On the Comparison of CPLEX-Computed Job Schedules with the Self-Tuning dynP Job Scheduler

Sven Grothklags[1] and Achim Streit[2]

[1] Faculty of Computer Science, Electrical Engineering and Mathematics,
Institute of Computer Science, Paderborn University, Germany
[2] Paderborn Center for Parallel Computing, Paderborn University, Germany
E-mail: sven,streit@upb.de

Abstract

In this paper we present a comparison of CPLEX-computed job schedules with the self-tuning dynP scheduler. This scheduler switches the active scheduling policy dynamically during run time, in order to re¤ect changing characteristics of waiting jobs. Each time the self-tuning dynP scheduler checks for a new policy a quasi off-line scheduling is done as the number of jobs are £xed. Two questions arise from this fact: what is the optimal schedule in each self-tuning step? and what is the performance difference between the optimal schedule and the best schedule generated with one of the scheduling policies?

For that we modelled the scheduling problem as an integer problem, which is then solved with the well-known CPLEX library. Due to the size of the problem, we apply time-scaling, i. e. the schedule is computed on a larger than one second precise scale. We use the CTC job trace as input for a discrete event simulation and evaluate the performance difference between the CPLEX-computed schedules and the schedules generated by the self-tuning dynP scheduler. The results show, that the performance of the self-tuning dynP scheduler is close to solutions computed by CPLEX. However, the self-tuning dynP scheduler needs much less time for generating the schedules than CPLEX.

1 Introduction

Modern high performance computing (HPC) machines are becoming increasingly faster in compute and interconnect speeds, memory bandwidth, and local £le I/O. An ef£cient usage of the machine is important for the users and owners, as HPC systems are rare and high in cost. Resource management systems (RMS) for modern HPC machines consist of many components which are all vital in keeping the machine fully operational. With regards to performance aspects all components of a modern RMS should perform their assigned tasks ef£ciently and fast, so that no additional overhead is induced. However, if resource utilization and job response time are addressed, the scheduler plays a major role. An ef£cient sorting and scheduling of submitted jobs and resource management is essential for a high utilization and short response times.

Due to being rare HPC machines usually have a large user community with different resource requirements and general job characteristics [2]. For example, some users primarily submit parallel and long running jobs, while others submit hundreds of short and sequential jobs. Furthermore, the arrival patterns vary between speci£c user groups. Hundreds of jobs for a parameter study might be submitted in one go via a script. Other users might only submit their massively parallel jobs one after the other.

This results in a non-uniform workload and job characteristics that permanently change. The job scheduling policy applied in a RMS is chosen in order to achieve a good overall performance for the expected workload. Most commonly used is £rst come £rst serve (FCFS) combined with back£lling [8, 12, 9], as on average a good performance for the utilization and response time is achieved. However, with certain job characteristics other scheduling policies might be superior to FCFS. For example, for mostly long running jobs, longest job £rst (LJF) is bene£cial, while shortest job £rst (SJF) is used with mostly short jobs. Hence, a single policy is not enough for an ef£cient resource management of HPC systems. Many modern RMSs have several scheduling policies implemented, or it is even possible to replace the complete scheduling component. We call the feature of dynamically switching the scheduling policy during the runtime of the scheduler, "dynP" [13].

It has to be decided when the scheduling policy is switched and which policy is to be used instead. This decision is typically dif£cult to make. Many aspects have to be considered, e. g. what are the characteristics of future jobs?, various scheduling policies have to be tested, and additional

Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

scheduler parameters need a precise setting. Besides, all this work has to be done on a frequent basis in order to re¤ect the changing job characteristics. A common approach is to conduct an experimental search on the basis of past and present workloads. This process is long lasting. Hence, a self-tuning scheduler is needed, which searches for the best setting on its own. Similar work on schedulers with dynamic policy switching is presented in [11] and in [1] a self-tuning system is presented which uses genetic algorithms to generate new parameter settings for a scheduling system.

The remainder of this paper is structured as follows. In Section 2 the basic concept of the self-tuning dynP scheduler is presented. In Section 3 the scheduling problem is modelled as an integer problem. For solving the problem with the CPLEX library, time-scaling had to be applied. This is presented at the end of Section 3. Finally in Section 4 results of the comparison between CPLEX-computed schedules and basic scheduling policies for the CTC trace are presented. The paper ends with a brief conclusion.

