BACKFILLING STRATEGIES FOR SCHEDULING STREAMS OF JOBS ON COMPUTATIONAL FARMS

A.D.Techiouba, G.Capannini, Ranieri Baraglia, D.Puppin, M.Pasquali
ISTI,CNR
Via Moruzzi, 1
Pisa, Italy
techioub@cli.di.unipi.it
(gabriele.capannini, ranieri.baraglia, diego.puppin, marco.pasquali)@isti.cnr.it

Laura Ricci
Department of Computer Science
Largo B.Pontecorvo
Pisa, Italy
ricci@di.unipi.it

Abstract This paper presents a set of strategies for scheduling a stream of batch jobs on the machines of a heterogeneous computational farm. Our proposal is based on a flexible backfilling, which schedules jobs according to a priority assigned to each jobs submitted for the execution. Priority values are computed as a result of a set of heuristics whose main goal is both to improve resources utilization and to meet the QoS requirements of the jobs. The heuristics consider the deadlines of the jobs, their estimated execution time and aging in the scheduling queue. Furthermore, the set of software licenses required by a job is also considered. The different proposals have been compared through simulations. Performance figures show the applicability of our approach.

Keywords: Scheduling, Grid Computing, Resource Management, Quality of Service

1. Introduction

In this paper we propose a set of strategies for scheduling a stream of batch jobs on the machines of a heterogeneous computational farm. A computational farm can integrate hw/sw heterogeneous resources such as workstations, parallel systems, servers, storage arrays, and software licenses. In such an environment, users should submit their computational requests without neces-

sarily knowing on which computational resources these will be executed. A fruitful exploitation of a computational farm requires scheduling algorithms able to efficiently and effectively allocate the user jobs on the computational resources.

Our proposal is based on the backfilling technique [2], which has been initially introduced for scheduling streams of jobs on parallel supercomputers.

Backfilling has been originally introduced to extend the *First Come First Served* ($FCFS$) approach in order to increase the efficiency of the resources usage. Backfilling improves resource utilization by allowing the first job J of the queue to reserve resources that are not available and by evaluating the possible execution of successive jobs in the submission queue. These jobs can be executed if and only if they do not exploit the resources reserved by J or their execution terminates within the *shadow time*, i.e. the time where all the reserved resources becomes available. This scheduling strategy requires the knowledge of an estimation of the execution time of any job.

In this paper we propose a set of extensions to the original backfilling introduced to support heterogeneity. The basic idea of our approach is to assign a priority to each job in the submission queue by considering both the optimization of the usage of the system resources and a set of QoS requirements of the jobs. Job priorities are computed at each scheduling events, i.e. at a job submission and at a job ending, by using a set of *heuristics*.

For instance, our solution supports job deadlines by dynamically increasing the priority of a job when its deadline is approaching, and by minimizing the priority of a job when its deadline is exceeded. Furthermore, it also takes into account the type of resources required by each job. We have considered, for instance, the set of software licenses required for the execution of any job. In a heterogeneous environment, some software licenses may require a specific operating system or may be installed on a specific machine only. Furthermore, a maximum number of copies of a software license, which can be simultaneously utilized, is often defined. This value is generally smaller than the number of machines of the computational farm. In this case the license requirements of the jobs must be considered in the scheduling process in order to optimize the usage of such resources. Our scheduler detects critical licenses, i.e. licenses whose number of concurrently usable copies is smaller than the number of copies required by the jobs in the submission queue, and assigns to each job a priority proportional to the amount of critical licenses it requires. Since, jobs requiring a large number of critical licenses release them after their termination, they should be executed as soon as possible in order to let the scheduler to define more flexible scheduling plans. Other heuristics reducing job starvation and improving job response time are defined as well.

We have developed two different extensions of the original backfilling algorithm. In the first one, the maximum priority is always assigned to the first job

in the queue. Furthermore, this job may reserve all the unavailable resources it needs and the reservation is preserved even if a higher priority job is submitted. The other jobs in the submission queue are ordered according to their priority and are considered by the scheduler according to the standard backfilling algorithm. As in backfilling, this strategy prevents starvation.

According to the second extension the job J with the highest priority is able to make resource reservation, but such reservation may be canceled if a job with a higher priority is submitted or if the priority of a previously submitted job is updated and exceeds that of J. In this way another job may be moved to the first position of the queue.

