ISSN:0975-9646

Information -Aware Scheduling Strategies for Desktop Grid Environment

Er. Jayoti Arora[1], Ms. Geeta Arora [2] , Dr. Paramjit[3] , Dr. Shaveta[4]

[1]*Computer Sc. & Engg, B.F.C.E.T, Bathinda, India*
[2]*Computer Application, S.A.S.I.I.T.R . Mohali, India*
[3,4]*Computer Sc. & Engg , G.Z.S.C.E.T,Bathinda, India*

***Abstract*— In this paper we will show how it is possible to build Information-aware schedulers able to outperform The Work Queue with Replication - Fault Tolerant Scheduler(WQR-FT). We propose different scheduling policies considering information about resources and applications. We will discuss two task selection policies and four machine selection policies that when combined give rise to 8 different scheduling algorithms. As a matter of fact, the results obtained shows that it is possible to achieve better performance than WQR-FT and reduce the wasted CPU cycles.**

***Keywords*— WQR-FT, GreatET, SmallET, BoT, NoInfo, CpuInfo, AvailInfo, AllInfo**

I. INTRODUCTION

The exploding popularity of the Internet has created a new much large scale opportunity for Grid computing. As a matter of fact, millions of desktop PCs, whose idle cycles can be exploited to run Grid applications, are connected to wide-area networks both in the enterprise and in the home. These new platforms for high throughput applications are called Desktop Grids [1, 2]. The inherent wide distribution, heterogeneity, and dynamism of Desktop Grids make them better suited to the execution of loosely-coupled parallel applications rather than tightly coupled ones. Bag-of-Tasks applications (BoT) [3, 4] (parallel applications whose tasks are completely independent from one another) have been shown [5] to be particularly able to exploit the computing power provided by Desktop Grids .

In order to take advantage of Desktop Grid environments, suitable scheduling strategies, tailored to BoT applications, must be adopted. More specifically, these strategies must be able to deal with the heterogeneity of resources, the fluctuations in the performance they deliver because of the simultaneous execution of competing applications, and their failures due to crashes/reboots or unplanned departures from the Grid. In response to this need, various scheduling algorithms have been proposed in the literature [6, 7, 5, 8, 9, 10], that typically attempt to minimize the makespan of BoT applications (that is, the time taken to execute all the tasks in a bag) in spite of resource heterogeneity, performance fluctuation, and failures.

These algorithms employ Information-free schedulers that do not base their decisions on information concerning the status of resources or the characteristics of applications, Achieving good performance in these situations usually requires the availability of good information about both the resources and the tasks, so that a proper scheduling plan can be devised. Therefore, we focus on choosing which task to execute next (task selection), and the machine on which it will be executed (machine selection).So we have proposed two task selection policies and four machine selection policies that when combined give rise to 8 different scheduling algorithms. As a matter of fact, the results obtained shows that it is possible to achieve better performance than WQR-FT and reduce the wasted CPU cycles.

The rest of the paper is organized as follows. In Section II, we discuss the information -aware scheduling strategies proposed. In particular, we describe the information-aware task selection in Section III and the information-aware machine selection in Section IV. Combining the information-aware task selection with the information-aware machine selection, we obtain various scheduling algorithms described in Section V. In Section VI, we present the results and, finally conclude the paper.

II. DEVISING INFORMATION -AWARE SCHEDULING STRATEGIES

When developing a information-aware scheduler, the obvious starting point is the identification of the types of information that is reasonable to assume to be available to the scheduler. Roughly speaking, there are two types of information that can be exploited by a scheduler, namely the information about the characteristics of the tasks, and the information about the characteristics of machines.

In this paper we will show how it is possible to build Information-aware schedulers able to outperform WQR-FT. We will start by considering task selection first, and then we will move to machine selection. In particular, we will discuss two task selection policies (called GreatET and SmallET) and four machine selection policies (called NoInfo, CpuInfo, AvailInfo and AllInfo) that, when combined, give rise to 8 different scheduling algorithms.

