On Job Scheduling for HPC-Clusters and the dynP Scheduler

Achim Streit

PC^2– Paderborn Center for Parallel Computing,
33102 Paderborn, Germany,
streit@upb.de
http://www.upb.de/pc2

Abstract. Efficient job-scheduling strategies are important to improve the performance and usability of HPC-clusters. In this paper we evaluate job-scheduling strategies (FCFS, SJF, and LJF) used in the resource management system CCS (Computing Center Software). As input for our simulations we use two job sets that are generated from trace files of CCS.
Based on the evaluation we introduce the dynP scheduler which combines the three scheduling strategies and dynamically changes between them online. The average estimated runtime of jobs in the waiting queue and two bounds are used as the criterion whether or not dynP switches to a new policy. Obviously the performance of dynP depends on the setting of the two bounds. Diverse parameter pairs were evaluated, and with the proper setting dynP achieves an equal or even better (+9%) performance.

1 Introduction

In recent years clusters of workstations became more and more popular which is surely based on the good price-performance ratio compared to common supercomputers (SP2, T3E, SX-5, Origin, or VPP). Nowadays two types of clusters are found: HTC-clusters (High Throughput Computing) are very popular as the nodes hardware consists of low-cost "components of the shelf" from the computer shop around the corner connected with one Fast Ethernet network. This network limits the scalability of the system, so that tightly coupled applications are less suitable for HTC-clusters.

In contrast HPC-clusters (High Performance Computing) combine a fast (high bandwidth and low latency) interconnect and powerful computing nodes. The 192-processor cluster at the PC^2 is such a HPC cluster. It was installed in 1998 as a prototype of the new Fujitsu-Siemens *hpcLine* product line. The cluster consists of 96 double processor Intel Pentium III 850 MHz nodes. The SCI interconnect combines high bandwidth (85 MByte/s) with low latency (5.1 μs for ping-pong/2). We use our own resource management system called CCS (Computing Center Software) [9,10] for accessing and managing the cluster. 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.

B. Monien, V.K. Prasanna, S. Vajapeyam (Eds.): HiPC 2001, LNCS 2228, pp. 58–67, 2001.

Currently three space-sharing scheduling strategies are implemented in CCS: First Come First Serve (FCFS), Shortest Job First (SJF), and Longest Job First (LJF) each combined with conservative backfilling. As the scheduler has a strong influence on the system performance, it is worth while to improve the performance of the scheduling strategy. In the past preliminary evaluations based on simulations were done by Feitelson *et al.* [1,14] for the IBM SP2, or by Franke, Moreira *et al.* [7,12] for the ASCI Blue-Pacific.

The remainder of this paper is organized as follows. Section 2 describes the scheduling strategies and backfilling in detail. The two job sets for the evaluation are described in Section 3. After the performance metrics are described in Section 4, the concept of the new dynP scheduler is presented in Section 5. Finally Section 6 compares the performance of dynP to SJF, FCFS, and LJF.

2 Scheduling Strategies

Today's resource management systems generally operate after a queuing system approach: jobs are submitted to the system, they might get delayed for some time, resources are assigned to them, and they leave the queueing system. A classification of queuing systems was done by Feitelson and Rudolph in [4]. Here we look at an open, on-line queueing model. In a simulation based evaluation of job scheduling strategies the two important phases of a scheduling strategy are: 1) putting jobs in the queue (at submit time) and 2) taking jobs out of the queue at start time.

Different alternatives are known for the two phases (details in [2]). Here we sort the waiting queue by increasing submission time (FCFS) and increasing (SJF) or decreasing (LJF) estimated runtime. Furthermore we always process the head of the queue. With that the scheduling process looks as follows:

```
WHILE (waiting queue is not empty) {
   get head of queue (job j);
   IF (enough resources are free to start j) {
      start j;
      remove j from queue;
   } ELSE {
      backfill other jobs from the queue;
   }
}
```