We have developed an event driven ad-hoc simulator to evaluate the different versions of the proposed schedulers.

Section 2 reviews some proposals based on backfilling. In Section 3 we describe the target architecture considered, while Section 4 introduces the heuristics designed to compute the job priorities. These heuristics are exploited in the algorithms described in sections 5 and 6. Section 7 shows experimental results. Finally, Section 8 describes conclusions and future works.

2. Related Work

First Come First Served (FCFS) is one of the simplest approach to job scheduling [12]. FCFS schedules jobs according to their submission order and checks the availability of the resources required by any job. If all the resources required by a job J are available, J is immediately scheduled for the execution, otherwise it is queued. Any job submitted while J is waiting for the execution is queued, even if the resources it requires are available. Despite its simplicity, this approach presents several advantages. FCFS does not require an estimation of the execution times of the jobs and its implementation is straightforward. Furthermore, it guarantees that the response time of a job J, i.e. the time elapsed between the submission time of the job and its termination time, does not depend on the execution times of the jobs submitted later. On the other hand, this fairness property implies a low utilization of the system resources, because a submitted job cannot be executed if the job queue is not empty, even if the resources it requires are all available.

The main goal of the *backfilling approach* is to improve FCFS by increasing the utilization of the system resources and by decreasing the average waiting time of the job in the queue of the scheduler [13]. Different variants of the basic backfilling approach have been proposed.

The *Conservative Backfilling* approach allows *each* job to reserve the resources it needs, when it is inserted into the job queue [13]. A job may be executed before those previously submitted, if its execution does not violate the reservations made by such jobs. This strategy improves system usage by

allowing jobs requiring a few available resources for a short time to overtake longer jobs in the queue. This way, the order of submission may be violated only if overtaking jobs do not delay the execution of jobs submitted earlier.

In a popular variant of backfilling, the *EASY (Extensible Argonne Scheduling system)* scheduler [3], developed for the IBM SP2 supercomputer, only the first job in the submission queue is allowed to reserve the resources it needs. This approach is more "aggressive" because it increases resource utilization, even if it does not guarantee that a job is not delayed by another one submitted later.

Most backfilling strategies consider jobs candidate both for execution and for backfilling according to a FCFS strategy. An alternative solution is introduced in *Flexible Backfilling*. Here jobs are prioritized, according to some policy. In [7] a backfilling solution combines three kind of priorities, an administrative, a user and a scheduler priority. The first two classes of priorities give the administrator and the user, respectively, the possibility to favor a class of jobs. The scheduler priority is introduced to guarantee that no job is starved.

Currently, many of these algorithms are exploited in commercial and open source job schedulers [11], such as Maui scheduler [14][5], and Portable Batch System [14]. However, none of these schedulers deal with an entire wide range of constraints and user requirements.

3. The System Model

The target architecture considered in this paper is a large computational farm, where each machine may be mono-processor or multi-processor. Each machine is characterized by its computational power and executes jobs using space-sharing (SS). In SS, the set of processors in a machine is partitioned and each partition is assigned to the exclusive use of a job. All the jobs are considered not preemptible.

Even if the proposed backfilling algorithms do not look for the best matching among jobs and machines, the machines of the farm are ordered according to their computational power in order to exploit the most powerful first. This strategy does not balance the computational load, but favors the response time of the jobs. We suppose that a set of software licenses may be activated on the machines of the computational farm. Each license may be activated on a subset of the machines of the computational farm, for instance because it requires a specific operating system or a specific CPU. Furthermore, the number of software licenses of a specific type is generally smaller than the number of machines on which they may be activated. These are *floating* licenses because they are activated on the proper machines according to the job requests. On the other hand, since non floating licenses are permanently activated on a machine, they may be considered like any other resource characterizing that machine.

Each job requires a set of software licenses for its execution and may be executed only on the subset of machines where all the required licenses may be activated. Any submitted job is characterized by its estimated execution time and may require a deadline for its execution.

4. Heuristics

This section defines a set of heuristics to assign priorities to submitted jobs. The main goal of this assignment is to fulfill a set of users and system administrator QoS requirements. Users may require, for instance, the compliance with job deadlines, while the goal of the system administrator is to optimize the use of the system resources. The value of the priority $P(J)$ assigned to each job J is the weighted sum of the values computed by each heuristics. This value may be dynamically modified at each scheduling session. We have defined the following heuristics: *Minimal Requirements*, *Aging*, *Deadline*, *Licenses*, and *Response*.

