COMPARISON OF MULTI-CRITERIA SCHEDULING TECHNIQUES

Dalibor Klusáček and Hana Rudová
Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
{xklusac, hanka}@fi.muni.cz

Ranieri Baraglia[1] and Marco Pasquali[1,2] and Gabriele Capannini[1]
[1] *ISTI - Institute of the Italian National Research Council,*
Via Moruzzi 1, Pisa, Italy
{ranieri.baraglia, marco.pasquali, gabriele.capannini}@isti.cnr.it

[2] *IMT Lucca, Lucca institute for advanced studies,*
Lucca, Italy
marco.pasquali@imtlucca.it

Abstract

This paper proposes a novel *schedule-based* approach for scheduling a continuous stream of batch jobs on the machines of a computational Grid. Our new solutions represented by dispatching rule Earliest Gap—Earliest Deadline First (EG-EDF) and Tabu search are based on the idea of filling gaps in the existing schedule. EG-EDF rule is able to build the schedule for all jobs incrementally by applying technique which fills earliest existing gaps in the schedule with newly arriving jobs. If no gap for a coming job is available EG-EDF rule uses Earliest Deadline First (EDF) strategy for including new job into the existing schedule. Such schedule is then optimized using the Tabu search algorithm moving jobs into earliest gaps again. Scheduling choices are taken to meet the Quality of Service (QoS) requested by the submitted jobs, and to optimize the usage of hardware resources. We compared the proposed solution with some of the most common queue-based scheduling algorithms like FCFS, EASY backfilling, and Flexible backfilling. Experiments shows that EG-EDF rule is able to compute good assignments, often with shorter algorithm runtime w.r.t. the other queue-based algorithms. Further Tabu search optimization results in higher QoS and machine usage while keeping the algorithm runtime reasonable.

Keywords: Grid, Scheduling, Dispatching Rule, Local Search, Backfilling

1. Introduction

The building of a Grid infrastructure requires the development and deployment of middleware, services, and tools. At middleware level the scheduler is central to efficiently and effectively schedule jobs on available resources. It should both maximize the overall resource utilisation and guarantee nontrivial QoS for the user's applications. The scheduling problem has shown to be *NP-complete* in its general as well as in some restricted forms. Moreover, to meet the QoS requirements of applications flexible scheduling mechanisms are required. A typical QoS requirement is the time at which user wants to receive results, i.e., the job turnaround time.

In the past, a lot of research effort has been devoted to solve both static and dynamic job scheduling problems. Many of the proposed algorithms, such as backfilling, are queued-based techniques. Current production systems like PBS [7], Condor [21], LSF [23] or meta-scheduling systems such as Grid Service Broker [22], GridWay [11] and Moab [6] are mostly queue-based solutions. On the other hand, solutions using *schedule-based* [10, 19] approaches are poorly investigated, in particular to solve dynamic job scheduling problems [4]. In dynamic environments, such as Grids, resource may change, jobs are not known in advance and they appear while others are running. Schedule-based approach allows precise mapping of jobs onto machines in time. This allows us to use advanced scheduling algorithms [16, 8] such as local search methods [8] to optimize the schedule. Due to their computational cost, these approaches were mostly applied to static problems, assuming that all the jobs and resources are known in advance which allows to create schedule for all jobs at once [2–3]. CCS [10] as well as GORBA [19] are both advanced resource management systems that use schedule instead of a queue(s) to schedule workflows (GORBA), or sequential and parallel jobs while supporting the advanced reservations (CCS). GORBA uses simple policies for schedule creation and an evolutionary algorithm for its optimisation while CCS uses FCFS, Shortest/Longest Job First when assigning jobs into the schedule and a backfill-like policy that fills gaps in the constructed schedule. Both CCS and GORBA re-compute the schedule from scratch when a dynamic change such as job arrival or machine failure appears. It helps to keep the schedule up to date, however for large number of jobs this approach may be quite time consuming as was discussed in case of GORBA [17]. Works [1] and [18] propose local search based methods to solve Grid scheduling problems. The schedule is kept valid in time without total re-computation, however no experi-

mental evaluation was presented in [1], and [18] does include resource changes but no dynamic job arrivals.