Backfilling is a well known technique to improve the performance of space-sharing schedulers. A backfilling scheduler searches for free slots in the schedule and utilizes them with suitable jobs without delaying other already scheduled jobs. For that, users must provide an estimated runtime for their jobs, so that the scheduler can predict when jobs are finished and others can be started. Jobs are cancelled, if they run longer than estimated. Two backfilling variants are known: *conservative* [3] and *aggressive* [11]. The aggressive variant is also known as EASY (Extensible Argonne Scheduling sYstem [13]) backfilling.
As the name implies aggressive backfilling might delay the start of jobs in the

queue. The drawbacks are 1) an unbounded delay for job start, and 2) an unpredictability of the schedule [1]. Primarily due to the unbounded delay (especially for interactive jobs), aggressive backfilling is not used in CCS.

3 Workload Model

The workload used in this evaluation is based on trace data generated by CCS. In these traces CCS logs the decisions (start and end time of a job) of the scheduler. Since CCS did not log the submit time of a job in the past, the waiting time (from submit to start) could not be derived from the here used original trace.

Using this trace again as input for a scheduler (and for the evaluation) would result in meaningless performance numbers. Each job will be started immediately with zero waiting time, regardless of the scheduling strategy. To achieve useful results the trace needs to be modified to generate a higher workload. When the workload is increased the scheduler is no longer able to schedule each newly submitted job immediately. This leads to a filled waiting queue (i.e. backlog) and waiting times greater zero. As the backlog grows, a backfilling scheduler has more opportunities to find suitable jobs to backfill.
Methods for modifying the trace are:

1. A smaller average interarrival time: Here the duration between each two job submits is reduced. Hence it is possible to simulate more users on the cluster. By reducing the duration down to 0, an offline scheduling is simulated.
2. Extension of job runtime: As the estimated runtime is also given, both values (actual and estimated runtime) have to be modified. This would remove common characteristics from the job set, like e.g. specific bounds for the estimated runtime (i.e. 2 hours). Furthermore the used area of jobs would be changed.

In this case we use the first method and introduce the *shrinking factor*. The submit time of each job is multiplied with this factor. We use values ranging from 0.75 down to 0.25 in steps of 0.05. This enables us to evaluate the performance of different scheduling strategies over a wide range of workloads.

As trace data from two years is available, we extracted two significant job sets from the data pool. These two job sets (Tab. 1) were chosen, because of their different characteristics and because these job-sets quite well represent our current users. In `set1` shorter jobs are submitted at a high rate compared to `set2`. In `set2` the jobs are more than twice as long, but are submitted with a greater lag.

As can be seen in Fig. 1 both job sets have a lot of narrow (small number of requested nodes) jobs and only a moderate number of wide (highly parallel) jobs. Furthermore peaks are noticeable at powers of 2. Computing the product of width (requested nodes) and length (actual runtime) leads to Fig. 2. Nodedays more likely represent the resource usage of jobs. A more detailed look on the estimated runtime of jobs can be found in [2].

Table 1. Job set characteristics

job set	number of jobs	average requested nodes	average actual runtime [sec.]	average estimated runtime [sec.]	average inter-arrival time [sec.]
`set1`	8469	13.49	2835.83	6697.88	628.19
`set2`	8166	10.11	6697.88	21634.00	1277.34

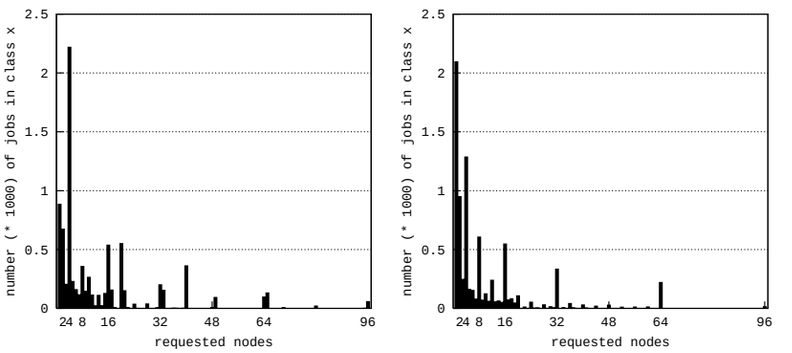

Fig. 1. Distribution of job width: number of jobs for each job width (requested nodes). `set1` left, `set2` right.