The Minimal Requirements heuristics fixes the associations among jobs and machines. It selects the set of machines that has the computational requirements suitable to perform a job. In our study, we considered only the following requirements: number of processors and sw licenses available on a machine.

The goal of the Aging heuristics is to avoid job starvation. For this reason higher scores are assigned to those jobs which have been present in the queue for a longer time. The value of the priority assigned to job J is increased as follow:

$$P(J) += age\_factor \cdot age(J)$$
$$age(J) = wall\_clock - submit(J)$$

where age_factor is a multiplicative factor set by administrator according to the adopted system management policies, $wall\_clock$ is the value of the system wall-clock when the heuristics is computed and *submit(J)* is the time when the job is submitted to the scheduler.

The main goal of the Deadline heuristics is to maximize the number of jobs, which terminates their execution within their deadline. It requires an estimation of the execution time of each job in order to evaluate their completion times, with respect to the current wall-clock time. The heuristics assigns a minimal value to any job whose deadline is far from its estimated termination time. When the distance between the completion time and the deadline is smaller than a threshold value, the score assigned to the job is increased in inverse proportion with respect to the distance. The threshold value may be tuned according to the importance assigned by the scheduler to this heuristics. Finally, if the job goes over its deadline before it is scheduled, its score is set to 0. As said before, a job is scheduled on the first available most powerful machine. Since, jobs with a closer deadline receive higher priority, this strat-

egy should improve the number of jobs executed within their deadline. Let $ex\_execution(J)$ be the estimated execution time of job J, and $dline(J)$ the deadline for the execution of J. Let us define

$$\begin{array}{ll} t_e(J) & = ex\_execution(J) + wall\_clock \\ over\_ex\_t(J) & = k \cdot ex\_execution(J) \quad \textit{with } k > 1 \\ t_s(J) & = dline(J) - over\_ex\_t(J) \\ \alpha(J) & = (max - min)/over\_ex\_t(J) \end{array}$$

where $t_e(J)$ denotes the estimated termination time of the job with respect to the current wall-clock, $over\_ex\_t(J)$ denotes an overestimation of the estimated execution time of job J, and $t_s(J)$ denotes the time corresponding to the threshold value of the distance from the deadline. $\alpha(J)$ is the growing factor computed according to the predefined range of assignable scores, and $over\_ex\_t(J)$. The value $P(J)$ is increased by the Deadline heuristics according to the following formula:

$$P(J)+= \begin{cases} min & \textit{if } t_e(J) < t_s(J) \\ min + \alpha(J) \cdot (t_e(J) - t_s(J)) & \textit{if } t_s(J) \leq t_e(J) \leq dline(J) \\ 0 & \textit{if } t_e(J) > dline(J) \end{cases}$$

The Licenses heuristics assigns a higher score to jobs requiring a larger amount of critical resources. The rationale is that when these jobs end their execution, a set of licenses may become non critical and the scheduler is able to compute more flexible scheduling plans. Let us define

$$\begin{array}{ll} \rho(l) & = requests(l)/total(l) \\ l_c(J) & = \{l \in \textit{licences required by } J : \rho(l) > 1\} \\ l_{\bar{c}}(J) & = \{l \in \textit{licenses required by } J : \rho(l) \leq 1\} \end{array}$$

$P(J)$ is increased in according to this formula:

$$P(J)+= \sum_{l \in l_{\bar{c}}} \rho(l) + d \cdot \sum_{l \in l_c} \rho(l)$$

where $d = max\{|\cup_{\forall J}\, l_{\bar{c}}(J)|, 1\}$.

