On the Combined Scheduling of Malleable and Rigid Jobs

Jan Hungershöfer
Paderborn Center for Parallel Computing
Fürstenallee 11, 33102 Paderborn, Germany
E-mail: hunger@upb.de

Abstract

The demand of the users of parallel systems for low response times contradicts the ambition of the system maintainers for a high utilization. A high utilization normally results in long waiting times for the users' jobs. To fullfil the concerns of both interest groups is a hard job to do. The usage of more flexible jobs models can be a way out of the dilemma. These models allow jobs to change their width at application start (moldable jobs) or even during execution (malleable jobs). We have analyzed the quality of schedules using job sets with moldable and malleable jobs and combinations of both. Tracefiles from supercomputer installations have been modified to contain varying fractions of moldable and malleable jobs. Using a special simulation environment for the more flexible job models the jobs have been scheduled virtually. The results show that both interest groups mentioned above can be pleased if these job models are used and the average response times become significantly better.

1. Introduction

In recent years a lot of work has been done on the research of parallel algorithms and systems and strategies for the efficient scheduling of parallel jobs on high performance computing systems. This has undoubtedly brought advancements in the scheduling of parallel jobs, e. g., shorter response times or more efficient system usage. An aspect which has been much less in focus of research is the benefit in scheduling if the jobs are more flexible in terms of requested resources. In this paper we present results of a comparison on scheduling of standard parallel jobs and flexible ones which can adapt their resource consumption to the current system load.

In the following we use a typical classification of jobs concerning their adaptiveness [8]. The class of *rigid* jobs builds the simplest class. They require a fixed number of resources, i. e., nodes, processors, or threads. If the number of vacant processors is smaller than their demand, they cannot be started. When the number of resources is adjustable at program start, the job is called *moldable*. A moldable job can run with any number of processors, within a certain range, and leaves the decision to the system, i. e., to the scheduler. During run time the number of resources cannot be changed. Jobs which have the flexibility to change the number of assigned resources during run time on system demand are called *malleable*. This kind of scheduling is also called *dynamic partitioning* [10]. If the request is initiated from the application itself, e. g., due to alternating sequential and parallel program phases, the job is denominated as *evolving*.

The evaluation of the combined scheduling of rigid, moldable, and malleable jobs and the resulting advantage in scheduling quality is the main objective of this paper. We have examined the scheduling of parallel jobs of the different classes, individual and in combination, and analyzed the quality of the resulting schedules.

Some work has been done in scheduling of moldable jobs. In [4, 6] is shown that an automatic scheduler can reduce the turn-around time of moldable jobs. It selects the best suited of a list of possible requests provided by the user. The state of the supercomputer is analyzed and the best suited request is selected. In [3] and [7] the authors present approximation algorithms for scheduling a set of independent moldable tasks.

The analysis of a survey among users of supercomputer centers shows that most jobs are moldable already [5]. Though malleable applications are still quite rare, the effort of turning a moldable application into a malleable one is often not too strong. In [13] an existing finite-element simulation program is being extended to malleability. Additionally, an approach is being presented for managing different malleable applications on a shared memory system. A comparison of a round-robin, equipartitioning, and dynamic policy shows the

Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)
1550-6533/04 $ 20.00 IEEE

trace	nodes/ CPUs	system	# jobs	recorded time range	source
CTC	512	IBM SP2	79 302	Jul 1996 – May 1997	Cornell Theory Center
KTH	100	IBM SP2	28 490	Oct 1996 – Aug 1997	Swedish Royal Institute of Technology
SDSC	128	IBM SP2	67 667	May 1998 – Apr 2000	San Diego Supercomputer Center
CHPC	266	cluster	20 000	Mar 2000 – Mar 2001	Center for High Performance Computing
PC2	96	cluster	35 094	Jan 2001 – Dec 2001	Paderborn Center for Parallel Computing

Table 1. Main characteristics of the tracefiles.

benefit of dynamic partitioning [16]. Whereas McCann et al. use synthetic job sets, measurements in this paper are based on real tracefiles.

