The Self-Tuning dynP Job-Scheduler *

Achim Streit

PC^2- Paderborn Center for Parallel Computing,
Paderborn University,
33102 Paderborn, Germany
E-mail: streit@upb.de

Abstract

In modern resource management systems for supercomputers and HPC-clusters the job-scheduler plays a major role in improving the performance and usability of the system. The performance of the used scheduling policies (e.g. FCFS, SJF, LJF) depends on the characteristics of the queued jobs. Hence we developed the dynP *scheduler family. The basic idea was to change between different scheduling policies during runtime. The basic* dynP *scheduler uses the average estimated runtime of all queued jobs together with two input parameters to decide when a policy change may be benefical. A disadvantage is that the performance of the basic* dynP *scheduler strongly depends on the right setting of the two input parameters.*

Therefore we present the self-tuning dynP *scheduler, which is totally independent from any parameter values. The basic concept is that the self-tuning* dynP *scheduler computes the full (virtual) schedule for each of the three policies in every scheduling step. Each computed schedule is rated by a criterion. Then the scheduler switches to that policy which generated the best schedule for the currently queued jobs. In this paper we are using simulations with trace based job sets to evaluate the performance of the scheduler. The achieved results are reasonably good compared to the parameterized* dynP *variant and the basic policies FCFS, SJF, and LJF.*

Keywords: job-scheduling, self-tuning, HPC-cluster, evaluation, simulation

1 Introduction

Supercomputers and especially HPC-clusters became more and more popular over the last years. With that a variety of different resource management systems were developed (e.g. PBS + Maui-Scheduler, LoadLeveler + EASY-Scheduler, Codine (now SUN GridEngine), or CCS). One important component of these software systems is the job-scheduler. In a production environment the performance of the scheduler is primarily responsible for a good or bad overall machine performance and user acceptance. Developing and improving job-schedulers is often done with the aid of simulations (Feitelson *et al.* [3, 9, 12] for the IBM SP2, or by Franke, Moreira *et al.* [4, 8] for the ASCI Blue-Pacific).

This paper presents a similar approach: improving the job-scheduler of CCS (Computing Center Software) for our Fujitsu-Siemens *hpcLine* Cluster. CCS [6, 7] was developed at the PC^2 over the last years to manage and access our machines. The *hpcLine* [5] consists of 96 double processor Intel Pentium III 850 MHz nodes. The nodes are connected with a SCI (Scalable Coherent Interface) network, which combines high application level bandwidth (85 MB/s) with low latency (5.1 μs for ping-pong/2).

The cluster is operated in space-sharing mode as users often need the complete compute power of the nodes and the exclusive access to the network interface for MPI parallelized applications. Currently three space-sharing scheduling policies are implemented in CCS: First Come First Serve (FCFS), Shortest Job First (SJF), and Longest Job First (LJF) each combined with conservative backfilling. CCS also provides trace information of its scheduled jobs.

We use these trace files as input for our simulation-based evaluations, to develop new and optimize existing scheduling algorithms. For that we built a simulation framework for job-scheduling, which comes with a large set of tools: job set converters (reads the Standard Workload Archive [11] and CCS format), a job set analyzer, programs to modify job sets, a schedule analyzer, a schedule viewer, and the numerous schedulers itself.

The remainder of this paper is structured as follows. In Section 2 the self-tuning dynP Scheduler is presented. In the first part of Section 3 the used performance metrics and job sets are described. Finally the performance is evaluated

*This work has been funded partly by BMBF (Federal Ministry of Education and Research) project UNICORE Plus, grant 01 IR 001.

and compared to previous results.

2 The Self-Tuning dynP Scheduler

In [10] we presented the basic dynP scheduler (for **dyn**amic **P**olicy), which changes its sorting policy (FCFS, SJF, and LJF) for the job queue online. During our first evaluations of FCFS, SJF and LJF with our trace-based job sets we found out that the job sets came with such different characteristics that no policy was superior to the others in every case. That lead us to the idea of the basic dynP scheduler.