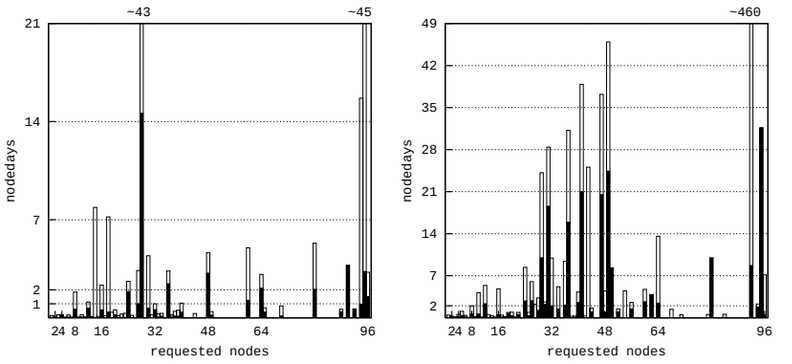

Fig. 2. Nodedays for actual (black boxes) and estimated (white boxes) runtimes. `set1` left, `set2` right.

4 Performance Metrics

For evaluating the simulated scheduling strategies and comparing them with each other, we basically focus on two metrics: the *average response time* and the *utilization* of the whole cluster. Herein the average response time is user centric whereas the utilization is owner centric. Note, that both numbers are contrary.

With these numbers we plot diagrams showing the utilization on the x-axis and the average response time on the y-axis. Combined with the shrinking factor we get several data points for each strategy and plot a curve. As the objective is to achieve high utilization combined with low average response times, curves have to converge to the x-axis.

Using

$t_j.submitT$	for the submit time of job j
$t_j.startT$	for the start time of job j
$t_j.endT$	for the end time of job j
n_j	for the occupied number of nodes of job j
J	for the total number of jobs
N	for the total number of available nodes
$firstSubmitT$	for the time the first submit event occurred ($\min_J t_j.submitT$)
$lastEndT$	for the time the last end event occurred ($\max_J t_j.endT$)

We define the average response time:

$$ART = \frac{1}{J} * \sum_{J} (t_j.endT - t_j.submitT)$$

and the utilization of the cluster:

$$UTIL = \frac{\sum_J (n_j * (t_j.endT - t_j.startT))}{N * (lastEndT - firstSubmitT)}$$

5 The dynP Scheduler

A first look at the performance figures of SJF, FCFS, and LJF shows that each scheduling strategy achieves an optimal performance for a specific job set and/or range of utilization (see [2] for details, or Fig. 3 and Fig. 4). That lead us to the idea of the dynP scheduler (for **dyn**amic **P**olicy), a scheduler that combines the three strategies and dynamically changes between them online. Now the problem is to decide when to change the sorting policy. We use the average estimated runtime of jobs currently in the waiting queue as criterion, because: 1) the performance of each strategy strongly depends on the characteristics of the job set (i.e. jobs in the waiting queue), and 2) SJF and LJF sort the queue according to the estimated runtime.

A similar approach was introduced in the IVS (Implicit Voting System) scheduler [6,8]. The IVS Scheduler uses the utilization of the machine and the ratio of batch and interactive jobs to decide which policy to take. The decider of IVS works as follows:

```
CASE
    (MPP is not utilized) --> switch to FCFS;
```

```
    (# interactive jobs > (# batch jobs + delta)) --> switch to SJF;
    (# batch jobs > (# interactive jobs + delta)) --> switch to LJF;
END;
```