In this paper we propose novel schedule-based solutions to schedule *dynamically arriving* batch jobs on the machines of a computational Grid. In comparison with other approaches [10, 19], we are using *dispatching rule* and *local search* in an *incremental* fashion [13]. It means that current computed schedule can be used as the starting point for building a new schedule after each job arrival. This leads to a reasonable computational cost since the schedule is not rebuilt from scratch. We propose a multi-criteria approach which is based on providing non-trivial QoS to the end users, while satisfying the system administrators requirements as well. User requirements are expressed by the objective function focusing on maximising the number of jobs that meet their deadline, while system administrators needs are expressed by a machine usage criterion [5]. Moreover we developed an efficient method which detects and fills existing gaps in the schedule with suitable jobs. It allows us to increase both the QoS and machine usage by limiting fragmentation of the processor time.

The feasibility of the solutions we propose has been evaluated by comparing with a FCFS, an EASY backfilling, and a Flexible backfilling algorithms. The evaluation was conducted by simulations with the Alea simulator [12] using different streams of synthetically generated jobs.

2. Problem Description

In our study we consider a continuous stream of sequential or parallel batch jobs, which arrive to the system and are placed into a single job queue (FCFS, Easy and Flexible backfilling) or into the schedule (EG-EDF, Tabu search). Each job J is characterized by a submission time $Submit_J$, which represents the time when the job arrives, a deadline $Deadline_J$, the number Req_J of CPUs requested for its execution, an estimation of its duration $Estimated_J$, and a benchmark score BM_m, which represents the CPU speed of a machine m used for the time estimation. Precise J execution time for a specific machine $\bar{m}$ is calculated as $(Estimated_J \cdot BM_m)/BM_{\bar{m}}$. All the jobs are considered non-preemptible.

The target architecture is a computational Grid made up of multi-processor machines. Each machine m is characterized by a number R_m of CPUs, and all CPUs within one machine have the same speed BM_m. Different machines may have different speeds and number of CPUs. Machines use the Space Sharing processor allocation policy which allows parallel execution of k jobs on machine m if $R_m \geq \sum_{J=1}^{k} Req_J$.

Various objective functions can be considered such as makespan or average flow time. Our scheduler aims to maximize both the resource utilisation and the number of jobs with the respected deadlines [5]. A higher resource utilisation fulfills resource owner expectations, while a higher number of non delayed jobs guarantees a higher QoS provided to the users.

3. Applied Approaches

In this section we describe two different approaches we propose to solve the considered job scheduling problem. First principles of queue-based Flexible backfilling are explained. Next we focus on the schedule-based solutions. They are represented by a dispatching rule, which is used to create an initial schedule and Tabu search algorithm, which optimizes the initial solution according to the objective function.

3.1 Flexible Backfilling

Flexible backfilling [20] is a variant of the EASY backfilling algorithm which is an extension of the original backfilling algorithm [14]. In the Flexible backfilling, a priority value $P(J)$ is computed for each job J by exploiting a set of heuristics. Each heuristics follows a different strategy to satisfy both users and system administrator requirements.

After selection of the set of machines suitable to perform a job, the priority value assigned to such job is the sum of the values computed by each heuristics. In our study, to select the set of machines we considered only the number of available processors on a machine. Priority values are re-computed at scheduling events, which are job submission and completion. We defined the following heuristics: *Aging*, *Deadline*, and *Wait Minimization*.

Aging aims 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 the job J is increased as follows: $P(J) += agefactor \cdot age(J)$, where $age(J)$ equals to $wallclock - Submit_J$ and $agefactor$ is a multiplicative factor set by the administrator according to the adopted system management policies. The value of the system wall-clock is represented by $wallclock$ parameter equal to the time when the heuristic is computed.

Deadline aims to maximize the number of jobs that terminate their execution within their deadline. It requires an estimation of the job execution time in order to evaluate its completion time with respect to the current wall-clock time. The heuristic assigns a minimal value (Min) 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 (Max), the score assigned to the job is increased in inverse proportion with respect to such distance. The threshold value may be tuned according to the importance assigned to this heuristics. Without loss of generality, in our work, when a job goes over its deadline before it is scheduled, its updating priority value is set to Min. Each job is scheduled on the first most powerful available machine. Since jobs with a closer deadline receive higher priority, this strategy should improve the number of jobs executed within their deadline. Let $Estimated_J$ to be the estimated execution time of job J, we define:

$$\begin{aligned} Nxtime_J &= Estimated_J \cdot \frac{BM_{\bar{m}}}{BM_m} \\ Extime_J &= Now + Nxtime_J \\ t_J &= Deadline_J - k \cdot Nxtime_J \end{aligned}$$

where BM_m is the most powerful cluster machine m (optimistic prediction), and $BM_{\bar{m}}$ is the power of the machine $\bar{m}$ utilised to estimate the execution time of J. $Nxtime_J$ denotes the job's estimated execution time and $Extime_J$ denotes the estimated termination time of the job with respect to the current wall-clock (Now). t_J is the time from which the job must be evaluated to meet its deadline (i.e., the job priority is updated to consider its deadline too). k is a constant value fixed by the installation, which permits us to overestimate $Nxtime_J$.