2 Concept of Self-Tuning dynP

At the PC^2 (Paderborn Center for Parallel Computing) the self-developed resource management system CCS (Computing Center Software, [7]) is used for managing the *hpcLine* cluster [5] and the *pling* Itanium2 cluster [10]. Three scheduling policies are currently implemented: FCFS, SJF, and LJF. According to the classi£cation in [4], CCS is a planning based resource management system. Planning based RMS schedule the present and future resource usage, so that newly submitted jobs are placed in the active schedule as soon as possible and they get a start time assigned. With this approach back£lling is done implicitly.

By planning the future resource usage, a sophisticated approach is possible for £nding a new policy. For all waiting jobs the scheduler computes a full schedule, which contains planned start times for every waiting job in the system. With this information it is possible to measure the schedule by means of a performance metrics (e. g. response time, slowdown, or utilization). The concept of self-tuning dynP is:

> The self-tuning dynP scheduler computes full schedules for each available policy (here: FCFS, SJF, and LJF). These schedules are evaluated by means of a performance metrics. Thereby, the performance of each policy is expressed by a single value. These values are compared and a decider mechanism chooses the best policy, i. e. the lowest (average response time, slowdown) or highest (utilization) value.

We call these previously described actions a *self-tuning step*. For the required decision which policy to switch to, several levels of sophistication are thinkable. In [15] we presented the *simple decider* that basically consists of three if-then-else constructs. It chooses that policy which generates the minimum value. However, the simple decider also has drawbacks, as it does not consider the old policy. Especially if two policies are equal and a decision between them is needed, information about the old policy is helpful. A detailed analysis of the simple decider showed, that in four cases even a wrong decision is made by the simple decider [14]. FCFS is favored in three and SJF in one case, although staying with the old policy is the correct decision with these cases. This is implemented in the advanced decider.

3 Optimal Schedules with CPLEX

The self-tuning dynP scheduler generates full schedules, which contain a start and end time for every submitted job. By that, it is possible to compute waiting and response times for each job. With a performance metrics the schedules are analyzed and the self-tuning dynP scheduler chooses the policy that generates the best schedule and switches to it. In each self-tuning step a quasi off-line scheduling is done as the number of jobs are £xed. However, it is not a classic off-line scheduling by optimizing the makespan. And the schedule does not start with an empty machine, i. e. some resources are not available. The history of resource usage has to be considered as a result of jobs started in the past.

From this two questions arise:

1. What is the optimal schedule with respect to the quasi off-line scheduling problem in each self-tuning step?
2. What is the performance difference between the optimal schedule and the best schedule generated with one of the scheduling policies?

The second question is interesting, as it gives answers to how much performance is lost when a common scheduling policy like FCFS (+ back£lling) is used.

An approach to compute optimal schedules is to model the scheduling problem as an integer problem, which is then solved with the well-known ILOG CPLEX library [6].

3.1 Modelling the Scheduling Problem

Following [17], we model the scheduling problem as an integer problem:

Three values are used to describe the properties of a job i. The number of requested resources is denoted with w_i (width). The estimated duration is described by d_i and the job is submitted at time s_i. Note, we are using the estimated duration of jobs, as we assume planning based resource management system, which require this information.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

The scheduler of the resource management system knows only the estimated duration at scheduling time and uses it for the planning of the schedule.

The history of resource usage is a list of tuples. A tuple consists of a time stamp and the number of resources that are free from that time on. Figure 1 shows an example. The number of free resources are increasing monotonously as only already running jobs are considered. And if more than one job ends at the same time, a single time stamp is suf£cient. Note, the estimated duration of already running jobs has to be used for generating the time stamps.

The variables are de£ned as:

$$x_{it} = \begin{cases} 1\,, & \text{if job i is started at time t} \\ 0\,, & \text{else} \end{cases} \quad (1)$$

We use the average response time weighted by width (ARTwW) as the objective function and it is de£ned as:

$$\text{Minimize} \qquad \sum_{i,t} x_{it}\ (t - s_i + d_i)\ w_i \quad (2)$$

The constraints are:

$$\sum_{t} x_{it}\ = 1 \qquad \forall i \quad (3)$$

$$\sum_{i,\max(0,t-d_i+1)\le j\le t} x_{ij}\ w_i\ \le M_t \qquad \forall t \in [0,T] \quad (4)$$

$$x_{it} \in 0,1 \quad (5)$$

Constraint 3 describes that every job is started exactly once. In constraint 4 T is the maximum possible length of the schedule. Usually this is the sum of all job durations ($\sum d_i$), but the resulting integer problem would contain too many variables. Assuming that the schedules for FCFS, SJF, and LJF are already computed, the best solution is to use the maximum makespan of the three schedules. This is most likely the makespan of the LJF-generated schedule. The sum in constraint 4 describes the fact that the machine consists of M_t resources in total. In order to re¤ect the machine history the number of available resources has to be reduced accordingly (see Figure 1).