The scheduler uses the average estimated runtime of all queued jobs to decide which policy to take. We use this criterion as 1) the performance of each strategy depends on the characteristics of the job set, and 2) two of the policies (SJF and LJF) are using this criterion to order the waiting queue. Furthermore two bounds are needed to decide when to switch from one policy to another. With that the core algorithm of the basic dynP scheduler works as follows:

```
IF (jobs in waiting queue >= 5)
   AERT = average estimated runtime of all
          jobs currently in the waiting queue;
   IF (0 < AERT <= lower_bound) {
      switch to SJF;
   } ELIF (lower_bound < AERT <= upper_bound) {
      switch to FCFS;
   } ELIF (upper_bound < AERT) {
      switch to LJF;
   }
   reorder waiting queue according to new policy;
}
```

We are using a threshold of 5 jobs in the waiting queue to prevent the scheduler from unnecessary policy changes.

A disadvantage of the basic dynP scheduler is, that its performance strongly depends on the right setting of the two bounds. With the proper setting (7 200 s for the lower bound and 9 000 s for the upper bound) the basic dynP scheduler outperforms FCFS, which is a good average for the used job sets (cf. Fig. 1, taken from [10]).

From this disadvantage the need for an adaptive or self-tuning algorithm arose. At the beginning of each decision step the scheduler computes the full schedule for each of the three policies and then rates each schedule according to a quality parameter. Then the scheduler changes to that policy with achieved the best performance in this step.

Similar work was done by Feitelson and Naaman [2]. They used genetic algorithms and a fitness function to search for optimal parameter values for their scheduling algorithms.

So with that the scheduler can automatically choose the best sorting policy without any parameter input. The core algorithm (decider) of the self-tuning dynP scheduler works as follows:

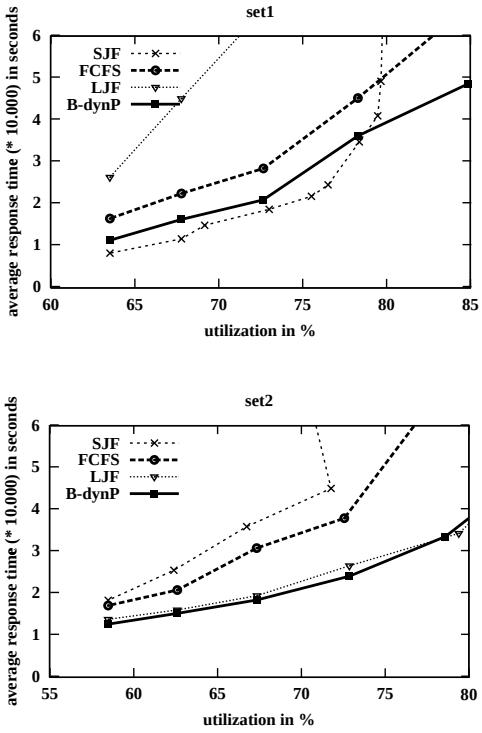

Figure 1. Average Response Time for the basic dynP scheduler (lower bound: 7200 s, upper bound: 9000 s).

```
IF (jobs in waiting queue >= 5) {
    mySchedule = generate_Schedule("FCFS");
    FCFS_Quality =
      mySchedule.getQuality(Quality_Parameter);
    mySchedule = generate_Schedule("SJF");
    SJF_Quality =
      mySchedule.getQuality(Quality_Parameter);
    mySchedule = generate_Schedule("LJF");
    LJF_Quality =
      mySchedule.getQuality(Quality_Parameter);
    IF (SJF_Quality <= LJF_Quality) {
        IF (FCFS_Quality <= SJF_Quality) {
            NewPolicy = "FCFS";
        } ELSE {
            NewPolicy = "SJF";
        }
    } ELSE {
        IF (FCFS_Quality <= LJF_Quality) {
            NewPolicy = "FCFS";
        } ELSE {
            NewPolicy = "LJF";
        }
    }
}
```

