Evaluation of an Unfair Decider Mechanism for the Self-Tuning dynP Job Scheduler

Achim Streit
PC^2- Paderborn Center for Parallel Computing
Paderborn University
33102 Paderborn, Germany
Email: streit@upb.de

Abstract

In this paper we present a new decider mechanism for the self-tuning dynP job scheduler for modern resource management systems. This scheduler switches the active scheduling policy dynamically during run time, in order to re¤ect changing characteristics of waiting jobs. The new decider explicitly prefers a single scheduling policy instead of being fair to all available policies. We use discrete event simulations to evaluate the achieved slowdown and utilization and compare the results with the fair decider mechanism and the static basic scheduling policies. The evaluation shows, that the self-tuning dynP scheduler in combination with the preferred decider achieves good results and that it is superior to common static scheduling approaches, which use only a single policy.

1 Introduction

An ef£cient usage of modern high performance computing (HPC) machines 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 times are addressed, the scheduler plays a major role. A clever resource management of submitted jobs is essential for a high utilization and short response times.

Due to being rare, HPC systems usually have a large user community with different resource requirements and general job characteristics [4]. For example, some users primarily submit parallel and long running jobs, whilst 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 used 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 [9, 18, 12], 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, whilst shortest job £rst (SJF) is used with mostly short jobs [1]. Hence, a single policy is not enough for an ef£cient resource management of HPC systems. Many modern RMSs (e. g. PBSpro [15, 14], Loadleveler [10]) 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 run time of the scheduler, "dynP" [19].

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. characteristics of future jobs, various scheduling policies have to be tested, and additional 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 time consuming. Hence, a self-tuning scheduler is needed, which searches for the best setting on its own.

The remainder of this paper is structured as follows. In Section 2 the basic concept of the self-tuning dynP scheduler is presented. The evaluation in Section 3 begins with details on the performance metrics and the job sets. Next,

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

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

the results are presented. The paper ends with a brief conclusion.

2 Related Work

In [17] the problem of scheduling a machine room of MPP-systems is described. Users either submit long running batch jobs or they work interactively (typically only for a short time). To accomplish this on a single MPP-system the resource management system has to switch from batch mode (preferring batch jobs) to interactive mode (preferring interactive jobs) and back. Usually this is done manually by the administrative staff, e. g. at £xed times of the day: interactive mode during working hours, batch mode for the rest of the day and over weekends. In general, the overall job throughput is the main objective of batch processing. As batch jobs typically have a long run time, waiting is not very critical. On the other hand, a user that works interactively counts the £ve minutes until he/she can start working with the requested resources. Other issues like the overall job throughput or the utilization are less important while operating in interactive mode. Which in comparison to batch mode jobs are rather short. The idea [5] is to allow the users to decide in which mode the system should be operating. Hence, the Implicit Voting System (IVS) is introduced, as users should not vote explicitly:

- If most of the waiting jobs are submitted for batch processing, IVS switches to LJF (longest job £rst). As batch jobs are typically long, they receive a higher priority in the scheduling process. Hence, resources are longer bound to jobs, less resource fragmentation is caused and the utilization and throughput of the system is increased.
- If most of the waiting jobs are submitted for interactive access, IVS switches to SJF (shortest job £rst). As interactive jobs are usually short in their run time and short jobs are preferred, the average waiting time is reduced.
- If the system is not saturated, the default scheduling strategy FCFS (£rst come £rst serve) is used. Note, a threshold for de£ning when a system is saturated and when not is de£ned by the administrative staff. For the authors a MPP system becomes saturated, if more than £ve jobs can not be scheduled immediately.

Unfortunately, the idea of IVS was never realized nor implemented and tested in a real environment.

