening increases up to 10%. In general, non-list scheduling strategies are more sensitive than list-based scheduling strategies to inaccuracies in timing information.

In a grid environment, where machines are heterogeneous and the conditions of the system change dynamically, the real execution times of the tasks are usually not known. Therefore, scheduling a DAG through a simple non-list scheduling policy such as min-min or max-min using estimated execution times for the tasks would lead to a poor makespan. This situation is alleviated if we take advantage of the information on the machines current situation, given by recording the execution time for those tasks that have just finished their execution, and by performing rescheduling. In other words, we only have an estimation of the nodes execution time, but are able to dynamically reschedule the remaining tasks after each task finishes.

Static list-based heuristics perform well in the presence of low or moderate inaccuracies in timing information. Nevertheless, dynamic versions of these heuristics are needed when high inaccuracies are detected.

Now that we have determined that rescheduling reduces makespan, especially in cases of high variations for DAG task execution time, the consequent question to address is which scheduling strategy introduces greater benefits when rescheduling. Fig. 4.9 and Fig. 4.10 depicts the average makespan for all the strategies using rescheduling for a 70% variation. The X-axis contains the number of machines, and the Y-axis the average makespan.

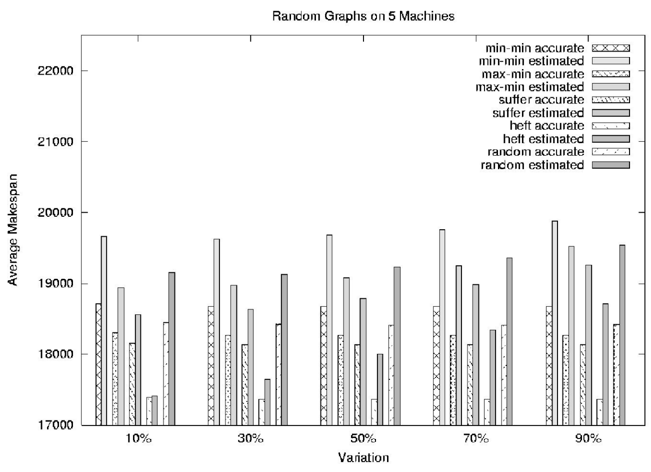


Fig. 4.1. *Effect of the variation of execution time considering 5 machines.*

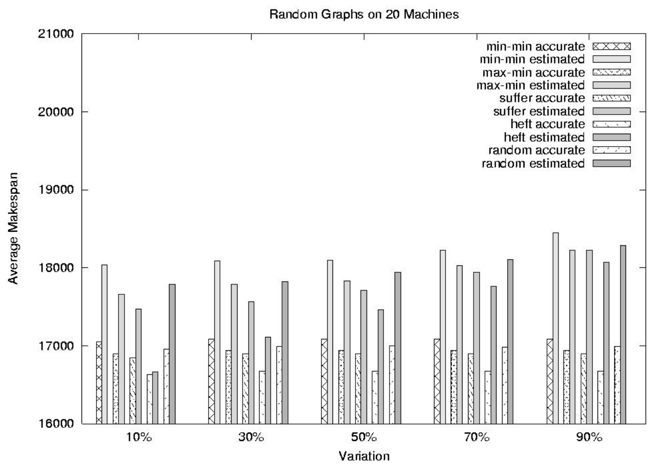


Fig. 4.2. *Effect of the variation of execution time considering 20 machines.*

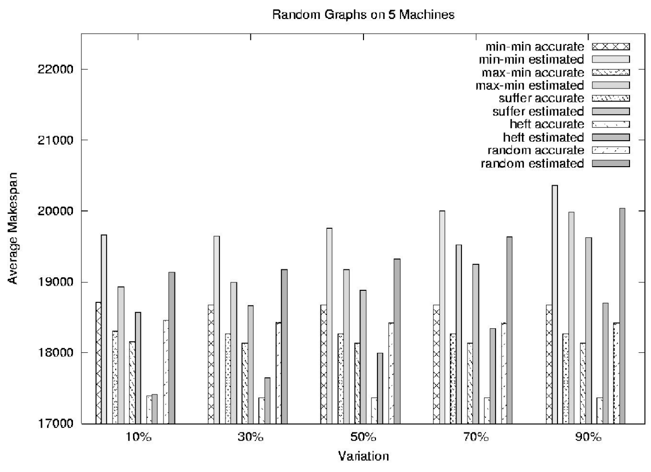


Fig. 4.3. *Effect of the variation of execution and communication times considering 5 machines.*