The *Quality_Parameter* specifies the metrics for evaluating the virtual schedules and is one of the following:

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

- Makespan:

$$MS = \max_{j \in Jobs} j.endTime$$

- Average Response Time:

$$ART = \frac{\sum_{j \in Jobs} j.responseTime}{|Jobs|}$$

- Average Response Time weighted by Width:

$$ARTwW = \frac{\sum_{j \in Jobs} (j.reqRes * j.responseTime)}{\sum_{j \in Jobs} j.reqRes}$$

- Average Response Time weighted by Area:

$$ARTwA = \frac{\sum_{j \in Jobs} (j.area * j.responseTime)}{\sum_{j \in Jobs} j.area}$$

with:

- $j.responseTime = j.submitTime - j.endTime$,
- $j.reqRes =$ number of requested resources by the job,
- $j.area = j.reqRes * j.reqRunTime$.

Later in this paper these four quality parameters and their impact on the performance is compared. The end time of a job ($j.endTime$) is computed from the scheduled start time and the requested (not actual !) run time of the job. Note, that the best schedule has the lowest quality number.

Benefit of the self-tuning dynP scheduler is that the two bounds of the basic dynP scheduler are left out, so that the system administrator does not has to tune these parameters. Therefore the scheduler can work totally on its own, without any administrative interference during runtime.

A worst case scenario for the self-tuning dynP scheduler could look like the following: Because of the characteristics of the incoming jobs the scheduler is forced to switch between SJF and LJF, without ever using FCFS. Then a job with a medium estimated runtime would wait forever, as for both policies such jobs would stay in the middle of the queue without being started. Note, that worst-case scenarios (regarding fairness to the submit time) can also be constructed for the basic SJF and LJF policies, in which long jobs (for SJF) or short jobs (for LJF) may wait forever.

In the following we use abbreviations to distinguish between the different variants: B-dynP is the basic dynP scheduler using two bounds, ST-dynP is the self-tuning dynP scheduler with a quality metrics appended (e.g. ST-dynP-ARTwW).

3 Performance Evaluation

This section is structured in three parts. At the beginning the metrics used for the evaluation are presented, followed by a description of the workload model and the used job sets. Finally the results are presented.

3.1 Metrics

Later we show diagrams where the average response time weighted by job width (ARTwW, see above) is plotted on the y-axis, and the utilization on the x-axis. We define the utilization of the machine as:

$$UTIL = \frac{\sum_{j \in Jobs} (j.reqRes * j.runTime)}{N * (lastEndTime - firstStartTime)}$$

with:

- N is the total number of available resources of the machine (i.e. nodes of the cluster),
- $j.runTime = j.endTime - j.startTime$, and
- $firstStartTime$ and $lastEndTime$ are the first and last event that occur in the schedule.

With that the utilization is the total squashed area of all jobs divided by the area of the total schedule. Note, that in a simulation the $firstStartTime$ is equal to the $submitTime$ of the first submitted job (usually 0 seconds), as this job can always be started right away.

We use the average response time weighted by job width as an user centric and the machine utilization as an owner centric criterion to measure the performance of the different scheduling algorithms.

3.2 Workload Model

As stated before the job sets used for the simulations are based on trace data generated by CCS. The original job sets are modified by reducing the average interarrival time between two consecutive job submits. For that a *shrinking factor* from 0.75 down to 0.25 in steps of 0.05 is used. This enables us to evaluate the performance of the ST-dynP scheduler over a wide range of workloads and utilizations respectively.