In [3] a similar approach for the NASA Ames iPSC/860 system is described. In the prime time during the day only a fraction of the resources is allocated to the batch partition, while most of the resources are available for interactive access. During non prime time all resources are assigned to the batch partition. The re-partitioning is done manually and at £xed times of the day. The problem is to get the best performance out of the resource management system. Commonly these are highly parameterized and the administrative staff has to perform a lot of trial and error testing in order to £nd a good parameter setting for the current workload. If the workload changes, new parameter settings have to be found. However, they are notoriously overworked and have little or no time for this £ne tuning, so the idea is to automate this process. Much information about the current and past workload is available, which is used to run simulations in the idle loop of the system (or on a dedicated machine). Various parameter settings are simulated and the best setting is chosen. The authors call such a system *self-tuning*, as the system itself searches for optimized parameter settings. To create new parameter settings for the simulations, genetic algorithms are used. New parameter settings are generated by randomly combining several potential combinations from the previous step. Speaking in biological terms: chromosomes are the binary representation of a parameter. A parameter setting is called individual and the according parameter values are concatenated in their binary representation. In this example the £tness function is the average utilization of the system achieved by the according parameter setting. All simulated parameter settings (individuals) in one step represent a generation. Now the chromosomes of the £ttest individuals of a generation are used to produce new individuals for the next generation. New generations are continuously created with the latest system workload as input. The process is started with default values. In a case study for scheduling batch jobs of the NASA Ames iPSC/860 system the authors observed, that with the self-tuning search for parameter settings the overall system utilization is improved from 88% (with the default parameters) to 91%.

In [13] the processor allocation to iterative parallel applications on a shared memory multiprocessor machine is evaluated. In their example some applications show a behavior where assigning more processors results in a degraded performance. Their de£nition of self-tuning is when "an application determines for itself the best number of processors to use". They present three levels of processor allocation, each being more sophisticated:

1. A good processor allocation is determined once at the beginning of the execution.
2. A new processor allocation is determined repeatedly, either in £xed intervals or after dramatic ef£ciency changes.
3. In each parallel phase of an iteration a new processor allocation is computed.

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

They show, that a dynamic selection of processor allocation matches the best static allocation. However, this is done with the potential bene£t of £nding even better solutions that outperform any static allocation.

In [11] on-line heuristics is for the dynamic mapping of a class of independent tasks onto heterogeneous computing systems are introduced. The SA (switching algorithm) heuristic uses two other heuristics (MCT and MET) in a cyclic fashion depending on the load distribution across the machines. MCT (minimum completion time) assigns each task to that machine which results in the task's earliest completion time. Tasks may be assigned to machines, which do not have the minimum execution time. In contrast, MET (minimum execution time) assigns each task to that machine that performs the task in the least amount of execution time. As the machines ready times are not considered by MET, load imbalance across the machines may occur. SA switches between these two heuristics to get a new heuristic with the desirable properties of the two single heuristics.

3 Concept of Self-Tuning dynP

At the PC^2 (Paderborn Center for Parallel Computing) the self-developed resource management system CCS (Computing Center Software, [8]) is used for managing the *hpcLine* cluster [7] and the *pling* Itanium2 cluster [16]. Three scheduling policies are currently implemented: FCFS, SJF, and LJF. According to the classi£cation in [6], CCS is a planning based RMS. A planning based RMS schedules 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.

Such a self-tuning dynP step is done each time the planning based RMS has to compute a new schedule, that is when jobs are submitted and when executed jobs £nish. An option for the self-tuning dynP step only e. g. when new jobs are submitted, but this option is not studied here.

For the required decision, several levels of sophistication are thinkable. In [21] 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 is shown in Table 1. In four cases (1, 6b, 8c, and 10c) a wrong decision is made by the simple decider [20]. FCFS is favored in three and SJF in one case, although staying with the old policy is the correct decision with these cases. The correct decision behavior is implemented in the advanced decider. Furthermore, the fairness among the policies was of major interest. However, it might be interesting to explicitly prefer one of the policies and neglect the others.

The new *preferred decider*, evaluated in this paper, stays with its preferred policy, unless any other policy is clearly better. Whenever any of the other, non-preferred policies are currently used, the preferred policy has to achieve only an equal performance and the preferred decider switches back.

4 Evaluation

In the following, we compare the performance of the advanced and preferred decider. For this purpose we use a discrete event simulation environment and use synthetic, trace based job sets as input.

4.1 Performance Metrics

We focus on the owner centric slowdown, as it comes without a dimension in contrast to e.g. the average response time. In general, the job slowdown is de£ned as $s = \frac{response\ time}{run\ time} = 1 + \frac{wait\ time}{run\ time}$. The bounded slowdown $s^{60} = max(\frac{response\ time}{\max(run\ time, 60)}, 1)$ is de£ned in [2] in order to exclude very short jobs, which might be the result of an error.