3.2 Time-Scaling

The smallest time step in resource management systems is usually one second. This requires t in Equation 1 to be at a second scale. However, this induces too many variables (number of jobs times T in seconds) and therefore too much memory (roughly the number of variables times T). For example: the maximum makespan is two days (172,800 seconds) and a schedule for eight jobs has to be computed. The number of variables is already more than a million, although the scheduling problem is rather small for a real world scenario. Such problems would need a considerable amount of memory (and time). If the problems grow in size, the 8 GB of the available simulation hardware (SUN 8-way Ultra-Sparc Server) are not enough. A commonly used solution to solve this, is to use time-scaling [3]. This means, that the schedule is computed at a greater time scale (e. g. one minute). By that a certain amount of time in the real schedule is reduced to a single point of time. For most real world scenarios this is enough, as jobs are usually estimated to run for several minutes, hours, or even days, so signi£cantly longer than one second.

We use the following approximation for computing a suitable time-scale: the size of the integer problem (i. e. the number of matrix entries and constraints) roughly depends on the number of jobs multiplied with the square of T. The variable matrix is sparse and the degree of sparseness depends on the accumulated run time of all jobs. Hence, the size of the integer problem in memory (i. e. the total amount of memory that should be used) is computed by:

$$\text{number of jobs} \cdot \left(\frac{\text{max. makespan}}{\text{time-scale}}\right)^2 \cdot \frac{\text{acc. run time}}{\text{max. makespan} \cdot \text{number of jobs}} \cdot x$$

x denotes the memory size of each matrix entry in bytes. In initial testings we discovered that good values for x are 0.1 kB or 0.0001 MB.

Then:

$$\text{time-scale} = \sqrt{\frac{\text{max. makespan} \cdot \text{acc. run time} \cdot x}{\text{available memory}}} \quad (6)$$

The time-scale is rounded up to the next 60 seconds, so that the schedules are solved in a full minute scale. Additionally, the amount of memory used for the integer problem should be about four times smaller than the total memory available, as the additional memory is needed by CPLEX during the solving phase. For the computations a machine with 8 GB of total main memory is used.

Time-scaling also has drawbacks. Jobs are scheduled at the beginning of a time-scaled interval, e. g. at the beginning of a one minute interval in the schedule. The duration of jobs is not time-scaled and they still end at any time in the time-scaled interval. Hence, from the time a job ends to the next interval start resources remain unused, although they could be utilized without time-scaling. To solve this problem each job is moved forward as much as possible after the starting order of the jobs is found. With that, unused slots in the schedule are avoided. The maximum time a job is moved forward is $(\text{time-scale} - 1)$ seconds and on average $\frac{\text{time-scale}-1}{2}$ seconds. To implement this in practice each job is inserted in the schedule according to the starting order of the schedule computed by CPLEX. Each job

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

time [sec.]	free resources
100	16
120	48
140	64
160	80
200	96

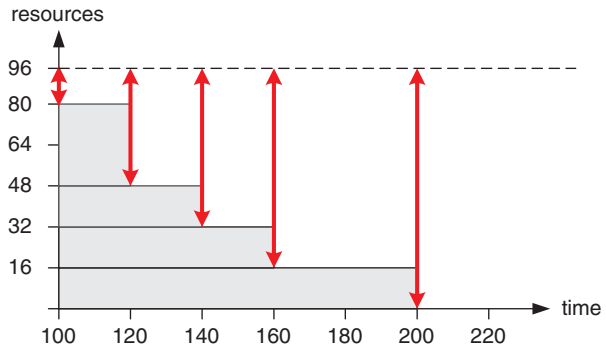

Figure 1. Example for a machine history.

is placed as soon as possible and unused time slots, due to time-scaling, do no longer occur. By applying time-scaling the £nal schedule might not be optimal, as heuristics (time-scaling) are used. However, if time-scaling is not applied, the problem might not £t in the available memory and no solution is computed.