The two job sets were already used in [10] before, so that results can be compared. At that time these two job sets were chosen as they represent the current user behavior during the lifetime of the machine best, combined with a different overall characteristic. As can be seen in Tab. 1 set1 consists of short jobs that are submitted at a high rate in contrast to set2, where jobs are more than twice as long, but are submitted with a greater lag.

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

job set	number of jobs	average requested nodes	average actual runtime	average estimated runtime	average interarrival time
set1	8 469	13.49	2 835 s	6 554 s	628 s
set2	8 166	10.11	6 697 s	21 634 s	1 277 s

Table 1. Job set characteristics

The different characteristics of the two job-sets can also be seen, if a classification for the estimated job runtime is done (cf. Fig. 2 and Tab. 2).

class	lower bound [sec.]	upper bound [sec.]	
0	0	300	5 min
1	301	600	10 min
2	601	1 200	20 min
3	1 201	1 800	30 min
4	1 801	3 600	1 h
5	3 601	7 200	2 h
6	7 201	14 400	4 h
7	14 401	21 600	6 h
8	21 601	43 200	12 h
9	43 201	∞	∞

Table 2. Classes for actual runtime in Fig. 2

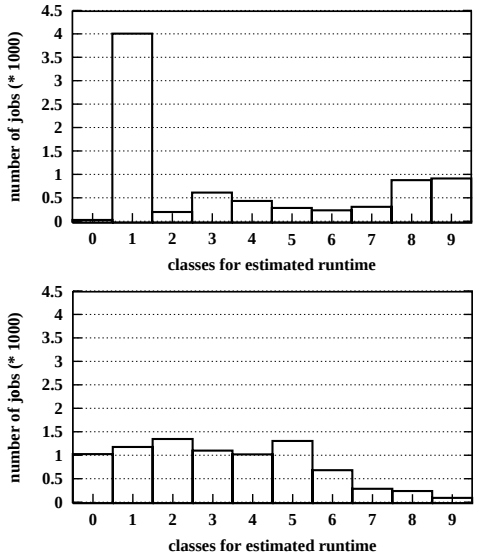

Figure 2. Number of jobs grouped by estimated runtime (cf. Tab. 2). `set1` **left,** `set2` **right.**

3.3 Results

When comparing the performance of schedulers we are looking at the behavior in two ways: 1) maximum utilization in saturated state and 2) ARTwW at medium utilizations ($\sim$ 60 to 80%). A saturated state is reached, when a further increase of workload does not increase the achieved utilization, but dramatically decreases the quality of the schedule (i.e. the ARTwW grows). In a real world scenario it would be more favorable for us to concentrate on medium utilizations, as our machines usually operate in these conditions.

At the beginning we take a look at the behavior of the ST-dynP scheduler with the four different quality parameters. For set1 (cf. top diagram in Fig. 3) all four quality parameters from Sec. 2 reach similar utilizations of $\sim$95%. But for set2 (cf. top diagram in Fig. 4) different maximum utilizations are achieved (between $\sim$90 and 95%).

One might think now that this is obvious, as we use the ARTwW metrics for optimizing and also evaluating (cf. Sec. 3.1) the schedule, and then a comparison with the other four metrics is done. But a similar characteristic is seen, when looking at the numbers for the unweighted ART instead of ARTwW (bottom diagrams in Fig. 3 and Fig. 4.