III. INFORMATION -AWARE TASK SELECTION

In this section, we describe two possible approaches to improve task selection with respect to WQR-FT that, as already recalled, chooses at random the next task to be executed among those tasks having the smallest number of running instances (candidate tasks). Rather than choosing this task at random, we propose to exploit the information about the residual execution time of candidate tasks. In particular, we propose two task selection policies, that we call smallest residual execution time and greatest residual execution time (or SmallET and GreatET, respectively).

In the SmallET, the scheduler selects the task with the smallest residual execution time, while, in GreatET, the scheduler selects the task with the greatest residual execution time. The residual execution time depends on when the last checkpoint has been saved for that particular task. In particular, if no checkpoint was saved the residual

execution time corresponds to the execution time, otherwise the residual execution time is the time needed to complete the task starting from the last checkpoint saved. Intuitively, the SmallET attempts to complete as soon as possible the smallest tasks, in order to have enough free hosts to submit replicas for the greatest tasks.

IV. INFORMATION-AWARE MACHINE SELECTION

As mentioned before, the second scheduling step consists in machine selection. Of course, the better the resource information that is available, the better the quality of the decisions that can be made by the scheduler. In this section, we discuss various scheduling policies that rely on increasing amounts of information and, as such, are able to obtain better performance (as shown later) at the expenses of higher costs of obtaining the information they need.

A. Machine selection without information

The first resource policy is very simple: it just selects the first available machine. This policy, called NoInfo, is illustrates in detail on Figure 1: it needs the set of resources contained in the Computational Grid (variable R, line 3) and a function called Avail(Ri) that returns true if the resource Ri is idle. For each resource (cycle for, line 6), NoInfo returns the first resource available (line 8), -1 otherwise (that is there is no available machine, line 11).

```
1: ---data structures and functions -----
2: M {is the number of resources}
3: R {is the set of resources (Ri is the ith resource)}
4: Avail(Ri){returns true if resource Ri is idle}
5: ------- NoInfo algorithm -----
6: for i = 0 to M do
7: if (Avail(Ri)) then
8: return i;
9: end if
10: end for
11: return -1;
```

Figure 1. NoInfo

B. Information-aware machine selection based on computational power

In this scenario, the scheduler is able to predict the computational power of the resources. This means that the scheduler can estimate the execution time of a specific task assigned to a specific resource. We propose a machine selection policy called CpuInfo that selects the available resource with the highest computational power. The rationale is that when there are fewer tasks to execute than ready hosts, this policy is a simple way of avoiding picking the "slow" hosts. Figure 2 illustrates in detail the CpuInfo policy, that uses two functions: CpuPwr(r) (line 4) is a function that returns the computational power of resource r, while Avail(Ri) (line 5) is a function that returns true if resource Ri is ready to receive task to execute. Initially, the algorithm sets the first resource as the selected resource (slctRsc parameter on line 8) and its computational power is set as the temporary highest computation power (maxCpuPwr on line 7). In the for cycle (line 9), the algorithm updates the selected resource if another resource is available and it has a higher computation power than maxCpuPwr (lines 10, 11 and 12). At the end of the cycle, the variable slctRsc contains the selected resource.

```
1: ---data structures and functions -----
2: M {is the number of resources}
3: R {is the set of resources (Ri is the ith resource)}
4: CpuPwr(r) {returns the computational power of resource r}
5: Avail(Ri) {returns true if resource Ri is idle}
6: ---CpuInfo algorithm-----
7: maxCpuPwr=CpuPwr(R0);
8: slctRsc=0;
9: for i = 0 to M do
10: if ((Avail(Ri)) AND (CpuPwr(Ri) > maxCpuPwr)) then
11: maxCpuPwr=CpuPwr(Ri);
12: slctRsc=i;
13: end if
14: end for
```