In the evaluation we use the slowdown weighted by job area (SLDwA), which uses the jobs area as a weight (area = run time $\cdot$ requested resources). Thereby, it is circumvented that jobs with the same run time, but with different resource requirements, have the same impact on the overall performance. For example, a job that runs for 0.5 seconds and has to wait for 10 minutes, suffers a slowdown of 1201. A job with the same wait time but a length of 20 seconds has a slowdown of only 31. Assume that both jobs request only a single processor, the 0.5 second job has a slowdown

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

case		combinations	simple decider	correct decision
1		FCFS = SJF = LJF	**FCFS**	**old policy**
2		SJF $<$ FCFS, SJF $<$ LJF	SJF	SJF
3		FCFS $<$ SJF, FCFS $<$ LJF	FCFS	FCFS
4		LJF $<$ FCFS, LJF $<$ SJF		
	a	FCFS $<$ SJF	LJF	LJF
	b	FCFS = SJF	LJF	LJF
	c	FCFS $>$ SJF	LJF	LJF
5		FCFS = SJF, LJF $<$ FCFS ($\Leftrightarrow$ LJF $<$ SJF)	LJF	LJF
6		FCFS = SJF, FCFS $<$ LJF ($\Leftrightarrow$ SJF $<$ LJF)		
	a	old policy = FCFS	FCFS (= old policy)	old policy (= FCFS)
	b	old policy = SJF	**FCFS**	**old policy (= SJF)**
	c	old policy = LJF	FCFS	FCFS
7		FCFS = LJF, SJF $<$ FCFS ($\Leftrightarrow$ SJF $<$ LJF)	SJF	SJF
8		FCFS = LJF, FCFS $<$ SJF ($\Leftrightarrow$ LJF $<$ SJF)		
	a	old policy = FCFS	FCFS (= old policy)	old policy (= FCFS)
	b	old policy = SJF	FCFS	FCFS
	c	old policy = LJF	**FCFS**	**old policy (= LJF)**
9		SJF = LJF, FCFS $<$ SJF ($\Leftrightarrow$ FCFS $<$ LJF)	FCFS	FCFS
10		SJF = LJF, SJF $<$ FCFS ($\Leftrightarrow$ LJF $<$ FCFS)		
	a	old policy = FCFS	SJF	SJF
	b	old policy = SJF	SJF (= old policy)	old policy (= SJF)
	c	old policy = LJF	**SJF**	**old policy (= LJF)**

Table 1. Detailed analysis of the simple decider. Decisions printed in bold highlight the differences.

weighted by area of $1201 \cdot 0.5 = 600.5$ and the 20 second job $31 \cdot 20 = 620$. The slowdown weighted by job area for all jobs is de£ned as $SLDwA = (\sum_{i=1}^{m} a_i \cdot s_i)/(\sum_{i=1}^{m} a_i)$, with a_i being the area of job i and s_i the slowdown as above. The average slowdown weighted by job area is equal to the average response time weighted by job width (number of requested resources). If the used jobs are not changed in their width or run time, the equation $SLDwA = ARTwW \cdot \frac{\sum_{i=1}^{m} w_i}{\sum_{i=1}^{m} a_i}$ holds.

4.2 Workload

An evaluation of job scheduling policies requires to have job input. In this work a job is de£ned by the submission time, the number of requested resources (= width), and the estimated run time (= length). As we model a planning based resource management system [6], run time estimates are mandatory. Additionally, for the simulation the actual run time is required.

In this paper, we use four synthetically generated job sets, which are based on traces from the Parallel Workload Archive [22]. We use the CTC, KTH, LANL, and SDSC data (characteristics are found in Table 2). Only these four come with information about run time estimates. In these traces actual run times are also logged, which we use for the simulation. A trace re¤ects the performance of the scheduler and the behavior of the users, which is in¤uenced by the schedulers actions. If such a trace is used again as input for an evaluation of scheduling strategies, the results are most likely not meaningful. An increased workload is more interesting to evaluate.