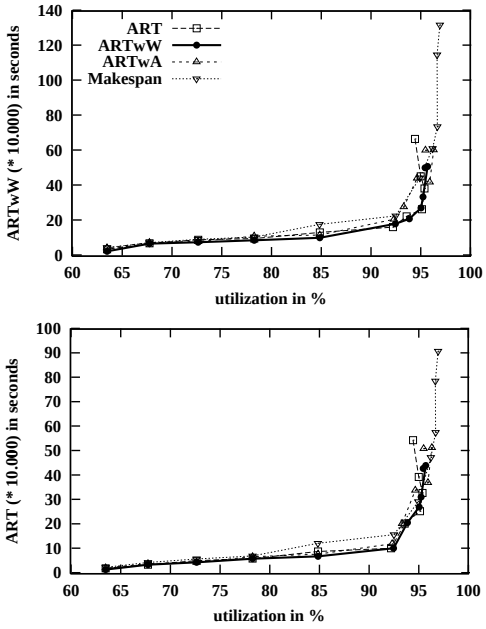

Figure 3. Impact of the four different metrics on the performance of the ST-dynP **scheduler, concentrating on the saturated state. Performance metrics are ART (top) and ARTwW (bottom). Here for set1, set2 in Fig. 4.**

Comparing the four different quality parameters at examplary data points (shrinking factors), Tab. 3 shows that almost always ARTwW is best. Especially for set1 (upper table) the performance gain is obvious. This can also be seen in Fig. 5 for medium utilizations, where the quality parameter ARTwW shows the best performance. Hence we

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

set1

shrinking factor	utilization	ST-dynP ART	**ARTwW**	ARTwA	Makespan
0.75	∼63%	40 231 s	**28 524** s	48 900 s	50 048 s
0.6	∼78%	101 444 s	**91 977** s	115 187 s	110 772 s
0.25	∼95%	670 089 s	**506 033** s	606 254 s	1 322 425 s

set2

shrinking factor	utilization	ST-dynP ART	**ARTwW**	ARTwA	Makespan
0.75	∼58%	25 691 s	**26 183 s**	29 086 s	29 119 s
0.6	∼73%	57 512 s	**57 249 s**	61 430 s	63 596 s
0.25	∼94%	944 126 s	**809 829 s**	1 144 552 s	1 067 972 s

Table 3. ST-dynP**: Average Response Time weighted by job Width (ARTwW) for exemplary workloads.**

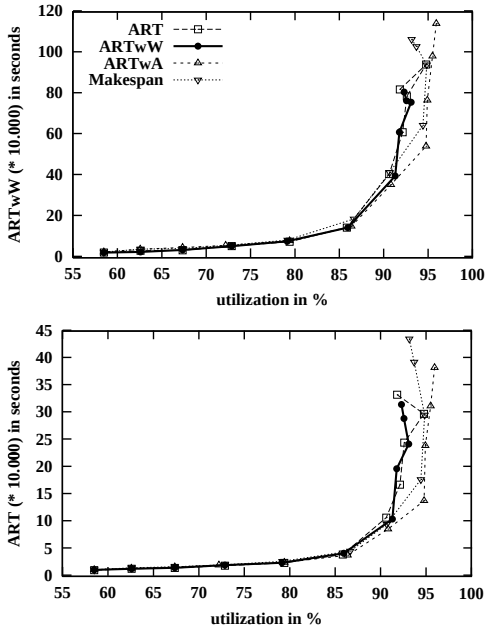

Figure 4. Same as Fig. 3, but now for set2.

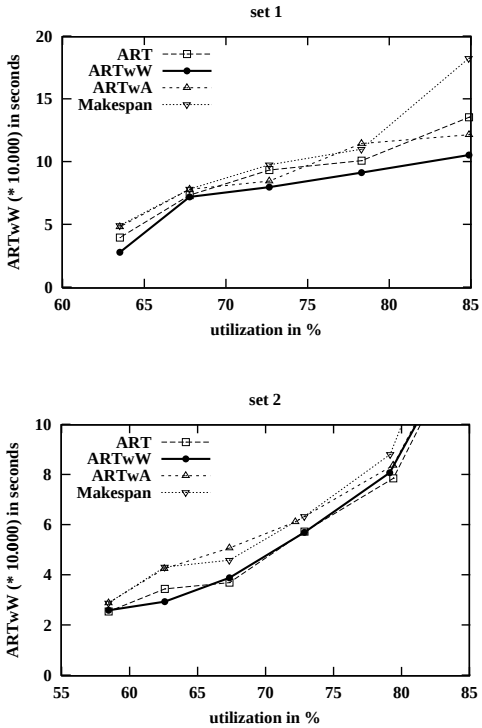

Figure 5. Same diagrams as in Fig. 3 and Fig. 4 (top diagrams, performance metrics: ARTwW), but now focusing on medium utilizations.

are using ST-dynP with the quality parameter ARTwW (so ST-dynP-ARTwW) in the rest of this paper for comparing the performance of ST-dynP with FCFS, SJF, LJF, and B-dynP (with bounds: 7 200 sec. and 9 000 sec.).