Figure 2. CpuInfo

C. Information-aware machine selection based on availability

In this scenario, the scheduler is able to predict the availability of the resources: namely, it is able to estimate the instant when a resource becomes unavailable. We propose a selection policy, called AvailInfo that selects the resource with the highest availability (that is, the resource that will become unavailable in the latest time). The rationale behind AvailInfo is to avoid (or to delay as much as possible) the occurrence of a machine failure during the execution of a task. In this way, a task can be completed without the resubmission due to the resources failures or, at least, it can live enough to save a "good" checkpoint (that is, a checkpoint which the residual execution time is small). Figure 3 illustrates in detail the AvailInfo policy.

```
1: ----- data structures and functions------
2: M {is the number of resources}
3: R {is the set of resources (Ri is the ith resource)}
4: Avail(Ri) {returns true if resource Ri is idle}
5: Relia(r) {returns reliability of resource r}
6: -----AvailInfo algorithm------
7: maxRelia=Relia(R0);
8: slctRsc=0;
9: for i = 0 to M do
10: if ((Avail(Ri)) AND (Relia(Ri)>maxRelia)) then
11: maxRelia=Relia(Ri);
12: slctRsc=i;
13: end if
14: end for
```

Figure 3. AvailInfo

This algorithm uses a function called Relia(r) (line 5) that returns the reliability level of resource r (that is, the amount of time from now to when the machine will become unavailable). Initially, the algorithm sets the first resource as the selected resource (slctRsc variable on line 8) and its reliability is set as the temporary best reliable resource (maxRelia variable on line 7). In the for cycle (line 9), the algorithm updates the selected resource if another resource is available and has a better reliability (lines 10, 11 and 12). At the end of the cycle, the variable slcRsc contains the selected resource.

D. Information about availability and computational power

In this scenario, the scheduler is able to predict both the availability and the computational power of the resources that is it is able to decide if a specific task can be completed in a specific machine without failure. This is due to the fact that, with full information about the resources, it is possible to calculate the execution time and determine if the task completion time precedes the fault event. We propose a selection policy, called AllInfo, that for each task in the BoT, it selects the available resource with the highest computational power able to complete the task considered without failure. If there is no one resource able to complete the task without failure, the AllInfo scheduling policy selects the most powerful machine (as CpuInfo). Figure 4 illustrates in detail the AllInfo policy. This policy uses a function called ET(r; t) (line 3) that returns the execution time of task t submitted on resource r. Initially, the algorithm sets the first resource as the selected resource (slctRsc variable on line 8) and its computational power is set as the temporary most powerful resource (maxCpuPwr variable on line9). In the for cycle (line

10), the algorithm updates the selected resource if another resource is available, is able to complete the task without failure and, finally, has a better reliability (lines 11, 12 and 13). At the end of the cycle, the variable slctRsc contains the selected resource.

```
1: ----- data structures and functions-----
2: M {is the number of resources}
3: ET(r,t) {is the execution time of task t on resource r}
4: Avail(Ri) {returns true if resource Ri is idle}
5: R {is the set of resources (Ri is the ith resource)}
6: t {is the task selected}
7: ----AllInfo algorithm-----
8: slctRsc=R0;
9: maxCpuPwr=CpuPwr(R0);
10: for i = 0 to M do
11: if ((Avail(Ri)) AND (Relia(Ri)>ET(Ri,t)) AND
(CpuPwr(Ri)>maxCpuPwr)) then
12: maxCpuPwr=CpuPwr(Ri);
13: slctRsc=i;
14: end if
15: end for
```

Figure 4. AllInfo

V. SCHEDULING POLICIES

Combining the task selection policies with the resource selection policies, we obtain 8 different scheduling algorithms (as we can observe in Table I). In this section, we discuss the characteristics of each scheduling policy proposes.

TABLE I

SCHEDULING POLICIES

	NoInfo	**CpuInfo**	**AvailInfo**	**AllInfo**
SmallET	SmallET-NoInfo	SmallET-CpuInfo	SmallET-AvailInfo	SmallET-AllInfo
GreatET	GreatET-NoInfo	GreatET-CpuInfo	GreatET-AvailInfo	GreatET-AllInfo

In order to simplify our discussion throughout the paper we will use a running example in which we consider 4 tasks (whose characteristics are reported in Table II) to be scheduled on 4 machines (whose features are listed in Table III). With this running example, we can observe the various performance of the scheduling algorithms propose in the same scenario.