Finally, the Wait Minimization heuristics favors jobs with the shortest estimated execution time. The rationale is that shorter jobs are executed as soon as possible in order to release the resources they have reserved and to improve the average waiting time of the jobs in the scheduling queue. Let $priority\_boost\_value$ be the factor set by administrator according to system managment policies and $min\_ex\_t = min\{ex\_execution(J) : J \in scheduling\_queue\}$, the value of $P(J)$ is increased by the heuristics as follows:

$$P(J)+= priority\_boost\_value \cdot \frac{min\_ex\_t}{ex\_execution(J)}$$

5. The BF_UNMOD Scheduler

BF_UNMOD implements a Flexible Backfilling strategy by assigning the highest priority to the first job in the queue and by ordering the remaining jobs according to the priority assigned by the heuristics introduced in the previous section. The first job of the queue preserves the highest priority until its execution starts, while the rest of the queue is reordered at each scheduling event. Like Easy Backfilling, BF_UNMOD adopts an "aggressive" strategy by enabling reservations for the first job in the queue only. The algorithm exploits priorities to improve jobs QoS and efficiency in the usage of the system resources. For instance, the priority of jobs approaching to their deadline is increased at each scheduling session. By increasing the priority of jobs exploiting critical licenses, *BF_UNMOD* increases efficiency.

6. The BF_MOD Scheduler

BF_MOD differs from BF_UNMOD because it preserves the reservation, at least, until a job with a higher priority is submitted. When a job J reaches the first position within the queue, it is allowed to reserve all the resources it needs. Further jobs are ordered according their priority and they can be used for backfilling. At next scheduling event, the reservation made by J is preserved if and only if BF_MOD assigns the highest priority to J. Otherwise, another job with the highest priority is allowed to reserve resources. Suppose, for instance, that a job with a forthcoming deadline is submitted. BF_MOD schedules this job as soon as possible by canceling the reservations of the first job in the queue at the next scheduling event. On the other way, the prediction of the starting execution time of a job is more difficult. A simple way to avoid job starvation is to increase the weight computed by the Aging heuristics.

7. Experimental Results

In this section we present the evaluation conducted to investigate the effectiveness of the scheduling solutions carried out by the proposed schedulers. The evaluation was conducted by simulations using different streams of jobs which inter-arrival times are generated according to a negative exponential distribution with a different parameter. To conduct our evaluation we developed an event driven ad-hoc simulator. For each simulation, we randomly generated a list of jobs and machines whose parameters are generated according to a uniform distribution in the ranges:

- Job Estimated execution time [500 $\div$ 3000].
- Job Deadline Margin [30 $\div$ 250].
- Number CPUs [1 $\div$ 8] required by a job or available on a machine.

- License Ratio [50% ÷ 70%] specifies the maximum number of concurrently usable copies of a sw license.
- License Suitability [90%] specifies that a sw license is usable on a machine.
- License Needs [30%] specifies the probability that a job needs a sw license.
- Number of jobs without deadline 30%.

Tests were conducted by simulating a cluster of 100 machines, 20 sw licenses, 1000 jobs, and using five job streams generated with average jobs inter-arrival time fixed equal to 4, 6, 12, 24 and 48 simulator time unit. Each stream leads to a different system workload (computed as the sum of the number of jobs ready to be executed and the number of the jobs in execution) through a simulation run. The closer job inter-arrival time is, the higher the contention in the system is. To obtain stable values each simulation was repeated 20 times with different job attributes values.

To evaluate the schedules carried out by BF_MOD and BF_UNMOD, we have considered the following metrics:

- *System Usage*. This measures the efficiency of the system, and it is defined as follows:

$$System\_Usage = \frac{\sharp CPU\_in\_use}{min(\sharp total\_CPUs, \sharp jobs\_in\_system)}$$

  where $\sharp CPU\_in\_use$ is the number of CPUs executing a job, $\sharp total\_CPUs$ is the available total number of CPUs, and $\sharp jobs\_in\_system$ sums the number of waiting jobs and those in execution.

- *Out Deadline*. This measures the number of jobs executed without respecting their deadline. This does not include jobs which must not be executed within a given deadline.

- *Slow Down*. This measures the ratio between the response time of a job, i.e. the time elapsed between its submission and its termination, and its execution time. It is defined as follows:
  - $SlowDown(j) = \frac{Tw_j + Texec_j}{Texec_j}$
  - $SlowDown = E[SlowDown(J)]$

  where Tw_j is the time spent by job J in the scheduling queue, and $Texec_j$ is the execution time of a job J.

We have compared BF_MOD and BF_UNMOD with FCFS and with BF_FCFS, which is an implementation of *EASY* backfilling. The implementation of these different versions of backfilling differ with respect to classical algorithm because of the target architecture which is an heterogeneous architecture rather than a homogeneous multiprocessor machine. As a consequence, jobs considered by our algorithms may require different sw/hw resources like, licenses. The original backfilling algorithms have been modified to consider all these resources when defining a scheduling plan.