Therefore in Fig. 6 we added the curve for ST-dynP-ARTwW to the performance diagrams of [10]. In Tab. 4 the numbers for exemplary shrinking factors are printed in detail.

Looking at the saturated state shows that ST-dynP-ARTwW achieves a higher maximum utilization for set1, but a lower one for set2 compared to B-dynP. But the differences are considerably small ($\sim$ 2%). The ARTwW values at these high utilizations are more different, as can be seen in Tab. 4 for shrinking factor 0.25 especially for set2. Note, that the ARTwW values for SJF are considerably lower, but also the reached utilization in saturated state is not as high as with the other schedulers.

Now focusing on medium utilizations (lower part of Fig. 6) shows, that the performance of ST-dynP-ARTwW compared to B-dynP is worse for set1, but better for set2 (see also Tab. 4 for details). We bought this loss with the convenience of a totally closed system which does not require an expert for setting the input parameters of the sched-

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

set1

shrinking factor	ST-dynP-ARTwW	B-dynP	FCFS	SJF	LJF
0.75	28 524 s 63.53 %	13 831 s 63.53 %	16 683 s 63.52 %	10 445 s 63.53 %	53 432 s 63.53 %
0.6	91 977 s 78.31 %	38 016 s 78.31 %	41 467 s 78.31 %	19 122 s 72.58 %	148 342 s 78.31 %
0.25	506 033 s 95.47 %	629 396 s 93.93 %	611 238 s 94.55 %	81 448 s 86.20 %	1 370 551 s 97.12 %

set2

shrinking factor	ST-dynP-ARTwW	B-dynP	FCFS	SJF	LJF
0.75	26 183 s 58.49 %	25 818 s 58.49 %	18 873 s 58.49 %	17 008 s 58.49 %	29 845 s 58.49 %
0.6	57 249 s 72.89 %	77 005 s 72.89 %	41 820 s 72.81 %	32 841 s 71.98 %	74 829 s 72.89 %
0.25	809 829 s 92.59 %	1 439 526 s 94.60 %	790 715 s 93.73 %	230 314 s 81.01 %	1 815 028 s 97.57 %

Table 4. Overall comparison with ST-dynP-ARTwW, B-dynP, and the core algorithms FCFS, SJF, LJF: ARTwW [s] and utilization [%] for exemplary shrinking factors.