The CPLEX-computed schedule is then analyzed and the results are compared to the performance of the three basic policies FCFS, SJF, and LJF. We de£ne the quality of a policy p and according to a performance metrics m as:

$$quality(p,m) = \frac{performance(CPLEX,m)}{performance(p,m)} \qquad (7)$$

If $quality(p,m) < 1$ the schedule computed by CPLEX is better. The percentage $(1 - quality(p,m)) \cdot 100$ depicts how much performance is lost by using the policy p. Due to time-scaling the $quality(p,m)$ might be > 1. In this case, the policy p is better than the schedule computed by CPLEX with time-scaling applied.

Despite solving the integer problem, the time CPLEX needs for the computation is of an additional importance. Computing an optimal schedule and using it in a resource management system surely improves the performance of the system. However, if the computation takes too much time (e. g. one hour), it is not practical. In general, the scheduling component of a resource management system should generate new schedules as fast as possible, which typically means close to one second or less in a real world scenario.

The time for scheduling a new job is not critical in queuing based resource management systems as the job is appended to a queue. In contrast, a planning based resource management system re-plans the resource usage and considers the new job [4]. This has to be done fast, as the updated schedule is required for the following requests. For example, a request for a reservation is submitted right after. An answer is expected immediately as other reservation requests might depend on the acceptance of this request. Hence, the updated resource plan has to be computed fast. Therefore, £nding and using optimal schedules in the real world is not practical. With the basic policies of the self-tuning dynP scheduler, the time of scheduling is less than 10 milliseconds for an average number of 25 waiting jobs.

Therefore, a mixture of quality and computational time has to be used in a comparison. In other words the physical de£nition of power, i. e. work per time unit, is well suited for measuring the performance of a scheduler.

4 Results

In the following we would like to answer the questions from the beginning of this section, i. e. How much performance is lost, compared with the CPLEX-computed schedule, by using common scheduling policies like FCFS or SJF? For this we use the self-tuning dynP scheduler. Initially, we planned that simulations are performed with several job sets. However, as solving the integer problem with the CPLEX library needs a lot of computational time, we use only one job set in the end, which is the CTC trace from the Parallel Workload Archive [16].

Schedules are computed by CPLEX in each self-tuning step, hence at every job submission. Although these schedules are available, they are not used for the actual scheduling process. They are only used for the comparison with the schedule of the best basic policy in each step of the self-tuning process. Hence, it is possible to directly state how much performance is lost in each step of the self-tuning dynP scheduler. If optimal schedules were to be used directly for scheduling the jobs, future scheduling decisions and optimal schedules would be in¤uenced by a different past resource usage and a fair comparison would not be possible.

As previously stated the CPLEX approach with solving an integer problem needs a lot of memory. This is due to the de£nition of the variable $x_{i,t}$ and the constraints.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

Hence, time-scaling is used to reduce the number of variables, which implies that schedules are only solved on a one minute or greater scale.

Table 1 shows exemplary CPLEX runs for various problem sizes, i. e. schedules. For each line in the table the time at which the integer problem was solved is given, i. e. a new job was submitted and self-tuning was invoked. The three columns show the values used to compute the time scaling according to Equation 6. The resulting time scale used for solving the integer problem is given in the following column. Finally, the last columns depict the results and performance of computing a schedule with CPLEX. We measure a schedule with the average slowdown weighted by job area (SLDwA) metrics. As stated above, if the performance loss is positive, the CPLEX computed schedule is better than the schedule generated by the scheduling policy. Of course this should be the normal case. However, sometimes the performance loss is negative. This indicates, that the scheduling generated by the best scheduling policy is better than the solution found by CPLEX. Obviously this is caused by the time scaling in the integer problem. If no time-scaling is applied and enough memory and compute time is available, the CPLEX should always at least £nd the same schedule as any scheduling policy and most likely a better one.

It is noted that the problem sizes in the £rst block are considerably large. This is indicated by the amount of jobs, the accumulated run time, and £nally the time scale. The performance loss of the scheduling policy (in all four cases SJF + back£lling) is very small and in the 1% range. CPLEX needs much compute time for achieving this result.

The second block of examples show, that it is impossible to predict the compute time of CPLEX from previous runs. In these two successive job submissions both scheduling problems are of an equal size. In the £rst case, CPLEX needs 2.5 hours to £nd a solution which is 1.3% better than the best scheduling policy. With the submission of the next job, the scheduling problem increases only slightly in size. The performance loss, i. e. the difference between the scheduling policy and the CPLEX-computed solution is much smaller than before. However, in this case almost 20 times more compute time is needed. This might be the results of the newly submitted job, which might have increased the degree of dif£culty substantially without changing the size of the problem.