TABLE II

WORKLOAD EXAMPLE

Task	**Execution Time (ET)**
T0	200
T1	20
T2	100
T3	160

TABLE III

COMPUTATIONAL GRID EXAMPLE

Resource	**Computational Power (CpuPwr)**	**Fault event**
Rsc0	1	100
Rsc1	20	40
Rsc2	10	160
Rsc3	4	20

For each scheduling algorithm presented in this paper, we will show its scheduling decisions, the execution time for each task assignment and we note if the task will be completed without resubmission (that is, the task will not incur in a fault). In Table IV, we show the assignments computed by WQR-FT(first column), the correspondent execution time (second column) and, finally the outcome of the assignment (third column). In particular, for our running example, the WQR-FT scheduling algorithm is able to complete only 2 tasks without resubmission, and we can note some "bad" decision as assign a long task (T0) to a slow resource (Rsc0).

TABLE IV

ASSIGNMENTS COMPUTED BY WQR-FT SCHEDULING Algorithm

Assignment	**Completion Time(CT)**	**Completed**
T0=>Rsc0	200	NO
T1=>Rsc1	1	YES
T2=>Rsc2	10	YES
T3=>Rsc3	40	NO

Combining the policies SmallET and GreatET with NoInfo, we obtain two new scheduling algorithms called SmallET-NoInfo and GreatET-NoInfo. In the first case, as we can observe in Table V, the SmallET-NoInfo scheduling policy selects the task from the shortest to the longest (that is, the task order selection is T1, T2, T3 and T0) assigning them to the available resources. For simplicity, we assume that the selection order for the resources is Rsc0, Rsc1, Rsc2 and Rsc3. Thus, SmallET-NoInfo completes the following

assignments: T1 => Rsc0, T2 => Rsc1, T3 => Rsc2 and T0 => Rsc3. Comparing these assignments with the WQR-FT, we note that SmallET is able to complete one more task without resubmission, and the greatest completion time is equal to 50 seconds (T0 => Rsc3).

TABLE V

ASSIGNMENTS COMPUTED BY SMALLET-NOINFO SCHEDULING ALGORITHM

Assignment	Completion Time (CT)	Completed
T1 => Rsc0	20	Yes
T2 => Rsc1	5	Yes
T3 => Rsc2	16	Yes
T0 => Rsc3	50	No

Conversely, the GreatET-NoInfo selects the tasks in this order: T0, T3, T2 and T1. As we can observe in Table VI, the GreatET-NoInfo is able to complete 3 tasks over 4, and the greatest completion time is equal to 100 (T0 => Rsc0). Thus, the GreatET-NoInfo is better than WQR-FT (since GreatET-NoInfo completes one more task) but worse than SmallET-NoInfo (SmallET-NoInfo completes the same number of tasks and the greatest completion time is 50)

TABLE VI

ASSIGNMENTS COMPUTED BY GREATET-NOINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T0 => Rsc0	200	No
T3 => Rsc1	8	Yes
T2 => Rsc2	10	Yes
T1 => Rsc3	5	Yes

Supposing to have information about computational power of the resources, we can combine the task selection policy with CpuInfo obtaining two new scheduling policies: SmallET-CpuInfo (that assigns the task with the smallest residual time to the most powerful resource available) and greatET-CpuInfo (that assigns the task with the greatest residual time to the most powerful resource available). The rationale behind the SmallET-CpuInfo is to complete the smallest tasks first in order to have more machines available to send replicas of the greater tasks as soon as possible. Conversely, the rationale behind the GreatET-CpuInfo is to assign the greatest tasks to the fastest machine to complete them as soon as possible since the average BoT completion time often depends on the completion of the greatest task. Observing the data in Table VII, we note that SmallET-CpuInfo scheduling algorithm is able to complete just two tasks without resubmission and the slowest execution time is equals to 100 (T0 => Rsc0)

TABLE VII

ASSIGNMENTS COMPUTED BY SMALLET-CPUINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T1 => Rsc1	1	Yes
T2 => Rsc2	10	Yes
T3 => Rsc3	40	No
T0 => Rsc0	200	No

Conversely, the GreatET-CpuInfo scheduling algorithm has better performance with respect to SmallET-CpuInfo: as we can observe from Table VIII, the GreatET-CpuInfo is able to complete three tasks over four without resubmission and the slowest completion time is just 25 seconds (T2 => Rsc3).