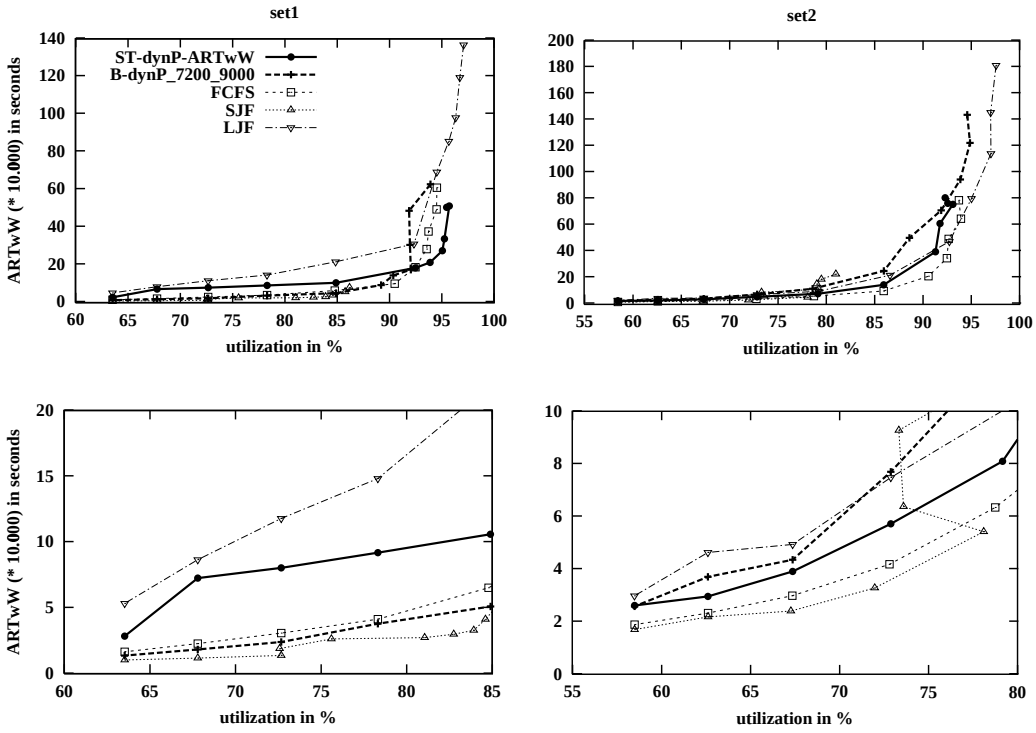

Figure 6. Comparing ST-dynP-ARTwW, B-dynP, and the core algorithms FCFS, SJF, LJF. Lower diagrams again focus on medium utilization.

uler. This leads to a good average case scheduler. Additionally the scheduler now reacts on different job characteristics without readapting any parameters.

This mechanism of reacting on different job characteristics can also be presented in a diagram, where the utilization of the machine is again printed on the x-axis. But now on the y-axis an accumulated percentage is printed, which depicts how much each policy (SJF, FCFS, LJF) was used during the whole scheduling process. The ST-dynP-ARTwW scheduler obviously favors SJF and FCFS for all shrinking

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

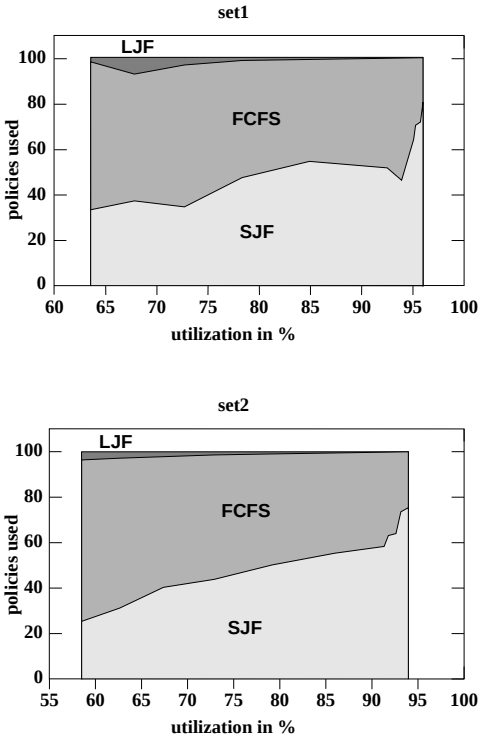

Figure 7. Accumulated percentages of policies (SJF, FCFS, LJF) used by the ST-dynP-ARTwW **scheduler.**

factors. This is ok for set1, as it consists of many short jobs (cf. Tab. 1 and Fig. 2), but when using set2 the ST-dynP-ARTwW scheduler should clearly favor LJF. However the scheduler tries to optimizes the generated schedule in each scheduling step, this local optimization also leads to good global results for set1. The specific characteristic of set2 on the other hand seems to be counterproductive for the local optimization done in each scheduling step.