The remainder of this paper is organized as follows. In the next section, we give some comments about the practical aspects using moldable and malleable jobs. In Sect. 3 we describe the simulation environment briefly. Afterwards, the input data sets used are presented. Also, the properties of the basic schedules are given. The results we obtain with it are shown in Sect. 6. A short conclusion closes the paper.

2. Availability of Moldable and Malleable Jobs

A high percentage of parallel jobs is already moldable. The popular and quasi-standard parallel programming paradigms like PVM, MPI, or threads support the selection of the number of processors at program start. Some of them, e. g., MPI-2, also allow to change the number of processors during run time, i. e. facilitate malleability. Some parallelization models are well suited for malleability, e. g., the farmer-worker concept permits easy addition or reduction of resources.

We have augmented a parallel finite element code into a malleable application. Details about the extension and how multiple instances are handled efficiently within a multi-user environment can be found in [12, 13]. Our practical experience shows that the extra work for development of malleable applications is often reasonable low and the higher flexibility pays off the effort.

3. The Simulation Environment

The evaluation has been made within an event driven simulation environment. For an input tracefile of submitted jobs a complete schedule is calculated and analyzed afterwards. As in todays environments and therefore in existing tracefiles only rigid jobs are included, the simulation environment has to convert a specified percentage of jobs into moldable or malleable ones. The percentage of moldable and malleable jobs is variable and can be configured while the jobs to be moldable or malleable are chosen randomly. To eliminate the influence of the system's random number generator, all runs are repeated several times and the results are averaged. For all simulations the scheduling policy first come first serve (FCFS) with EASY backfilling is used [15, 17].

4. The Input

The focus of our work is the examination of moldable and malleable jobs within schedules of planning based schedulers. Planning systems – in contrast to *queuing* systems – do the resource planning for the present and future, all requests get start times assigned [11]. Therefore, planning systems require run time estimates for each job to build their schedules. The input for the simulations are job sets where each job has a submission time, a desired width, i. e. the number of processors, a run time estimation, and a real run time. The job description is augmented by the job's class since the traces contain only rigid jobs.

We have decided to use real tracefiles from supercomputer installations as input data. Only a few tracefiles which fulfill the requirements mentioned above can be found on the Internet. The traces CTC, KTH, and SDSC are taken from the Parallel Workloads Archive [9], whereas the CHPC trace is downloaded from the Maui Scheduler Workload Trace Repository [2]. The PC2 trace has been recorded by CCS, the scheduling environment for the systems located at our institute [1]. Tab. 1 compares the main characteristics of the traces.

At first, all tracefiles have been scheduled unmodified, i. e., all jobs are rigid. Table 2 shows the performance of the basic schedules. The metrics used are makespan (MS), average waiting time (AWT), average response time (ART), slowdown weighted by job area (SLDwA), and the system utilization (UTIL). We use the following notation for the parameters of job i:

trace	AWT s	ART s	SLDwA	UTIL %
CTC	4829	15787	2.05	65.70
KTH	7992	16849	3.10	68.72
SDSC	21060	27137	6.83	82.48
CHPC	3	47841	1.00	38.39
PC2	216	4562	1.08	46.07

Table 2. Performance of the basic schedules.

t_i^a is the arrival (submit) time,
t_i^s is the start time,
t_i^e is the end (completion) time, and
w_i is the job width (number of processors or nodes).

With that the following parameters can be determined:

$l_i = t_i^e - t_i^s$ the length (run time, duration) of job i,
$t_i^w = t_i^s - t_i^a$ the waiting time,
$t_i^r = t_i^e - t_i^a$ the response time,
$s_i = \frac{t_i^r}{l_i}$ the slowdown, and
$a_i = l_i \cdot w_i$ the area of allocated resources.