The value $P(J)$ is increased by the Deadline heuristics according to the following formula:

$$P(J)+= \begin{cases} Min & \text{if } Extime_J \leq t_J \\ a(Extime_J - t_J) + Min & \text{if } t_J < Extime_J \leq Deadline_J \\ Min & \text{if } Extime_J > Deadline_J \end{cases}$$

where a is the angular coefficient of the straight line passing through the points (t_J, Min) and $(Deadline_J, Max)$.

Finally, *Wait Minimization* 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 *boostvalue* be the factor set by administrator according to system management policies and $minext = \min(Estimated_J)$. The value of $P(J)$ is increased by the heuristics as follows:

$$P(J)+= \frac{boostvalue \cdot minext_J}{Estimated_J}$$

In this paper the parameters used in Flexible backfilling were hand tuned to following values: $agefactor = 0.01$, $k = 2.0$, $max = 20.0$, $min = 0.1$ and $boostvalue = 2.0$. At each scheduling event the value of $P(J)$ for all queued jobs is reset to zero and then these heuristics are applied for each job to compute new $P(J)$ values so that $P(J) = Aging + Deadline + Wait\ Minimization$. Then the queue is sorted according to new $P(J)$ values and the backfilling procedure starts.

3.2 Earliest Gap—Earliest Deadline First Rule

In this section the proposed schedule-based approach *Earliest Gap—Earliest Deadline First (EG-EDF)* dispatching rule is described. It places a new submitted job into the *existing schedule* to built the schedule incrementally. It permits us to compute a new job scheduling plan saving running time for scheduling since the new plan is not re-computed from scratch. To do this, it is necessary to choose a good place in the schedule for the job being scheduled, otherwise resource utilisation may drop quickly due to the *gaps* appearing in the schedule. A gap is considered to be a period of idle CPU time. A new gap appears in the schedule every time the number of currently available CPUs by the machine is lower than the number of CPUs requested by a job. In such situation job has to be placed in the schedule to a time when a sufficient number of CPUs is available. Gaps can also appear when there are more CPUs than required by the jobs. They generally lead to processor fragmentation which results in a bad system utilisation.

In order to reduce the processor fragmentation, we developed a method that is able to optimize the schedule by placing the jobs into existing gaps. It is a key part of EG-EDF rule which works in the following way. Suppose a new job J arrives to the system. Using the existing schedule the *Earliest Gap (EG)* suitable for J is identified for each machine. Let S denotes the number of found EGs ($S \leq \#\ of\ Machines$). We consider three different cases: $S \geq 2$, $S = 1$, and $S = 0$. $S \geq 2$ means there is more than one EG for the job assignment. A *weight* is computed for each assignment of J to EG according to Equation 1, and the EG with the highest weight is chosen. The weight function is defined as:

$$\begin{aligned} weight &= weight_{makespan} + weight_{deadline} \\ weight_{makespan} &= \frac{makespan_{old} - makespan_{new}}{makespan_{old}} \\ weight_{deadline} &= \frac{nondelayed_{new} - nondelayed_{old}}{nondelayed_{old}} \end{aligned} \tag{1}$$

Here the $makespan_{old}$ is the expected makespan[1] of the current schedule, $makespan_{new}$ is the makespan of the new schedule. $nondelayed_{old}$ and $nondelayed_{new}$ are the number of jobs executed within their deadline before and after the job assignment, respectively.

$S = 1$ simply means there is just one EG for the job J and this is used for the job assignment. $S = 0$ means there are no suitable gaps. In such case the job is placed into each machine's schedule according to the Earliest Deadline First (EDF) strategy. Each of these assignments is evaluated separately and the one with the highest weight is accepted.