In the third block, examples for extremely long compute times of CPLEX are given. Note, 237 hours equals approximately 10 days. However, the CPLEX-computed solutions are clearly better than the schedules of the best basic policies. A time scaling of 6 minutes is used, so that an even larger improvement might be possible, if a second precise scaling is applied. Of course this would require a considerable amount of main memory.

In all previous examples the performance loss of the scheduling policy is within the 1% range. However, the case might be that the scheduling policy is signi£cantly worse. As shown, the largest measured performance loss is close to 11%. About 3.5 hours are needed by CPLEX to compute this solution. The other two examples show, that it is also possible that the CPLEX-computed schedule is worse than the SJF-generated schedule. Obviously this is due to the time scaling in the integer problem. However, the long compute time (almost 15 hours) of CPLEX has to be considered.

Finally, the last row in Table 1 shows the averages of all CPLEX computations. It is seen that the performance loss of the scheduling policies is only 0.7% compared to the CPLEX-computed solution with a 5 minute average time scaling applied. On average more than 5 hours are needed to solve the integer problems. The average size of the according scheduling problem is 22 jobs with a maximal makespan of close to 2 days as an upper bound. In real world scenarios such schedule would be considered to be small. Many more jobs are usually processed in a resource management system, with a much larger makespan. This indicates, that CPLEX-computed schedules are unpractical for a real implementation in a resource management system. The response times of real schedulers need to be considerably small, so that user decisions (e. g. accepting or declining a reservation) can be made quickly. Approaches are thinkable, where the scheduling policy is used to generate an initial schedule and CPLEX is used to £nd better schedules while the initial schedule is active and implemented. However, in online scheduling systems jobs are submitted continuously and with short average interarrival times (on average 369 seconds for the CTC trace). Hence, a new schedule is required, while CPLEX is still solving the previous scheduling problem.

5 Conclusion

In this paper we presented a comparison of the performance of the self-tuning dynP scheduler and CPLEX-computed schedules. The self-tuning dynP scheduler is designed for modern resource management systems and switches the active scheduling policy dynamically during run time, so that changing characteristics of incoming jobs are re¤ected. In a so called self-tuning step, where the scheduler checks for a new policy, schedules for every available policy are computed and analyzed with means of a performance metrics. Then the scheduler switches to the best policy. In this step a quasi off-line scheduling is done, as the number of jobs is £xed. Hence, two questions were interesting for use: what is the optimal schedule with respect to the quasi off-line scheduling problem in each self-tuning step? and what is the performance difference between the optimal schedule and the best schedule generated by the self-tuning

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

submission time	CPLEX problem size jobs	makespan [sec.]	acc. run time [sec.]	time scale [min.]	CPLEX result quality	perf. loss	comp. time hr.	min.	sec.
38,589	40	189,559	1,798,837	7	0.9920	0.80%	8	14	17
38,590	40	189,596	1,862,437	8	0.9869	1.31%	1	47	35
40,284	31	190,899	1,395,637	7	0.9934	0.66%	12	58	51
40,493	32	191,509	1,395,937	8	0.9968	0.32%	23	22	48
50,356	18	194,121	1,030,782	6	0.9865	1.35%	2	26	32
50,360	19	194,161	1,095,582	6	0.9974	0.26%	40	52	51
70,628	17	259,141	715,905	6	0.9821	1.79%	76	15	8
71,271	17	256,741	723,405	6	0.9804	1.96%	129	16	25
71,285	17	256,741	728,505	6	0.9804	1.96%	237	7	59
36,037	39	178,232	1,617,937	7	0.8913	10.87%	3	26	4
52,698	19	201,983	906,331	6	1.0014	-0.14%	14	43	40
69,073	9	239,417	519,300	5	1.0022	-0.22%	0	23	40
averages	21.7	166,766	931,168	5	0.9930	0.70%	5	34	29

Table 1. Examples of CPLEX problem sizes, the quality, and the compute time. The last line with average values are generated from all CPLEX computations.

dynP scheduler?

For answering the two questions, we modelled the scheduling problem as an integer problem. We chose the average response time weighted by the job width (ARTwW) as the objective function, which has to be minimized. We used the well-known ILOG CPLEX library for solving the integer problem. In our model the variable x_{it} is one, if job i is started at time t, otherwise zero. As the scheduler can start jobs every second, the number of variables (number of jobs times the length of the schedule) and thereby the size of the problem in memory is large even for small scheduling instances. Hence, we applied time-scaling, so that the schedule is computed by CPLEX at a larger than one second precise scale.