With m being the total number of jobs and N the number of resources, we use the following definitions:

$MS = \max_i t_i^e - \min_i t_i^a$ the makespan, i. e., the last completion time of a job,

$AWT = \frac{1}{m} \cdot \sum_i t_i^w$ the average waiting time of all jobs,

$ART = \frac{1}{m} \cdot \sum_i t_i^r$ the average response time of all jobs,

$SLDwA = \frac{\sum_i a_i \cdot s_i}{\sum_i a_i}$ the slowdown weighted by job area, and

$UTIL = \frac{\frac{1}{N} \cdot \sum_i a_i}{\max_i t_i^e - \min_i t_i^a}$ the utilization.

5. Scheduling of moldable and malleable Jobs

The standard FCFS scheduler with backfilling has been augmented for the scheduling of moldable and malleable jobs. Still, all jobs are scheduled using the first come first serve policy. When the scheduler has to insert a moldable job into the schedule it reshapes the job to fit into the gaps and selects the alternative which minimizes its completion time. In the example of Fig. 1 the moldable job (chequered box) has to be scheduled. The scheduler decides for the left solution. The alternative placement (middle) is rejected since it leads to a later completion time. The right example in Fig. 1 shows the insertion of a malleable job. The malleable job (checkered area) is placed into the schedule like a moldable one, i. e., optimizing the expected completion time. During the run time it expands if resources are vacant and reshrinks if other jobs show up. The number of processors allocated at start is set to be the minimum the job can be reduced to (dashed lines in Fig. 1). This prevents malleable jobs to be ruled out by jobs of other classes.

If there are several malleable jobs running concurrently, the available processors are distributed evenly among them. This policy, so called *equipartitioning*, is easy to implement since no further information about the job characteristic is necessary [19]. And it is robust with respect to the response times for a variety of workloads [14].

The completion time of a malleable job cannot be calculated in advance due to the changing job width. Therefore, the job's remaining work is decremented until it reaches zero. The variable is initialized with the job's area and reduced by the current number of assigned processors in every step of the simulation. From the scheduler's point of view running malleable jobs are handled like moldable jobs. The minimum number of processors is reserved by the scheduler, additional processors of malleable jobs are treated as vacant for scheduling. For simplicity, the area of a moldable or malleable job is kept constant when it is being reshaped, e. g., doubling the number of processors halves the run time. Likewise, no overhead for repartitioning a malleable job is being accounted in our standard scenario.

6. Evaluation

For the examination of the influence of the presence of moldable and malleable jobs the original input tracefiles have been modified. The fraction of moldable or malleable jobs has been varied between 0 % and 100 %, the remainder stays rigid. Also the combination of moldable and malleable jobs in one tracefile has been studied. Then, 10 %, 25 %, 33 %, or 40 % have been selected to be moldable and the same amount to be malleable. Therefore, between 20 % and 80 % of the jobs are non-rigid in the scenarios.

As another parameter the interarrival times of the jobs have been modified. They have been shortened by up to 40 % of the original values. Since the jobs' areas

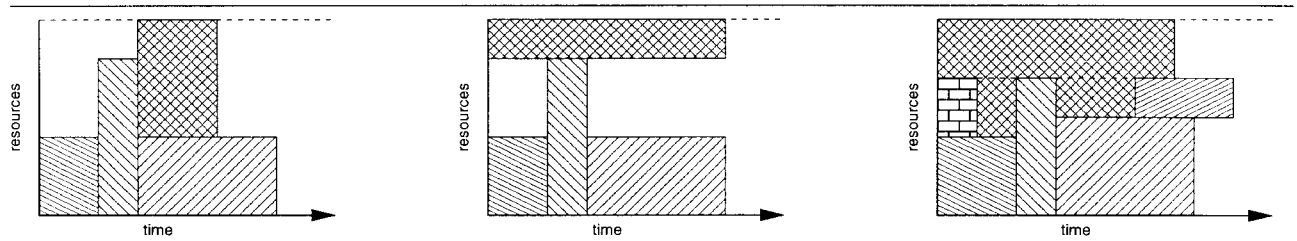

Figure 1. Scheduling of a moldable job (left, middle) and of a malleable job (right).

are unchanged, this results in a higher workload for the scheduler. We call the factor applied *shrinking factor* [18].