3.3 Tabu Search

Although EG-EDF rule is trying to increase the machine usage and also to meet the job deadlines by finding suitable gaps, it only manipulate with the newly arrived job. The previously scheduled jobs are not considered by EG-EDF rule when building a new schedule. In such case many gaps in the schedule may remain. To reduce their effect, we propose a Tabu search [9] optimization algorithm which increases both machine usage and the number of jobs executed respecting their deadline. It only works with jobs prepared for running—jobs already running are not affected since the job preemption is not supported.

Tabu search selects the last job from the schedule of a certain machine, which has the highest number of delayed jobs. Such job must not be in the *tabu list* to prevent cycling. The tabu list contains jobs that were selected in previous iterations. It has limited size and the oldest item is always removed when the list becomes full. Selected job is put into the tabu list and then the method for finding the *Earliest Gap (EG)* for this job in a specific machine's schedule is executed. If suitable EG is found the job is moved to it and the weight value is computed according the Equation 1. If $weight > 0$ the move is accepted since it improves the quality of current schedule, $makespan_{old}$ and $nondelayed_{old}$ values are updated, and a new iteration is started. Otherwise, the move is not accepted, the job is not moved, and next machine's schedule is used to find an EG for this job. If none of the remaining machines has a suitable gap in its schedule, a new iteration is started by selecting a different job, since the previous choice is now banned by the tabu list. It can happen that the machine with the highest number of delayed jobs contains only jobs present in the tabu list. Then the machine with the second highest number of delayed jobs is selected. The process continues until there are no delayed non-tabu jobs or the upper bound of iterations is reached.

[1]Makespan is the completion time of the last job in the schedule.

4. Experimental Evaluation

In order to verify the feasibility of the EG-EDF and Tabu search solutions, some experiments have been conducted. The evaluation was performed by comparing our solutions with FCFS, EASY backfilling (Easy BF), and Flexible backfilling (Flex. BF). Concerning the Flex. BF, job priorities are updated at each job submission or ending event and the reservation for the first queued job is maintained through events. We used our Alea Simulator [12], which is an extended version of the GridSim toolkit. The evaluation was conducted by simulations using five different streams of jobs synthetically generated according to a negative exponential distribution with different inter-arrival times between jobs [5, 20]. According to the job inter-arrival times a different workload is generated through a simulation. Smaller this time is, greater the system workload is. Inter-arrival times were chosen in a way that the available computational power is able to avoid the job queue increasing when it is fixed equal to 5. Moreover, each job and machine parameter was randomly generated according to a uniform distribution[2].

The experimental tests were conducted by using a Grid made up of 150 machines with different CPU number and speed, and 3000 jobs. Job scheduling plans were carried out by exploiting the Space Sharing processor allocation policy, and both parallel and sequential jobs were simulated—up to now parallel jobs are always executed on only one machine with a sufficient number of CPUs. In order to obtain stable values, each simulation was repeated 20 times with different job attributes values. The experiments were conducted on an Intel Pentium 4 2.6 GHz machine with 512 MB RAM.

To evaluate the quality of schedules computed by EG-EDF rule and Tabu search, we exploited different criteria: the percentage of jobs executed do not respecting their deadline, the percentage of system usage, the average job slowdown, and the average algorithm runtime.

The system usage was computed at each simulation time by using the following expression:

$$System\ usage = \frac{\#\ of\ active\ CPUs}{min(\#\ of\ available\ CPUs, \#\ of\ CPUs\ requested\ by\ jobs)}$$

It permits us to not consider situations when there are not enough jobs in the system to use all the available machines. It happens at the beginning and at the end of the simulation.

[2]Following ranges were used: Job execution time [500–3000], jobs with deadlines 70%, number of CPUs required by job [1–8], number of CPU per machine [1–16], machine speed [200–600]

4.1 Discussion

In Figure 1 (left) the percentage of jobs executed not respecting their deadline is shown. As expected, when the job inter-arrival time increases, the number of late jobs decreases. Moreover, it can be seen that both EG-EDF rule and Tabu search produced much better solutions than Flexible backfilling, Easy backfilling, and FCFS. Tabu search outperforms all the other algorithms. In particular, it obtains nearly the same results of EG-EDF rule when the system contention is low (job inter-arrival time equal to 5). In Figure 1 (right), the percentage of system usage is shown. Schedule-based algorithms are, in general, able to better exploit the system computational resources. However, when there is not contention in the system the solutions we propose obtained worse results than the other ones. When the available computational power is able to avoid the job queue increasing, the Tabu search and EG-EDF solutions do not improve, or improve very little, the previous schedule. In this situation, the schedule-based approach is less effective concerning the resource utilisation. In such situation the schedule is almost empty so a newly arrived job is often immediately executed on an available machine, therefore the Tabu search has a very limited space for optimization moves.