In Figure 1 and Figure 2 the results obtained for the different strategies with respect to the metrics previously defined are compared. Figure 1(a) shows the percentage of the jobs executed that do not respect their deadline. It can be seen that BF_MOD and BF_UNMOD obtain better results in each test. When the available computational power is able to maintain low the system contention (i.e. for 24, 48 average job inter-arrival times), the use of backfilling technique leads to a higher system usage, which permits to improve the percentage of the jobs that are executed respecting their deadline. On the other hand, when the system contention is higher (i.e. for 4, 6, 12 average job inter-arrival times) the exploitation of the job priority leads to better results. Figure 1(b) shows the percentage of system usage. It can be seen that backfilling technique leads to a better system usage, in particular when the system contention is higher.

Figure 2(a) shows the slow down trend through simulation runs. It can be seen that the backfilling technique is enough to drastically reduce the average job waiting time.

Figure 2(b) shows the percentage of both the sw licenses usage and the number of jobs executed out of their deadline by changing the License Ratio parameter.

The first table shows that to assign higher priorities to jobs requiring a higher number of critical sw licenses leads to an improvement only when the sw licenses contention is high. When the sw licenses contention decreases the proposed schedulers lead to worse results. This because jobs requiring a fewer number of critical licenses, but with a closer deadline, receives higher priorities delaying the execution of jobs requiring a higher number of sw licenses but with a far deadline.

The second table shows that, when the contention on sw license increases, the scheduler changing the reservation for the first queued job at each job scheduling event permits us to obtain a higher number of jobs executed respecting their deadlines.

The experimental results show the applicability of the proposed strategy. Both BF_UNMOD and BF_MOD outperforms FCFS and BF_FCFS in term of system usage and in the number of jobs that are scheduled respecting the proposed deadline. The differences are strong at any load level (i.e. job inter-arrival time). With higher inter-arrival times, i.e. when the scheduling is less

critical, BF_FCFS shows a performance similar to our two strategies.

BF_UNMOD and BF_MOD do not greatly improve the slow-down over BF_FCFS, which already does a very good scheduling job w.r.t. standard FCFS.

We were not able to measure any difference between BF_UNMOD and BF_MOD. This means that the simpler approach followed by BF_UNMOD is sufficient for the task at end. We are investigating this issue.

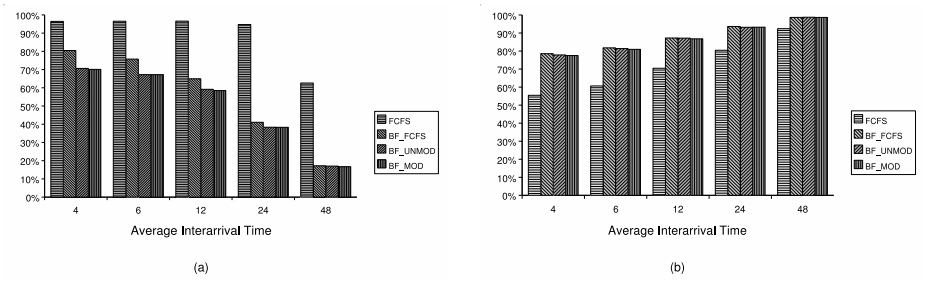
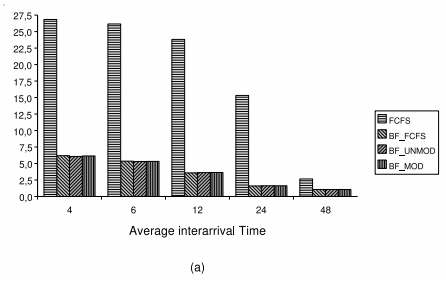


Figure 1. (a) shows the percentage of the jobs executed that do not respect their deadline. (b) shows the percentage of used system hw resources



LICENSE RATIO	% LICENSE USAGE			
	FCFS	BF_FCFS	BF_UNMOD	BF_MOD
30%-50%	55,7	78,5	80,5	79,7
40%-60%	52,1	76,1	75,5	76,3
50%-70%	49,5	74,8	72,2	72,4

LICENSE RATIO	% of JOBS OUT THEIR DEADLINE			
	FCFS	BF_FCFS	BF_UNMOD	BF_MOD
30%-50%	95,6	78,9	74,5	73,5
40%-60%	95,5	74,8	69,4	69,0
50%-70%	95,5	74,6	66,2	66,2

(b)

Figure 2. (a) shows the slow down trend (b) shows the tables representing the percentage of used licenses and relative percentage of the jobs executed that do not respect their deadline.