Three approaches are thinkable for increasing the workload: decreasing the average interarrival time, increasing the run time (both estimated and actual), and multi-submitting the jobs. In our work we arbitrarily use the £rst approach, as it does not change the outlook (i. e. area) of all processed jobs. The submission time of the jobs is given in seconds from the £rst job submit. We multiply every submission time by the *shrinking factor*. With shrinking factors smaller than one, jobs are submitted with shorter inter-

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

trace	number of jobs	requested resources min	requested resources avg.	requested resources max	available resources on machine
CTC	79,302	1	10.72	336	430
KTH	28,490	1	7.66	100	100
LANL	201,387	32	104.95	1,024	1,024
SDSC	67,667	1	10.54	128	128

trace	estimated run time [sec.] min	estimated run time [sec.] avg.	estimated run time [sec.] max	actual run time [sec.] min	actual run time [sec.] avg.	actual run time [sec.] max	average overest. factor	interarrival time [sec.] min	interarrival time [sec.] avg.	interarrival time [sec.] max
CTC	0	24,324	64,800	0	10,958	64,800	2.220	0	369	164,472
KTH	60	13,678	216,000	0	8,858	216,000	1.544	0	1,031	327,952
LANL	1	3,683	30,000	1	1,659	25,200	2.220	0	509	201,006
SDSC	2	14,344	172,800	0	6,077	172,800	2.360	0	934	79,503

Table 2. Basic properties of the four traces (86,400 seconds = 1 day).

arrival times and the workload to be processed is increased. We use shrinking factors from 1.0 down to 0.6 in steps of 0.1.

To exclude singular effects resulting from the synthetic generation process, ten synthetic job sets, with 10,000 jobs each, are generated for each trace and are used as input for the simulations. After the simulation run is completed and all schedules are analyzed, the results are combined. This is done by neglecting the maximum and minimum value, so that the average is computed from the remaining eight results. Thereby, extreme singular results (both good and bad) are not considered in the £nal results that are presented in the following.

4.3 Results

We begin with studying the results of the three basic policies FCFS, SJF, and LJF (back£lling is implicitly done, as planning based resource management systems are assumed [6]). In Figure 1 and Figure 2 the slowdowns and utilizations, respectively, are shown at different shrinking factors (or equivalently, different workloads). It is seen that no clear winning policy can be found for all job inputs and workloads. For the KTH jobs, SJF is the best choice for all applied workloads. With CTC and SDSC, FCFS is the best choice up to a shrinking factor of 0.8, whilst with a further increased workload, SJF is better. At very high workloads (i. e. small shrinking factors) the system runs in a saturated state, in which a further increase of workload does not yield to a signi£cantly higher utilization. Jobs simply wait longer until they are started. A good example for this is seen with the SDSC jobs.

Observing slowdown and utilization, LJF can generate high utilizations, but at the cost of immense slowdowns. Similar applies to SJF, that achieves good slowdowns, but poor utilizations. Only FCFS has slowdowns as low as those of SJF and utilizations as high as those of LJF. Exact values are given in Table 4. From these numbers it is seen, that with an unmodi£ed workload (shrinking factor of 1.0), FCFS is a good choice for the CTC, LANL, and SDSC jobs, whilst SJF is the best policy for the KTH jobs. This shows, that for different job characteristics, different scheduling policies are required in order to get the best possible performance. However, for the evaluation of the self-tuning dynP scheduler and the preferred decider, we use SJF as the basis of our evaluations. This is, because we mostly focus on good slowdowns for satisfying the users. We de£ne SJF as the preferred policy of the unfair, preferred decider. With de£ning a preferred policy the decider only switches to a different policy, if the preferred policy is clearly worse. On the other hand the decider switches back to the preferred policy, if its performance is at least equal to the currently used policy. The aim is to receive an improved overall performance, i. e. a smaller slowdown at an increased utilization.

Figure 3 and Figure 4 show the performance (slowdown and utilization) curves for the self-tuning dynP scheduler with the advanced and the SJF-preferred decider. As reference the performance curve of SJF is also printed. The according numbers are found in Table 5. In addition to Table 4, relative differences of the achieved slowdowns (in %) and absolute differences of the utilization (in percentage-points) compared to SJF are shown. Unfortunately, no signi£cant differences between the advanced and the SJF-preferred decider are seen in the diagrams. However, it is seen, that with CTC jobs and light workloads (shrinking factors 1.0, 0.9 and 0.8) the improvements range between 10 and 18%. For the SDSC jobs slightly smaller improve-

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

ments in slowdown are achieved (around 7% with a maximum of 14% at shrinking factor 0.9). At the same time, the achieved utilizations are also improved. Improving the utilization by one percentage-point seems to be small, but the higher the utilization is, the more dif£cult it is for the scheduler to further improve it, as at some point the saturated state is reached.