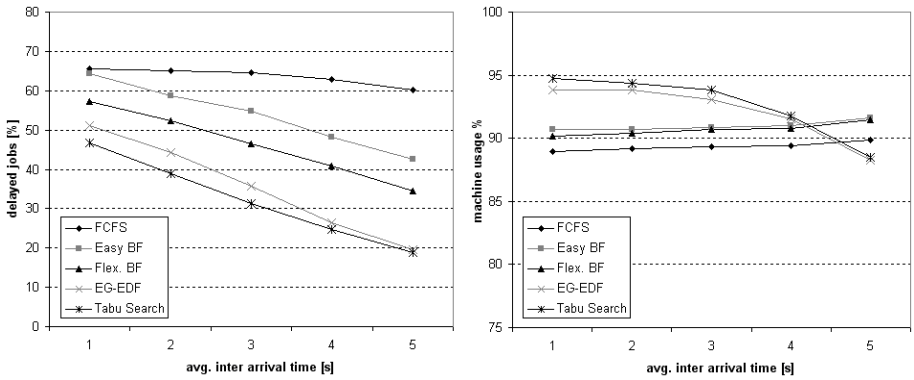

Figure 1. Average percentage of delayed jobs (left) and system usage (right).

Figure 2 (left) shows the average scheduling times spent by the scheduler for conducting the tests on the simulated computational environment. It is computed by measuring the scheduling time at each scheduling event. The runtime of FCFS is very low w.r.t. to Easy and Flexible backfilling for which it grows quickly as a function of the job queue length. Although the Flexible backfilling has to re-compute job priorities at each scheduling event, and then has to sort the queue accordingly, it

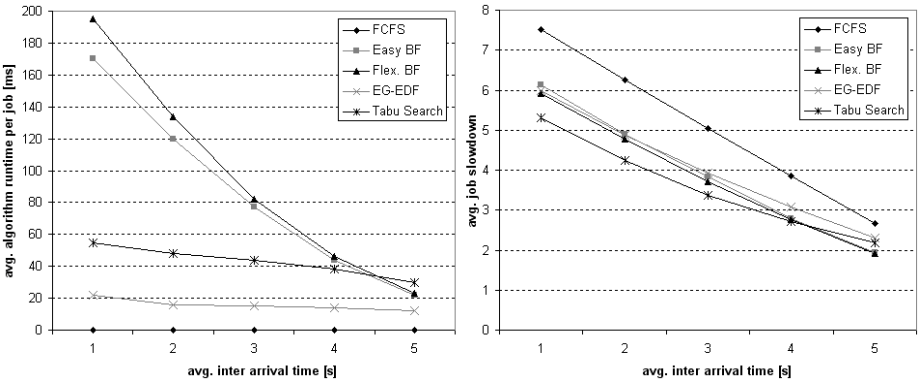

Figure 2. Average algorithm runtime (left) and the average job slowdown (right).

causes minimal growth of its run time compared to the Easy backfilling. This is due to the application of an efficient sort algorithm.

Local search based algorithms are often considered to be very time consuming. Our implementation, which exploits an incremental approach based on the previously computed schedule, is able to guarantee a shorter and stable execution time w.r.t. the other algorithms. In particular, EG-EDF rule is fast and it always generates acceptable schedule, so we can stop Tabu search optimization at any time if prompt decisions are required.

Figure 2 (right) shows the average job slowdown. It is computed as $(Tw + Te)/Te$, with Tw the time that a job spends waiting to start its execution, and Te the job execution time [15]. This shows us how the system load delays the job execution. As expected, greater the system contention is, greater the job slowdown is. In this case the better results are obtained by Tabu search, which are enough close to those obtained by the Flexible backfilling algorithm.

5. Conclusion and Future Work

Both Flexible backfilling and schedule-based algorithms demonstrated significant improvement when decreasing the number of late jobs while keeping the machine usage high. This would not be possible without the application of effective gap-filling method in case of the schedule-based algorithms. Tabu search algorithm proved to be more successful in decreasing the number of delayed jobs over Flexible backfilling—on the other hand precise job execution time was known in this case so the advantage of schedule-based solution took effect. The incremental approach used in the schedule-based solutions allowed to keep the al-

gorithm runtime stable and low. From this point of view both Easy and Flexible backfilling are more time consuming since their runtime is growing with the size of the queue more quickly.