We use two bounds to decide when the policy should be changed. If the average estimated runtime in the queue is smaller than the *lower_bound* the policy is changed to SJF. LJF is used when the average estimated runtime is greater than the *upper_bound*. A change of policy also necessitates a reordering of the waiting queue, as always the head of the queue is processed. If the reordering is omitted, certainly the wrong job according to the new policy is scheduled. The decider of the 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;
    } ELSE IF (lower_bound < AERT <= upper_bound) {
            switch to FCFS;
    } ELSE IF (upper_bound < AERT) {
            switch to LJF;
    }
    reorder waiting queue according to new policy;
}
```

We use a threshold of 5 to decide whether or not a check for policy change is done. With that we prevent the scheduler from unnecessary policy changes.

6 Performance of the dynP Scheduler

Obviously the performance of the dynP scheduler strongly depends on the parameter settings. We simulated various combinations of lower and upper bound settings [2]. After a first evaluation we restricted the parameters for a second run. We found out that the combination of 7200 s (2h) for the lower bound and 9000 s (2h:30min) for the upper bound is superior to all other combinations.

Table 2. Lower and upper bounds used in a first (top) and second (bottom) run.

lower	600, 1200, 1800, 3600
upper	3600, 7200, 14400, 21600, 43200

lower	6600, 7200, 7800
upper	8400, 9000, 9600

Note, if lower and upper bound are equal, FCFS is not used at all. Then the decider switches only between SJF and LJF. In a real environment the parameters for lower and upper bound have to be set by the system administrator based on a profound knowledge. This fine-tuning is necessary to get the best performance out of the dynP scheduler and its environment (machine configuration, user behavior).

As the dynP scheduler has been developed to be superior to FCFS, SJF, and LJF independent from the job set, we have to compare the performance to these strategies.

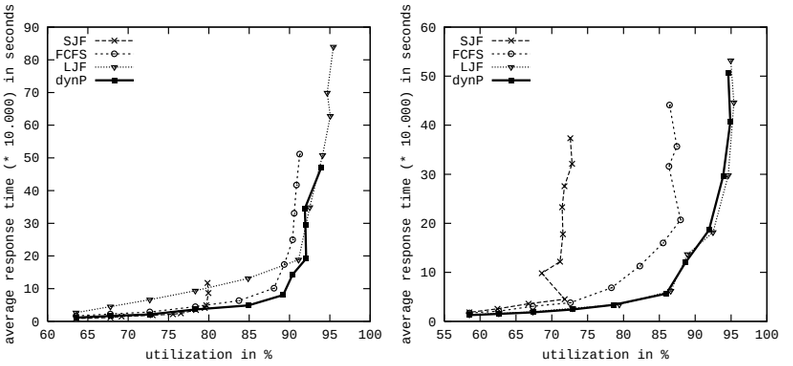

Fig. 3. Performance (average response time) focusing on maximal utilization in saturated state for the dynP scheduler with lower bound 7200 s and upper bound 9000 s. `set1` left, `set2` right.

When looking at the maximum utilization achieved in the saturated state dynP achieves similar results as LJF (∼92% for `set1` and ∼94% for `set2`). Especially in the diagram for `set1` the dynamic character of dynP is obvious. At low utilizations the performance (i.e. average response time) is as good as SJF. With increasing utilization the dynP curve follows FCFS and is even slightly better, even though LJF is rarely selected.

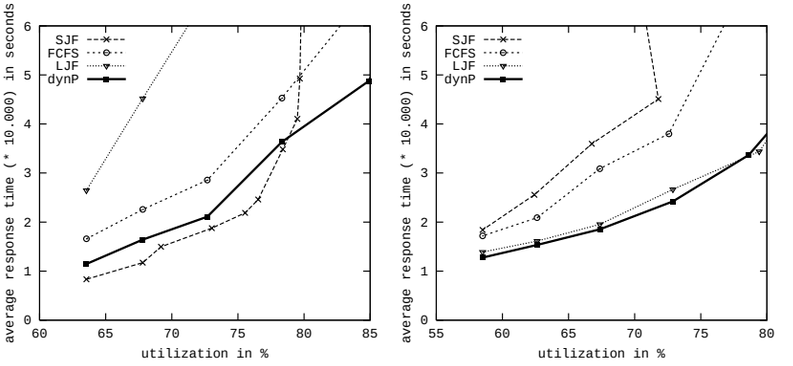

Fig. 4. Performance (average response time) focusing on medium utilizations for the dynP scheduler with lower bound 7200 s and upper bound 9000 s. `set1` left, `set2` right.

Focusing again on medium utilizations (Fig. 4) the performance of dynP is almost as good as SJF for `set1` and LJF for `set2`. In `set2` the performance can even be increased by about 8% (see Tab. 4).

Table 3. Average response times for SJF, FCFS, and LJF at medium utilizations. FCFS is used as reference for computing the percentages. `set1` left, `set2` right.

	SJF	FCFS	LJF
∼63%	8335 s	16587 s	26459 s
	(+49%)	(0%)	(-60%)
∼78%	34840 s	45292 s	93641 s