As the two used metrics slowdown and utilization are contrary, improving one usually worsens the other. A good example are the two basic scheduling policies SJF and LJF. The self-tuning dynP scheduler is able to improve both metrics for the CTC and SDSC jobs. For the other two job sets this is not always possible at all applied workloads. For the KTH jobs and a shrinking factor of 0.7 the slowdown and utilization are actually worse than SJF.

In Table 3 averages of the differences to SJF for all applied shrinking factors are computed in order to join all results in a single value. This is done because of the scaling of the y-axis in the diagrams, where small differences are dif£cult to spot. The numbers indicate, that except for the KTH jobs the self-tuning dynP scheduler always improves the performance of SJF averaged over all workloads, regardless of using the fair advanced decider, or the unfair, SJF-preferred decider. Only for the CTC jobs it is bene£cial to prefer a single policy in the decision process of the self-tuning dynP scheduler. Particularly, if the workload is only increased up to a shrinking factor of 0.8. This shows that preferring a policy may result in a further increase of performance.

5 Conclusion

In this paper we presented a new decider mechanism for the self-tuning dynP scheduler. The self-tuning dynP scheduler switches the active scheduling policy dynamically during the run time of the resource management system. The aim of development is to react on changing job characteristics, without a human interaction. In contrast to the previously presented *advanced decider*, which is fair in its decisions, the new decider explicitly prefers one policy. We evaluated this *preferred decider* with means of a discrete event simulation environment. As job input we used four synthetic job sets, which are based on traces from the Parallel Workload Archive. The preferred decider is compared with the advanced decider and SJF is used as the preferred policy. The evaluation shows, that the *SJF-preferred* decider can outperform the basic policy SJF, in slowdown and utilization at the same time. With the CTC jobs in particular, the improvements can be up to 18% for the slowdown and about 2 percentage-points for the utilization. Unfortunately, such improvements are not possible for all used job sets and applied workloads.

We showed, that the presented self-tuning scheduler with dynamic policy switching is bene£cial to commonly used static scheduling approaches. Therefore, we think, that self-tuning schedulers, which are able to adapt their scheduling behavior according to the job characteristics of the currently waiting jobs, should be implemented in modern resource management systems.

References

[1] D. G. Feitelson. A Survey of Scheduling in Multiprogrammed Parallel Systems. Research report rc 19790 (87657), IBM T.J. Watson Research Center, Yorktown Heights, NY, 1995.

[2] D. G. Feitelson. Metrics for Parallel Job Scheduling and their Convergence. In D. G. Feitelson and L. Rudolph, editor, *Proc. of 7th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 190–208. Springer, 2001.

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

[4] 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.

[5] J. Gehring and F. Ramme. Architecture-Independent Request-Scheduling with Tight Waiting-Time Estimations. 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 65–80. Springer, 1996.

[6] 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.

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

[8] 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.

[9] 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.

[10] IBM Scalable Parallel (SC) software - LoadLeveler. `http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/loadlevel%er.html`, October 2003.

[11] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

	SLDwA: relative difference to SJF in %		utilization: absolute difference to SJF in percentage-points	
	advanced	SJF-preferred	advanced	SJF-preferred
CTC	9.04	9.92	1.22	1.21
KTH	0.15	-0.72	0.13	0.12
LANL	1.51	1.29	0.07	0.09
SDSC	6.36	6.22	0.93	0.91

Table 3. Condensed results of Table 5. Shown are the average differences of all workloads/shrinking factors for each job set. Note, positive differences for the slowdown are good, negative are bad.

[12] 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.

[13] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Maximizing Speedup through Self-Tuning of Processor Allocation. In *Proc. of the 10th International Parallel Processing Symposium*, pages 463–468. IEEE Computer Society Press, 1996.

[14] PBS - Portable Batch System. `http://www.openpbs.org/`, October 2003.

[15] PBS Pro Home. `http://www.pbspro.com`, October 2003.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

About The Authors

Achim Streit is a staff member at the Paderborn Center for Parallel Computing (PC^2) of Prof. Kao. In 1999 he received his diploma (MSc) in computer science at the Computer Engineering Institute of Prof. Schwiegelshohn. In 2003 he received his Ph.D. from the Faculty of Computer Science, Electrical Engineering, and Mathematics at the Paderborn University. His research interest is on parallel and distributed computing, in particular cluster, HPC, and grid computing. Herein he is working on the development and evaluation of job-scheduling algorithms for the resource management of HPC-clusters and grid environments.