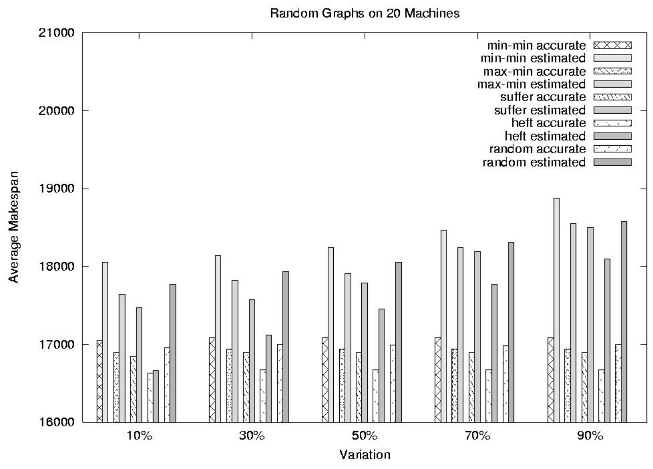


Fig. 4.4. *Effect of the variation of execution and communication times considering 20 machines.*

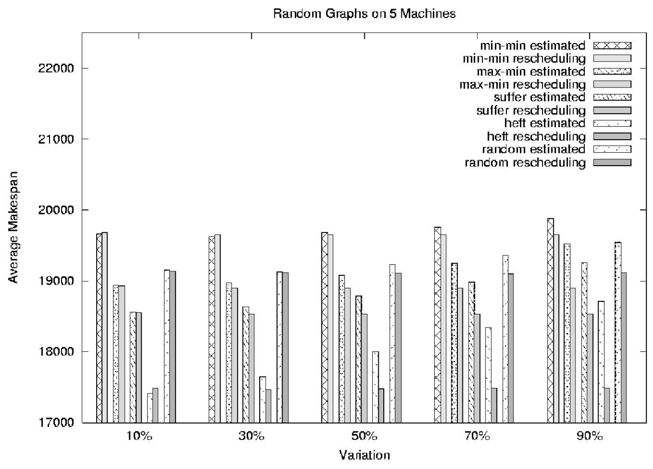


Fig. 4.5. *Rescheduling reduces makespan worsening. Case: 5 machines with execution time variations only.*

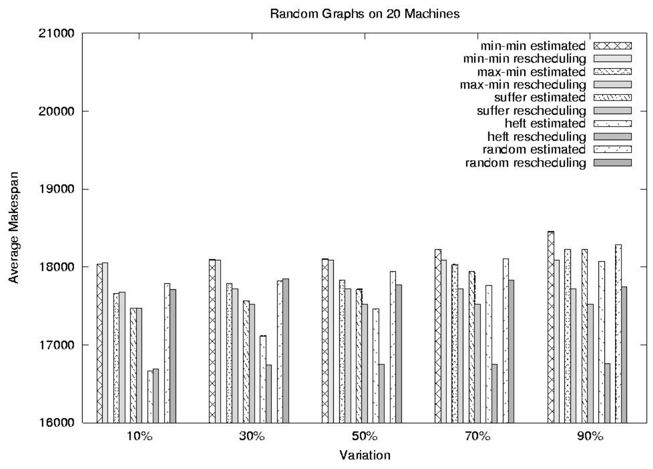


Fig. 4.6. *Rescheduling reduces makespan worsening. Case: 20 machines with execution time variations only.*

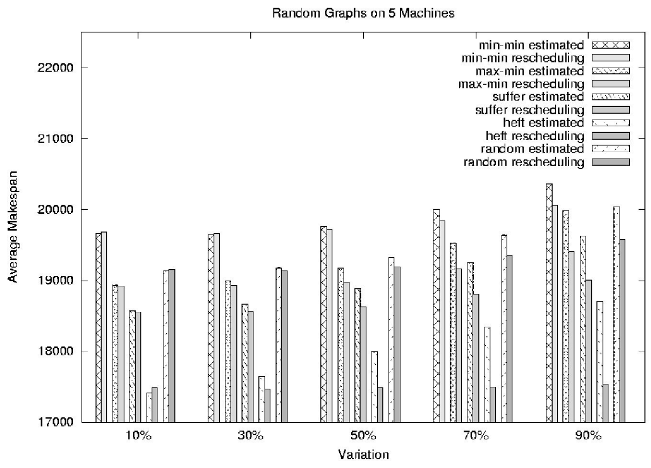


FIG. 4.7. *Rescheduling reduces makespan worsening. Case: 5 machines with execution and communication time variations.*