We used the CTC job trace from Dror Feitelsons Parallel Workloads Archive for evaluating the performance differences. Exemplary results for various problem sizes, i. e. schedules, show that the performance of the self-tuning dynP scheduler is close to the performance of CPLEX. In average the difference is below 1%. However, sometimes the schedule generated by the self-tuning dynP scheduler is better than the solution found by CPLEX. Obviously this is caused by the time scaling in the integer problem. If no time-scaling is applied and enough memory and compute power is available, CPLEX should always at least £nd the same schedule as any scheduling policy and most likely a better one.

We observed, that CPLEX needs a lot of compute time for certain problems. For certain scheduling instances we studied compute times of 237 hours, which equals approximately 10 days. Furthermore, it is impossible to predict the compute time from previous runs. We presented examples were a single additional job increased the compute time from 2.5 hours to almost 41 hours, without changing the size of the integer problem signi£cantly. Due to the unpredictable and long compute times of CPLEX, this solution is obviously not practicable for a real implementation in a modern resource management system for high performance computers.

References

[1] D. G. Feitelson and M. Naaman. Self-Tuning Systems. In *IEEE Software 16(2)*, pages 52–60, April/Mai 1999.

[2] D. G. Feitelson and B. Nitzberg. Job Characteristics of a Production Parallel Scienti£c Workload on the NASA Ames iPSC/860. In D. G. Feitelson and L. Rudolph, editor, *Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 337–360. Springer, 1995.

[3] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling To Minimize Average Completion Time: Off-line and On-line Algorithms. In *Proc. of the of the Seventh ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 142–151, 1996.

[4] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC Resource Management Systems: Queuing vs. Planning. In D. G. Feitelson and L. Rudolph, editor, *Proc. of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*. Springer, 2003, to appear.

[5] The *hpcLine* at the Paderborn Center for Parallel Computing (PC^2). `http://www.upb.de/pc2/services/systems/psc/index.html`, October 2003.

[6] ILOG CPLEX. `http://www.ilog.com/products/cplex/`, October 2003.

[7] A. Keller and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. In *Annual Review of Scalable Computing, vol. 3, Singapore University Press*, pages 1–31, 2001.

[8] D. A. Lifka. The ANL/IBM SP Scheduling System. In D. G. Feitelson and L. Rudolph, editor, *Proc. of 1st Workshop on*

Job Scheduling Strategies for Parallel Processing, volume 949 of *Lecture Notes in Computer Science*, pages 295–303. Springer, 1995.

[9] A. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Back£lling. In *IEEE Trans. Parallel & Distributed Systems 12(6)*, pages 529–543. IEEE Computer Society Press, June 2001.

[10] The *pling* Itanium2 Cluster at the Paderborn Center for Parallel Computing (PC^2). `http://www.upb.de/pc2/services/systems/pling/index.html`, October 2003.

[11] F. Ramme and K. Kremer. Scheduling a Metacomputer by an Implicit Voting System. In *3^{rd}Int. IEEE Symposium on High-Performance Distributed Computing*, pages 106–113, 1994.

[12] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY — LoadLeveler API Project. In D. G. Feitelson and L. Rudolph, editor, *Proc. of 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer, 1996.

[13] A. Streit. On Job Scheduling for HPC-Clusters and the dynP Scheduler. In *Proc. of the 8th International Conference on High Performance Computing (HiPC 2001)*, volume 2228 of *Lecture Notes in Computer Science*, pages 58–67. Springer, 2001.

[14] A. Streit. A Self-Tuning Job Scheduler Family with Dynamic Policy Switching. In D. G. Feitelson and L. Rudolph, editor, *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2002.

[15] A. Streit. The Self-Tuning dynP Job-Scheduler. In *Proc. of the 11th International Heterogeneous Computing Workshop (HCW) at IPDPS 2002*, pages 87 (book of abstracts, paper only on CD). IEEE Computer Society Press, 2002.

[16] Parallel Workloads Archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`, October 2003.

[17] J. van den Akker, C. Hurkens, and M. Savelsbergh. Time-Indexed Formulations for Single-Machine Scheduling Problems: Column Generation. *Journal on Computing*, 12(2):111–124, 2000.