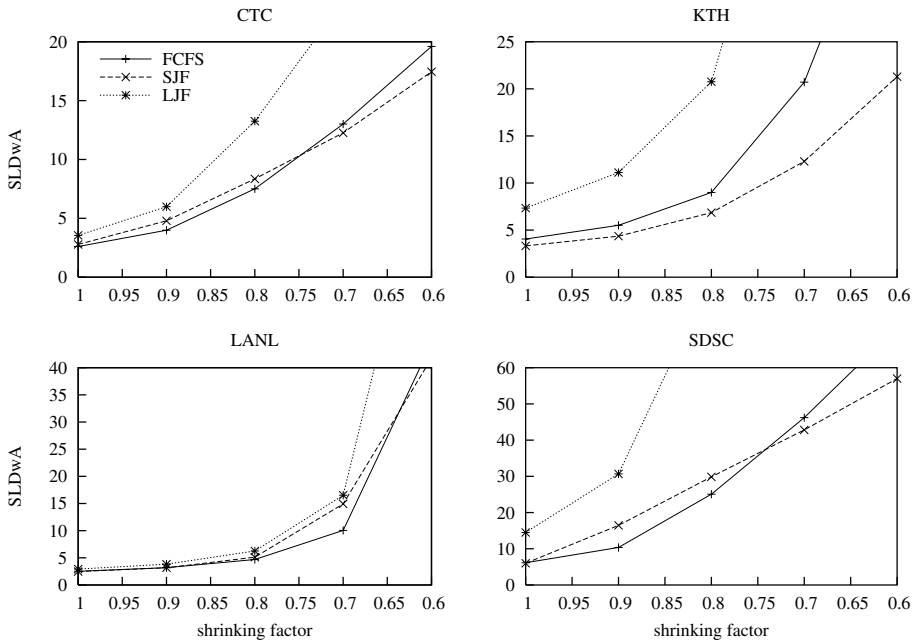

Figure 1. Average slowdown weighted by area (SLDwA) of FCFS, SJF, and LJF with different workloads/shrinking factors.

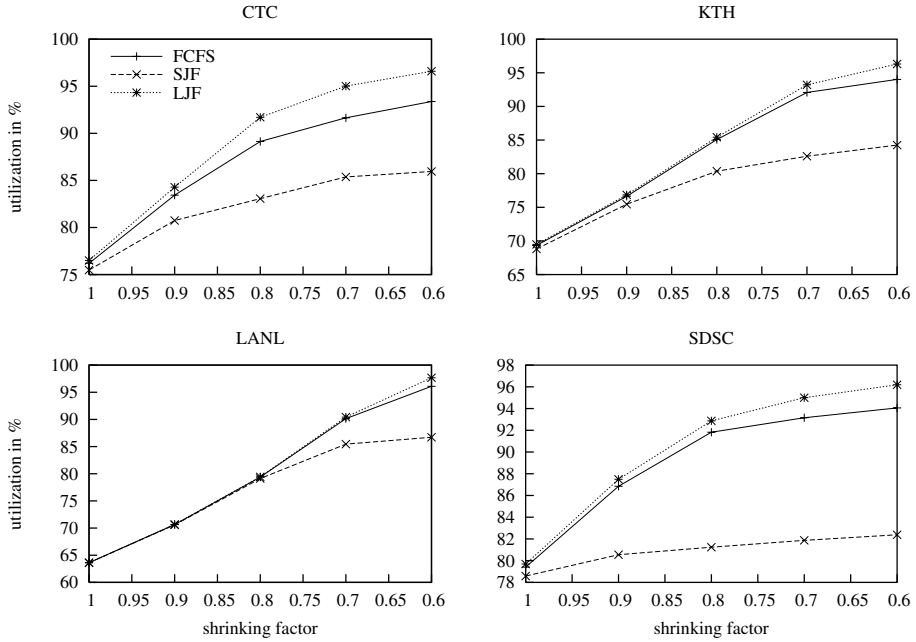

Figure 2. Utilization of FCFS, SJF, and LJF with different workloads/shrinking factors.