TABLE VIII

ASSIGNMENTS COMPUTED BY GREATET-CPUINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T0 => Rsc1	10	Yes
T3 => Rsc2	16	Yes
T2 => Rsc3	25	No
T1 => Rsc0	20	Yes

Other two scheduling policies can be obtained combining the task select policies with AvailInfo. In this case, we have SmallET-Avail Info that assigns the smallest task to the most reliable machines and GreatET-AvailInfo that assign the greatest task to the most reliable machine. In Table IX, we observed the assignments of the SmallET-AvailInfo scheduling policy: the algorithm is able to complete three tasks within 50 seconds (T2 =>Rsc0). Conversely, the GreatET-Avail Info scheduling policy is able to complete three tasks and the slowest execution time is 80 seconds (T3 =>Rsc0), as we can observe in Table X. Finally, if the scheduler is able to predict both the availability and the computational power of the resources, it can use two new scheduling policies: SmallET-AllInfo and GreatET-AllInfo. In Table XI, we can observe the scheduling decisions of SmallET-AllInfo. This scheduling algorithm is able to complete just two tasks and the slowest execution time is 100 second

TABLE IX

ASSIGNMENTS COMPUTED BY SMALLET-AVAILINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T1 => Rsc2	2	Yes
T2 => Rsc0	100	Yes
T3 => Rsc1	8	Yes
T0 => Rsc3	50	No

TABLE X

ASSIGNMENTS COMPUTED BY GREATET-AVAILINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T0 => Rsc2	20	Yes
T3 => Rsc0	160	No
T2 => Rsc1	5	Yes
T1 => Rsc3	5	Yes

TABLE XI

ASSIGNMENTS COMPUTED BY SMALLET-ALLINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T1 => Rsc1	1	Yes
T2 => Rsc2	10	Yes
T3 => Rsc3	40	No
T0 => Rsc0	200	No

(T0 =>Rsc0). In Table XII, we can observe the scheduling decisions of GreatET-AllInfo. This scheduling algorithm is able to complete all tasks the slowest execution time is 50 seconds (T2 => Rsc0). In our example,GreatET-AllInfo is the only one scheduling policy that is able to complete all tasks without resubmission, as we can observe in the summary table XIII. In this last table, for each scheduling policy, we report the number of tasks completed without resubmission and the time necessary to complete them.

TABLE XI1

ASSIGNMENTS COMPUTED BY GREATET-ALLINFO SCHEDULING ALGORITHM

Assignment	Completion Time(CT)	Completed
T0 => Rsc1	10	Yes
T3 => Rsc2	16	Yes
T2 => Rsc0	100	Yes
T1 => Rsc3	5	Yes

TABLE XIII

SUMMARY OF THE SCHEDULING ALGORITHMS PERFORMANCE FOR OUR RUNNING EXAMPLE

Scheduling policy	Final Completion Time(FCT)	Num. of tasks completed
WQR-FT	10	2
SmallET-NoInfo	20	3
GreatET-NoInfo	10	3
SmallET-CpuInfo	10	2
GreatET-CpuInfo	20	3
SmallET-AvailInfo	100	3
GreatET-AvailInfo	20	3
SmallET-AllInfo	10	2
GreatET-AllInfo	100	4

VI. PERFORMANCE ANALYSIS

In order to assess the performance of the scheduling policies proposed, we compare them with the plain WQR-FT. Our comparison is based on metric: average no of completed tasks

A. Results

In this section, we described the results we obtained in our experiments. In order to verify if the policies proposed achieve better performance than plain WQR-FT, we performed a set of experiments in which we progressively increase the number of tasks for each bag (the parameter called RR), computed the average No of completed tasks with respect to WQR-FT.

In particular, Figure 5(a) illustrates the average No of completed tasks of SmallET-NoInfo and GreatET-NoInfo relative to WQR-FT, for different values of RR. As we can observe, the GreatET-NoInfo & SmallET-NoInfo scheduling policy achieves better performance with respect to WQR-FT.