	(+23%)	(0%)	(-106%)

	SJF	FCFS	LJF
∼58%	18388 s	17182 s	13883 s
	(-7%)	(0%)	(+19%)
∼73%	45080 s	37998 s	26673 s
	(-19%)	(0%)	(+30%)

Table 4. Comparing average response times for dynP with a lower bound of 7200 s and an upper bound 9000 s at medium utilizations. As reference for computing the percentages SJF for `set1` and LJF for `set2` is used (best in each case).

set1	SJF	dynP_7200_9000
∼63%	8335 s	11454 s
	(0%)	(-37%)
∼78%	34840 s	36324 s
	(0%)	(-4%)

set2	LJF	dynP_7200_9000
∼58%	13883 s	12769 s
	(0%)	(+8%)
∼73%	26673 s	24214 s
	(0%)	(+9%)

As the dynP scheduler dynamically changes its policy, Fig. 5 shows how long each policy is used. The utilization is again plotted on the x-axis. The time each policy is used (in percent) is shown on the y-axis. Note, that the percentages are accumulated. Therefore the percentage for SJF is found between the x-axis and the first curve, the gap between the first and second curve represents the percentage of FCFS, and finally the gap between the second curve and the 100% percentage line represents LJF. For `set1` the dynP scheduler utilizes the LJF policy only for 15% and less, as the pure LJF scheduler achieves only poor results (cf. Fig. 3). On the other hand FCFS and SJF are used more often. FCFS starts at about 30% for low utilizations and ends up at ∼80%. At the same time the usage of SJF drops from ∼56% to 19%. Obviously a different

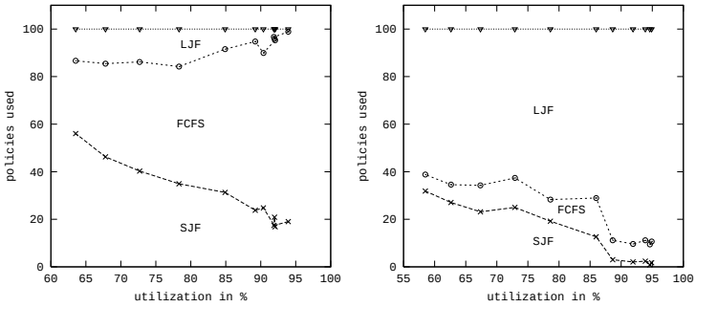

Fig. 5. Accumulated percentages of policies used by the dynP scheduler with lower bound 7200 s and upper bound 9000 s. `set1` left, `set2` right.

characteristic can be seen for `set2`. LJF dominates from the beginning, starting at 61% and ending at slightly over 90%. FCFS and SJF are less used with increasing utilization (FCFS: between 8 and 16%, SJF: from 31% down to 1%).

Based on its adaptability the dynP scheduler achieves a good performance regardless of the used job sets, i.e. the jobs in the waiting queue in a real environment. Of course the performance strongly depends on the parameter settings for the lower and upper bound. The complete set of simulated combinations (cf. 1st run in Tab. 2) shows that if awkward pairs of lower and upper bounds are chosen, the performance can drop significantly.

7 Conclusion

In this paper we compared space-sharing strategies for job-scheduling on HPC-clusters with a scheduling simulator. Therein we focused on FCFS, SJF, and LJF which are used in CCS (Computing Center Software). Each strategy was combined with conservative backfilling to improve its performance. To compare the results we used the average response time as an user centric, and the utilization of the whole cluster as an owner centric criterion.

As job input for the simulations we used two job sets that were extracted from trace data of the *hpcLine* cluster. To increase the workload for the scheduler we used a shrinking factor (0.75 down to 0.25 in steps of 0.05) to reduce the average interarrival time. With that a wide range of utilization (55 - 95%) was achieved. Each scheduling strategy reached a saturated state, in which a further increase of workload does not lead to a higher utilization, but only to higher average response times. LJF achieved the highest utilization for both job sets, with a peak at 95%. The performance at medium utilizations up to approx. 80% showed an unbalanced behavior. None of the three strategies was superior for both job sets. SJF performs best for `set1` with a performance gain of $\tilde{2}3\%$ at 78% utilization compared to FCFS. For the other job set at an equal utilization SJF was $\tilde{1}9\%$ worse (cf. Tab. 3). FCFS proved to be a good average. These anticipated results rest upon the different characteristics of the job sets combined with the basic ideas of the scheduling strategies.