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

shrinking factor	SLDwA FCFS	SLDwA SJF	SLDwA LJF	utilization in % FCFS	utilization in % SJF	utilization in % LJF
CTC						
1.0	2.61	2.78	3.55	76.20	75.48	76.50
0.9	3.99	4.80	5.99	83.43	80.74	84.29
0.8	7.51	8.36	13.25	89.13	83.07	91.70
0.7	13.01	12.27	23.42	91.65	85.36	95.01
0.6	19.61	17.46	36.22	93.38	85.94	96.60
KTH						
1.0	4.06	3.32	7.33	69.33	68.81	69.48
0.9	5.51	4.35	11.11	76.64	75.46	76.84
0.8	9.00	6.85	20.75	85.08	80.37	85.41
0.7	20.72	12.29	54.58	92.08	82.59	93.20
0.6	45.73	21.29	120.84	94.03	84.25	96.30
LANL						
1.0	2.53	2.47	2.92	63.61	63.61	63.63
0.9	3.20	3.16	3.83	70.64	70.59	70.66
0.8	4.69	5.11	6.26	79.37	79.11	79.42
0.7	10.05	14.93	16.52	90.13	85.46	90.43
0.6	44.46	41.73	82.88	96.10	86.71	97.67
SDSC						
1.0	6.16	6.00	14.49	79.41	78.59	79.69
0.9	10.36	16.48	30.70	86.85	80.55	87.49
0.8	25.06	29.86	84.77	91.83	81.23	92.87
0.7	46.20	42.83	121.05	93.15	81.87	95.00
0.6	71.08	57.01	162.54	94.05	82.38	96.19

Table 4. Detailed numbers for Figure 1 and Figure 2.

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

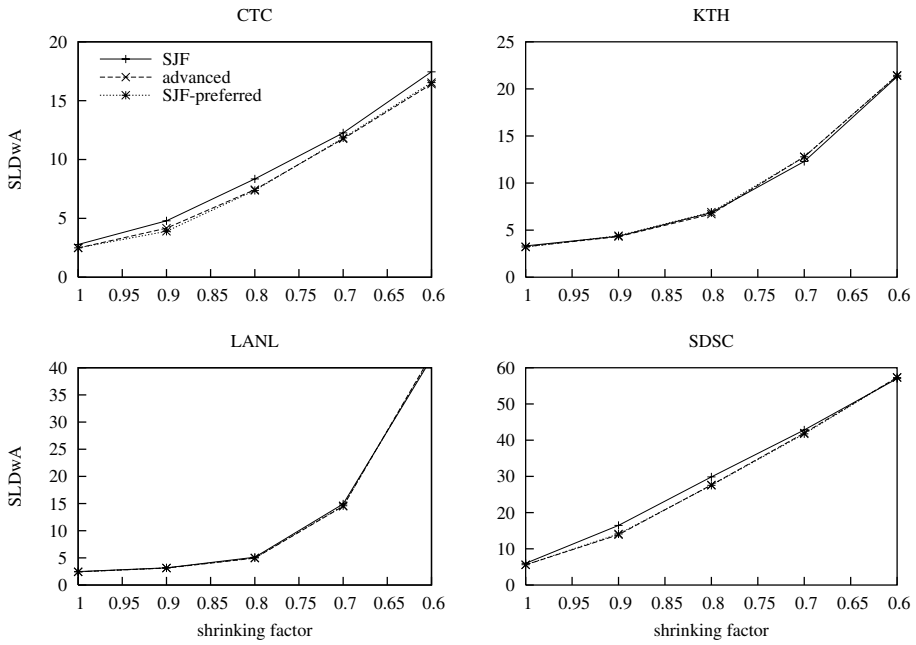

Figure 3. Average slowdown weighted by area (SLDwA) of the self-tuning dynP scheduler with the advanced and SJF-preferred decider.