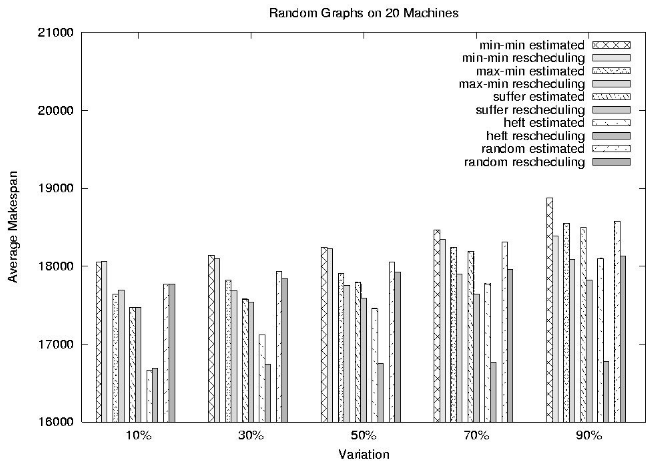


FIG. 4.8. *Rescheduling reduces makespan worsening. Case: 20 machines with execution and communication time variations.*

As it is shown in Fig. 4.9 and Fig. 4.10, the best results when rescheduling are achieved for the HEFT strategy. This is not surprising, since this is a list-based scheduling strategy that considers the DAG as a whole, while the remaining policies (i. e., non-list scheduling strategies) only consider those nodes ready to be executed. What is unexpected is the fact that min-min performs worse than the random policy. The performance of the remaining non-list scheduling policies (max-min and sufferage) is not entirely different from the performance achieved by the random policy.

4.4. Discussion. We now summarize the main results that have been derived from all the simulations performed.

Fig. 4.11 shows the average worsening percentage obtained for all the studied strategies, considering variations on the estimated execution time of 10% 30% 50% 70% and 90% when using 20 machines. For each strategy, the first number represents the worsening percentage when scheduling using estimated times, while the second number shows how this percentage is reduced when rescheduling.

As a consequence of the simulations carried out, we conclude that rescheduling performs well in terms of makespan in most cases, especially when task execution time is uncertain (variation of 70% and 90%). For example, the HEFT policy shows a worsening percentage of 8.4% when variation is 70%, and this effect is reduced to 1.2% when rescheduling. When the variation is low (10% and 30%), estimated execution time values are not far from real values, therefore the different policies do not frequently make erroneous decisions. In general, the higher the inaccuracy of estimated execution times, the greater the benefits of rescheduling.

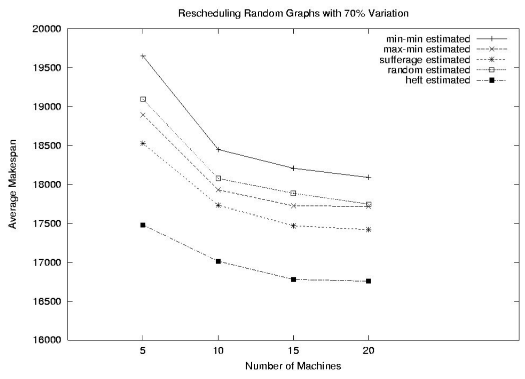


FIG. 4.9. *Overall strategy behavior when rescheduling in the presence of execution time variations only.*

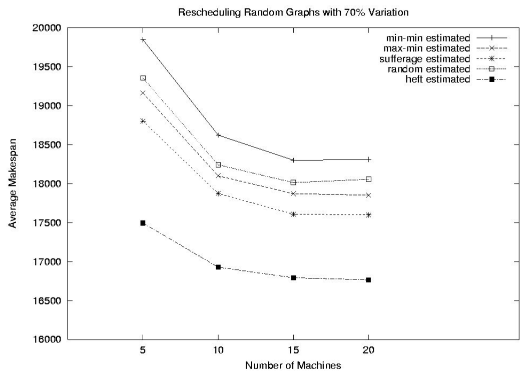
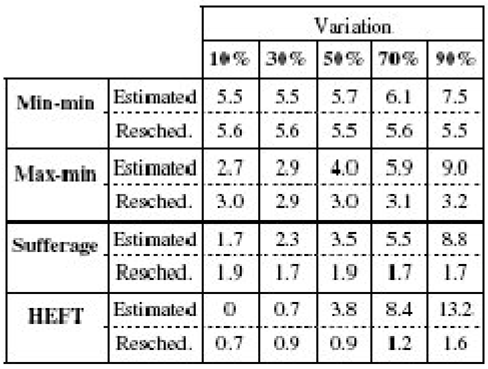


FIG. 4.10. *Overall strategy behavior when rescheduling in the presence of execution and communication time variations.*

In general, HEFT (list-based heuristic) significantly outperforms non-list based heuristics in both static and dynamic situations, under almost all the variations considered. Therefore HEFT constitutes a very suitable strategy for workflow scheduling on the grid.