The ST-dynPscheduler has to compute three full schedules in each scheduling step, so one might ask: How much does it cost respectively how long does it take ? In a real world scenario the time for the scheduling phase is negligible compared to the configuration of the partition on the cluster.

In the future we have to check, if a similar behavior of the ST-dynP-ARTwW scheduler also occurs for job sets based on traces from others machines (like used in [1]), which come with different characteristics.

4 Conclusion

In this paper we presented the self-tuning dynP scheduler for scheduling HPC-clusters. It is based on the concept of changing the policy (FCFS, SJF, and LJF) for sorting the job queue dynamically during runtime. The basic dynP scheduler needed two parameters which settings have a strong influence on the schedulers performance.

Therefore we developed the self-tuning dynP scheduler, which automatically finds the best solution in each scheduling step. The basic idea is to generate virtual schedules for the three policies and compute their quality (average response time weighted by job width). Then the best schedule/policy is chosen in this scheduling step.

Comparing self-tuning dynP to the basic dynP scheduler showed, that always a better performance was reached when in saturated state. At medium utilizations apposing results were performed (worse for set1, better for set2). But this loss of performance earned us the self-tuning ability, which relieves the system-administrator from precisely setting the input parameters of the basic dynP scheduler.

As we used preliminary simulations to evaluate the performance of self-tuning dynP, it is also planned to add this new scheduler to our resource management system CCS. Then the everyday usage of the self-tuning dynP scheduler will show, whether the simulated results from this paper are achieved or not.

Acknowledgements

We like to thank our colleague Axel Keller who gracefully provided the job traces from CCS and also helped in understanding the scheduling process in CCS.

References

[1] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R.Yahyapour. On Advantages of Grid Computing for Parallel Job Scheduling. In *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002)*, to appear 2002.

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

[3] D. G. Feitelson and A. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98)*, pages 542–547, Los Alamitos, March 1998. IEEE Computer Society.

[4] H. Franke, J. Jann, J. Moreira, P. Pattnaik, and M. Jette. An Evaluation of Parallel Job Scheduling for ASCI Blue-Pacific. In *Proceedings of SC'99, Portland, Oregon*, pages 11–18. ACM Press and IEEE Computer Society Press, 1999.

[5] The *hpcLine* at the Paderborn Center for Parallel Computing (PC^2). http://www.upb.de/pc2/services/systems/psc/index.html, Januar 2002.

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

[6] A. Keller and A. Reinefeld. CCS Resource Management in Networked HPC Systems. In *Proc. of Heterogenous Computing Workshop HCW'98 at IPPS, Orlando, 1998; IEEE Computer Society Press*, pages 44–56, 1998.

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

[8] J. E. Moreira, H. Franke, W. Chan, and L. L. Fong. A Gang-Scheduling System for ASCI Blue-Pacific. In *Proceedings of the 7th International Conference in High-Performance Computing and Networking (HPCN'99)*, volume 1593 of *Lecture Notes in Computer Science*, pages 831–840. Springer, 1999.

[9] A. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. Technical Report 2000-33, Institute of Computer Science, The Hebrew University, July 2000.

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

[11] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, Januar 2002.

[12] D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP2 Scheduler Using Slack-Based Backfilling. TR 98-13, Hebrew University, Jerusalem, April 1999.

About The Authors

Achim Streit has studied computer science combined with electrical engineering at Dortmund University (Germany). During 1999 he did his diploma in computer science at the Computer Engineering Institute of Prof. Schwiegelshohn. Since November 1999 he is a staff member at the Paderborn Center for Parallel Computing (PC^2) and is working on his Ph.D. His research interest focuses in the field of parallel computing, in particular high-performance and meta/grid-computing. Herein he is studying and developing new scheduling algorithms for HPC-clusters and grid environments.

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**