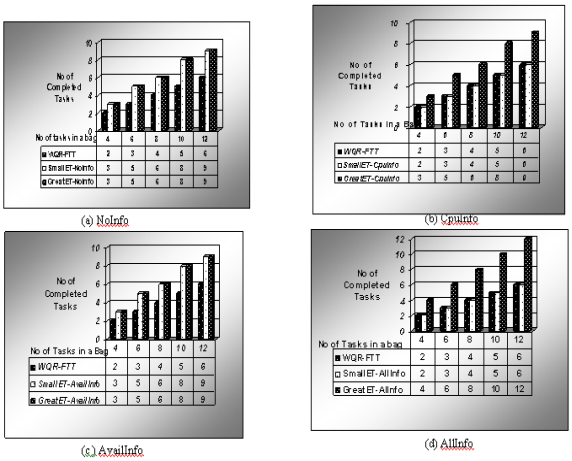

Fig. 5 Average No of completed tasks with respect to WQR-FT

In particular, when RR = 4, GreatET-NoInfo is able to complete the BoT almost 25% faster than WQR-FT.

In Figure 5(b),(c),(d), we illustrates the performance of the scheduling policies for the various levels of information We can observe that the policies GreatET-* (that is, GreatET-NoInfo, GreatET-CpuInfo, GreatET-AvailInfo) achieve similar performance independently from the information about the resources & the policies SmallET-* (that is, SmallET -NoInfo, SmallET- AvailInfo) achieve similar performance independently from the information about the resource.

In Figure 5(b), we observe the average BoT completion time of the scheduling strategies SmallET-CpuInfo and GreatET-CpuInfo relative to WQR-FT. Choosing the resource based on their computation power introduce benefits for GreatET -CpuInfo scheduling policiy. For instance GreatET-CpuInfo outperforms WQR-FT and this is due, once again, to the importance of having information about the computational power of the resources. Observing Figure 5(c), we can note that the information about the availability is important than the information about the computational power. As a matter of the fact, the

SmallET-Availinfo achieves better performances than SmallET-CpuInfo.Finally, combining availability information and computational power of the resource we are able to achieve good performance. For instance GreatET-AllInfo outperforms GreatET-CpuInfo and this is due, once again, to the importance of having information about the computational power & availability information of the resources. GreatET-AllInfo is the best scheduling policy in term of the average BoT completion time

B. References

S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang. Characterizing and Classifying Desktop Grid. In CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, pages 743–748, Washington, DC, USA, 2007. IEEE Computer Society

D. Kondo, A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In Proc. of Super Computing Conference, 2004.

Real R., Yamin A. et al., "Resource scheduling on Grid: handling uncertainty", Proceedings of the fourth international workshop on Grid Computing, 2003

J. Smith and S. Srivastava. A System for Fault-Tolerant Execution of Data and Compute Intensive Programs over a Network of Workstations. In Proc. of EuroPar'96, volume 1123 of Lecture Notes in Computer Science, 1996

D. da Silva, W. Cirne, and F. Brasileiro. Trading Cycles fro Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In Proc. of EuroPar 2003, volume 2790 of Lecture Notes in Computer Science, 2003

C. Anglano, J. Brevik, M. Canonico, D. Nurmi, and R. Wolski. Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids. In Proc. of 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, Sept. 2006. IEEE Press.

C. Anglano and M. Canonico. Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications. In Proc. of the 2005 European Grid Conference, number 3470 in Lecture Notes in Computer Science, Amsterdam, The Netherlands, Feb. 2005. Springer, Berlin

D. Kondo, A. Chien, and H. Casanova. Scheduling Task Parallel Applications for Rapid Application Turnardound on Enterprise Desktop Grids. Journal of Grid Computing, 2007.

Y. C. Lee and A. Y. Zomaya. Practical scheduling of bagof- tasks applications on grids with dynamic resilience. IEEE Trans. Comput., 56(6):815–825, 2007.

D. Zhou and V. Lo. Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-Based Desktop Grid Systems. In Proc. of 11thWorkshop on Job Scheduling Strategies for Parallel Processing, number 3834 in Lecture Notes in Computer Science, Boston, MA, USA, June 2005. Springer, Berlin.