Based on these results we presented the dynP scheduler, which changes its policies (FCFS, SJF, and LJF) online. As criterion when to change the policy, dynP uses the average estimated runtime of jobs currently in the queue. A lower and upper bound specifies when to change the policy. Obviously the performance of the dynP scheduler strongly depends on the chosen bounds. The best setting found for the two jobs sets were 7200 s (2h) for the lower and 9000 s (2h30m) for the upper bound. Compared to SJF for `set1` and LJF for `set2` the dynP scheduler performed only slightly worse (-4%) for `set1` and even better (+9%) with `set2` at medium utilizations. The maximum utilization achieved in the saturated state ($\sim$95%) is as good as with LJF for both job sets.

As already mentioned the performance of dynP depends on the proper settings of the bounds. To ease the usability of the dynP scheduler we are currently working on an adaptive version of dynP with does not require anymore startup

parameters. Self-tuning systems [5] might be used for this. In the future we will also add the dynP scheduler to CCS and use it for everyday work. This will show, if the simulated results from this paper can also be achieved in a real environment.

References

1. A. Mu'alem and D. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *IEEE Trans. Parallel & Distributed Systems 12(6),* pages 529–543, June 2001.
2. A. Streit. On Job Scheduling for HPC-Clusters and the dynP Scheduler. TR-001-01, PC2 - Paderborn Center for Parallel Computing, Paderborn University, July 2001.
3. D. 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. D. G. Feitelson and L. Rudolph. Metrics and Benchmarking for Parallel Job Scheduling. *Lecture Notes in Computer Science,* 1459:1–24, 1998.
5. D.G. Feitelson and M. Naaman. Self-Tuning Systems. In *IEEE Software 16(2),* pages 52–60, April/Mai 1999.
6. F. Ramme and T. Romke and K. Kremer. A Distributed Computing Center Software for the Efficient Use of Parallel Computer Systems. *Lecture Notes in Computer Science,* 797:129–136, 1994.
7. H. Franke and J. Jann and J. Moreira and 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.
8. J. Gehring and F. Ramme. Architecture-Independent Request-Scheduling with Tight Waiting-Time Estimations. *Lecture Notes in Computer Science,* 1162:65–80, 1996.
9. A. Keller, M. Brune, and A. Reinefeld. Resource Management for High-Performance PC Clusters. *Lecture Notes in Computer Science,* 1593:270–281, 1999.
10. 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.
11. D. A. Lifka. The ANL/IBM SP Scheduling System. *Lecture Notes in Computer Science,* 949:295–303, 1995.
12. J.E. Moreira, H. Franke, W. Chan, and L. L. Fong. A Gang-Scheduling System for ASCI Blue-Pacific. *Lecture Notes in Computer Science,* 1593, 1999.
13. J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY – LoadLeveler API Project. *Lecture Notes in Computer Science,* 1162:41–47, 1996.
14. 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.