		Variation				
		10%	30%	50%	70%	90%
Min-min	Estimated	5.5	5.5	5.7	6.1	7.5
	Resched.	5.6	5.6	5.5	5.6	5.5
Max-min	Estimated	2.7	2.9	4.0	5.9	9.0
	Resched.	3.0	2.9	3.0	3.1	3.2
Sufferage	Estimated	1.7	2.3	3.5	5.5	8.8
	Resched.	1.9	1.7	1.9	1.7	1.7
HEFT	Estimated	0	0.7	3.8	8.4	13.2
	Resched.	0.7	0.9	0.9	1.2	1.6

FIG. 4.11. *Worsening percentages.*

5. Conclusions. In this paper, we have analyzed the sensitivity to inaccurate execution and communication times of several heuristics used to schedule computational workflows on the grid. We compared the performance of a list-scheduling strategy (HEFT) and four non-list scheduling strategies (min-min, max-min, sufferage and random) both when applied as a pure static strategy or as a pure dynamic strategy computing a new schedule each time a new task is completed. Our results show that, in a system with heterogeneous processors exhibiting congruent behavior, HEFT outperforms the other strategies in all cases, even where task computation and task communication times show a high variation at run-time. Interestingly, static schedules obtained by HEFT perform well for low-variation degrees (less than 50%). HEFT significantly benefits from a dynamic scheduling when time variations are greater than 50%. In practice, HEFT constitutes the most attractive strategy because the effectiveness of its dynamic version could be combined with a selective rescheduling policy, as presented in [14], in order to avoid high run-time overheads.

REFERENCES

[1] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco: Morgan Kauffman, 1999.

[2] *GriPhyN: The Grid Physics Network.*, `http://www.griphyn.org`

[3] *European DataGrid Project*, `http://www.eu-datagrid.org`

[4] *GridLab: A Grid Application Toolkit and Testbed*, `http://www.gridlab.org/`

[5] F. Berman et al, *The GrADS Project: Software Support for High-Level Grid Application Development*,International Journal of High Performance Computing Applications, Vol. 15, No. 4, 327–344 2001.

[6] *The CrossGrid Project*, `http://ww.crossgrid.org`

[7] *Ligo Scientific Collaboration*, `http://www.ligo.org/`

[8] S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz, *Image Processing on the Grid: A Toolkit or Building Grid-enabled Image Processing Applications*, 3rd International Symposium on Cluster Computing and the Grid, 2003.

[9] *Atlas Collaboration*, `www.cern.ch/Atlas/`

[10] R. Wolski, N. Spring, J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, Journal of Future Generation Computing Systems, Vol. 15, Num. 5–6, pp. 757–768, October, 1999.

[11] H. Topcuoglu, S. Hariri, and M. Y. Wu, *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,*, IEEE Trans. Parallel and Distributed Systems, Vol. 13, no. 3, pp. 260–274, 2002.

[12] A. Radulescu and A. J. C. van Gemund, *On the complexity of list scheduling algorithms for distributed memory systems*, Proc. ACM International Conference on Supercomputing, 1999.

[13] R. Sakellariou and H. Zhao, *A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems*, Proc. Of the 13th Heterogeneous Computing Workshop (HCW04), 2004.

[14] R. Sakellariou, H. Zhao, *A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems*, Scientific Programming, 12 (4), pp. 253–262, Dec. 2004.

[15] Min-You Wu, Wei Shu, Hong Zhang, *Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems.*, 9th Heterogeneous Computing Workshop,2000.

[16] Y.-K. Kwok, I. Ahmad, *Static scheduling algorithms for allocating directed task graphs to multiprocessors*, CM Computing Surveys, 31(4), pp. 406–471, 1999.

[17] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, *Dynamic mapping of a class of independent tasks onto heterogeneous computing systems*, Journal of Parallel and Distributed Computing, 59(2), pp. 107–131, 1999.

[18] C. Raghavendra, A. Alhusaini, V. Prasanna, *A Framework for Mapping with Resource Co-Allocation in Heterogeneous Computing Systems*, 9th Heterogeneous Computing Wksp.: HCS 2000 (Cancun, Mexico, 1 May 2000), pp. 273–286.

[19] A. Mandal et al, *Scheduling Strategies for Mapping Application Workflows onto the Grid*, In 14th IEEE Symposium on High Performance Distributed Computing (HPDC 2005). IEEE Computer Society Press.

[20] R. Van der Wijngaart, M. Frumkin, *NAS Grid Benchmarks Version 1.0* , NASA Technical Report NAS-02-005 July 2002.

[21] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, *Grid Information Services for Distributed Resource Sharing,* Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, pp. 181, August 2001.

Edited by: Dana Petcu
Received: June 30, 2007
Accepted: July 26, 2007