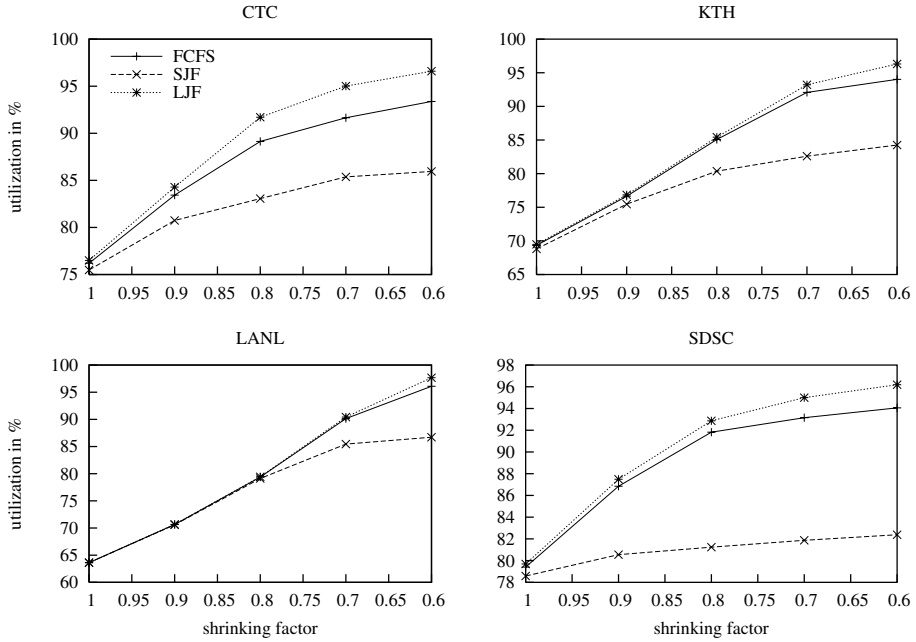

Figure 4. Utilization of the self-tuning dynP scheduler with the advanced and SJF-preferred decider.

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

shrinking factor	SLDwA SJF	SLDwA adv.	SLDwA SJF-pref.	rel. difference to SJF in % adv.	rel. difference to SJF in % SJF-pref.	utilization in % SJF	utilization in % adv.	utilization in % SJF-pref.	abs. difference to to FCFS in %-points adv.	abs. difference to to FCFS in %-points SJF-pref.
CTC										
1.0	2.78	2.48	2.49	10.92	10.39	75.48	76.07	76.13	0.59	0.64
0.9	4.80	4.16	3.90	13.19	18.65	80.74	82.09	82.54	1.35	1.80
0.8	8.36	7.44	7.37	10.92	11.79	83.07	84.84	84.72	1.77	1.65
0.7	12.27	11.76	11.83	4.12	3.56	85.36	86.32	86.30	0.96	0.94
0.6	17.46	16.40	16.54	6.03	5.23	85.94	87.39	86.95	1.45	1.01
average				9.04	9.92				1.22	1.21
KTH										
1.0	3.32	3.25	3.20	2.15	3.56	68.81	69.04	68.98	0.23	0.17
0.9	4.35	4.31	4.42	1.04	-1.46	75.46	75.68	75.68	0.22	0.22
0.8	6.85	6.70	6.91	2.17	-0.87	80.37	80.72	80.63	0.35	0.26
0.7	12.29	12.79	12.80	-4.04	-4.07	82.59	82.37	82.42	-0.22	-0.17
0.6	21.29	21.41	21.45	-0.57	-0.75	84.25	84.33	84.40	0.08	0.15
average				0.15	-0.72				0.13	0.12
LANL										
1.0	2.47	2.43	2.42	1.81	1.96	63.61	63.61	63.61	0.00	0.00
0.9	3.16	3.13	3.13	1.11	0.95	70.59	70.63	70.63	0.04	0.04
0.8	5.11	4.95	5.00	3.28	2.19	79.11	79.14	79.12	0.03	0.01
0.7	14.93	14.50	14.58	2.92	2.34	85.46	85.64	85.57	0.18	0.11
0.6	41.73	42.37	42.13	-1.55	-0.97	86.71	86.81	87.00	0.10	0.29
average				1.51	1.29				0.07	0.09
SDSC										
1.0	6.00	5.56	5.59	7.29	6.78	78.59	78.75	78.73	0.16	0.14
0.9	16.48	13.90	14.09	15.66	14.48	80.55	81.99	82.20	1.44	1.64
0.8	29.86	27.64	27.54	7.43	7.76	81.23	82.59	82.42	1.36	1.19
0.7	42.83	41.95	41.74	2.05	2.56	81.87	83.01	82.96	1.14	1.08
0.6	57.01	57.35	57.29	-0.61	-0.50	82.38	82.94	82.86	0.56	0.49
average				6.36	6.22				0.93	0.91

Table 5. Detailed numbers for Figure 3 and Figure 4. Additionally printed are differences to SJF of the slowdown in % and of the utilization in percentage-points. Note, positive differences for the slowdown are good, negative are bad. The abbreviations 'adv.' stands for the advanced decider and 'SJF-pref.' for SJF-preferred decider respectively.

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