In the future we would like to include job execution time estimations to study their effect on the performance of schedule-based methods. Usually this is not a crucial issue for the queue-based algorithms because they are designed to deal with dynamic changes. However, the schedule-based approach relies on the precision of execution time prediction much more. We expect that some changes will have to be done when the estimates will not meet the real job execution time. It is probable that in such situation local change or limited rescheduling will be necessary. Also, we would like to introduce failure tolerance and investigate job preemption and job migration effects. Next we plan to compare these solutions with other scheduling techniques such as Convergent Scheduling [5].

Acknowledgments

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419, by the Grant Agency of the Czech Republic with grant No. 201/07/0205, and by the EU CoreGRID NoE (FP6-004265) which we highly appreciate.

References

[1] A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational Grids. In *The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)*, pages 45–52, 2000.

[2] V. A. Armentano and D. S. Yamashita. Tabu search for scheduling on identical parallel machines to minimize mean tardiness. *Journal of Intelligent Manufacturing*, 11:453–460, 2000.

[3] R. Baraglia, R. Ferrini, and P. Ritrovato. A static mapping heuristics to map parallel applications to heterogeneous computing systems: Research articles. *Concurrency and Computation: Practice and Experience*, 17(13):1579–1605, 2005.

[4] J. Bidot. *A General Framework Integrating Techniques for Scheduling under Uncertainty.* PhD thesis, Institut National Polytechnique de Toulouse, 2005.

[5] G. Capannini, R. Baraglia, D. Puppin, L. Ricci, and M. Pasquali. A job scheduling framework for large computing farms. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'07)*, 2007.

[6] Cluster Resources. *Moab Workload Manager Administrator's Guide*, 2008.

[7] J. P. Jones. *PBS Professional 7, Administrator Guide.* Altair, April 2005.

[8] F. W. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer, 2003.

[9] F. W. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998.

[10] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC Resource Management Systems: Queuing vs. Planning. In *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *LNCS*, pages 1–20. Springer, 2003.

[11] E. Huedo, R. Montero, and I. Llorente. The GridWay framework for adaptive scheduling and execution on Grids. *Scalable Computing: Practice and Experience*, 6(3):1–8, 2005.

[12] D. Klusáček, L. Matyska, and H. Rudová. Alea – Grid Scheduling Simulation Environment. In *Workshop on scheduling for parallel computing at the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007)*, LNCS. Springer, 2008. To appear.

[13] D. Klusáček, L. Matyska, H. Rudová, R. Baraglia, and G. Capannini. Local Search for Grid Scheduling. In *Doctoral Consortium at the International Conference on Automated Planning and Scheduling (ICAPS'07)*, USA, 2007.

[14] D. A. Lifka. The ANL/IBM SP Scheduling System. In *Job Scheduling Strategies for Parallel Processing*, volume 949 of *LNCS*, pages 295–303. Springer, 1995.

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

[16] M. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, 2005.

[17] K. Stucky, W. Jakob, A. Quinte, and W. Süß. Solving scheduling problems in Grid resource management using an evolutionary algorithm. In R. Meersman and Z. Tari, editors, *OTM Conferences (2)*, volume 4276 of *LNCS*, pages 1252–1262. Springer, 2006.

[18] R. Subrata, A. Y. Zomaya, and B. Landfeldt. Artificial life techniques for load balancing in computational Grids. *Journal of Computer and System Sciences*, 73(8):1176–1190, 2007.

[19] W. Süß, W. Jakob, A. Quinte, and K. Stucky. GORBA: A global optimising resource broker embedded in a Grid resource management system. In S. Zheng, editor, *International Conference on Parallel and Distributed Computing Systems, PDCS 2005*, pages 19–24. IASTED/ACTA Press, 2005.

[20] A. D. Techiouba, G. Capannini, R. Baraglia, D. Puppin, and M. Pasquali. Backfilling strategies for scheduling streams of jobs on computational farms. In *CoreGRID Workshop on Grid Programming Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments*. Springer, 2008. To appear.

[21] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[22] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for scheduling distributed data-oriented applications on global Grids. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 75–80. ACM Press, 2004.

[23] M. Q. Xu. Effective metacomputing using LSF multicluster. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 100–105. IEEE Computer Society, 2001.