8. Conclusion and Future Work

In this work, we presented a set of extensions to the Backfilling Scheduling algorithm, designed to allow scheduling over heterogeneous resources. Our BF_MOD and BF_UNMOD strategies extend Flexible Backfilling, by utilizing a variety of heuristics to re-assign priorities to queued jobs.

Our proposed heuristics covered deadline requirements, license usage, aging (to prevent starvation). We designed two schedulers: one reassign the priorities of all jobs at every scheduling event, the other keeps the reservation of the first job fixed (unless another job gets higher priorities).

The proposed strategies outperform BF_FCFS, with a bigger margin for heavy workloads. We are investigating the relative value of BF_UNMOD and BF_MOD.

9. Acknowledgment

This work is in the activity of the European CoreGRID NoE (European Research Network on Foundations, Software Infrastructures and Applications for Large Scale, Distributed, GRID and Peer-to-Peer Technologies, contract IST-2002-004265)

References

[1] D. Talby, D.G: Feitelson, Supporting Priorities and Improving Utilization of the IBM SP using slack based backfilling in 13-th Parallel Processing Symposium,pp. 513-517, April 1999.

[2] D.Tsafrir, Y.Etsion, D.Feitelson,Backfilling Using System-Generated Predictions Rather than User Runtime Estimates, *IEEE Transactions on Parallel and Distributed Systems*, vol.18, n. 6, pp. 789-803, June 2006.

[3] D. Lifka, The ANL/IBM SP scheduling system, in *1st Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, D. G. Feitelson and L. Rudolph (eds.), pp. 295-303, Springer-Verlag, Apr 1995. Lect. Notes Comput. Sci. vol. 949.

[4] W. Gentzsch, Sun grid engine: towards creating a compute power grid , in IEEE Symp. on Cluster Computing and the Grid (CCGrid), pp. 35-36,May 2001.

[5] D. Jackson, Q. Snell, and M. Clement, Core algorithms of the Maui scheduler. In *7th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, D. G. Feitelson and L. Rudolph (eds.), pp. 87-102, Springer-Verlag, Jun 2001. Lect. Notes Comput. Sci. vol. 2221.

[6] A. W. Mu'alem and D. G. Feitelson Utilization, Predictability, Workloads and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling *IEEE Trans. Parallel and Distributed Systems 12(6)*, pp. 529-543, Jun 2001.

[7] S-H. Chiang, A. Arpaci-Dusseau, M. K. Vernon, *The impact of more accurate requested runtimes on production job scheduling performance*. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 103-127, Springer Verlag, LNCS, vol. 2537, 2002.

[8] Markov TwoStage Optimization of Job Scheduling and assignment in Heterogeneous Compute Farm *10th IEEE International Workshop on Future Trends of Distributed Computing Systems)*, 2004

[9] Q.O.Snell, M.J.Clement, D.B.Jackson 'Preemption Based Backfill in *8th International Workshop on Job Scheduling Strategies for Parallel Processing*

[10] I.Foster, C.Kesselman "The Grid: Blueprint for a new Computing Infrastrucure"(2nd edition), Morgan Kaufmann Publishers, 1999.

[11] Yoav Etsion, Dan Tsafrir, *A Short Survey of Commercial Cluster Batch Schedulers*, Technical Report 2005-13, School of Computer Science and Engineering, The Hebrew University of Jerusalem, May 2005.

[12] Uwe Schwiegelshohn and Ramin Yahyapour, *Analysis of first-come-first-serve parallel job scheduling*, In SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[13] D. Feitelson, L. Rudolph, and U. Schwiegelshohn, *Parallel job scheduling - a status report*, June 2004.

[14] Dan Tsafrir Yoav Etsion, *A short survey of commercial cluster batch schedulers*, School of Computer Science and Engineering, Jerusalem, The Hebrew University, Israel.