For a general comparison of all tracefiles with rigid jobs, the slowdown weighted by the area of the jobs is given in Fig. 2. On the x-axis the shrinking factor is shown. The unmodified traces have a shrinking factor of 1.0 and the slowdown is close to the minimum of 1.0. With decreasing shrinking factor the workload of the scheduler increases. The weighted slowdown increases differently for the traces. Whereas the CTC, CHPC, and PC2 trace hardly show any influence, the SDSC job set reacts with a strong raise. Looking at Fig. 2 we find a similar behavior. The diagram shows the average response time, given in seconds, of the same tracefiles over the shrinking factor. The initial response time is below 50 000 seconds for all traces. With a shrinking factor of 0.60 it stays below 100 000 seconds for the CTC, CHPC, and PC2 traces. The KTH and SDSC job sets raise it over 850 000 seconds.

In Fig. 3 the average job width and the average response time of the tracefiles are shown. The left column of diagrams shows the average job width and in the right column the response times are given. In all graphs the x-axis shows the percentage of moldable or malleable jobs, respectively. The red (upper) line shows the results for the moldable jobs, the green (lower) line for the malleable runs. The third line (blue) is the result of the combined runs where rigid, moldable, and malleable jobs run concurrently. The shrinking factors have been adjusted that the utilization is about 90 % for all traces. For the malleable jobs the average of the width has been calculated weighted by the duration.

In general, the average job width increases as the percentage of non-rigid jobs gains. The average width of the original job set is 7.66 for the KTH trace. With moldable jobs only it is 80.99, with the same amount of malleable jobs it is much smaller, only 9.58. For the mixed case the average job width is 21.96 with 20 % rigid jobs left. Since the maximum width of a moldable and malleable jobs is not limited, except by the system width, moldable jobs tend to allocate the whole system as the jobs starts to minimize the run time. When

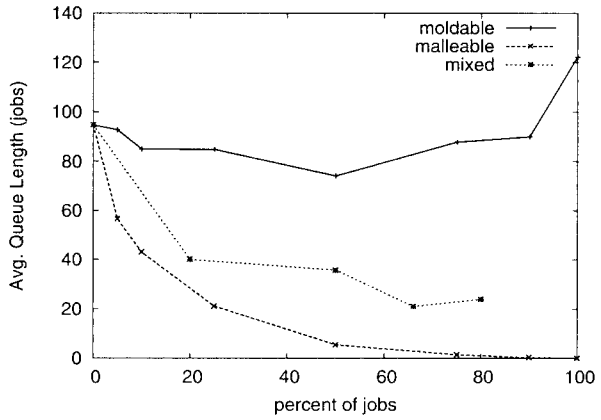

Figure 4. Average queue length of SDSC trace.

a high fraction of malleable jobs is running the processors are (re-)partitioned equally among the jobs which leads to more slender jobs. The width diagrams for the other traces show similar behavior.

The right plots of Fig. 3 show the average response time for the scenarios. For the KTH trace the initial, i. e. fully rigid, value is 101 381 seconds. With increasing amount of moldable or malleable jobs the response time decreases. It reaches its mininum at 11 046 seconds with 100 % malleable jobs. The best value when using moldable jobs is 49 970 seconds at 50 %. More moldable jobs increase the value to slightly over 70 000 seconds. The results of the combined runs are between the moldable and malleable ones. Nearly all job sets behave similar, except for the PC2 trace. When moldable and rigid jobs are scheduled we obtain a response time of 93 113 seconds which is nearly the same as the fully rigid run with 94 858 seconds. For the CHPC trace we find a peak in the graph for the moldable jobs traces which even exceeds the rigid value by about 11 %.

Figure 4 shows the average queue length of the scheduler when a job is being submitted. The values for the SDSC tracefile shown are exemplary. The queue length of the original trace is about 95 jobs. It increases

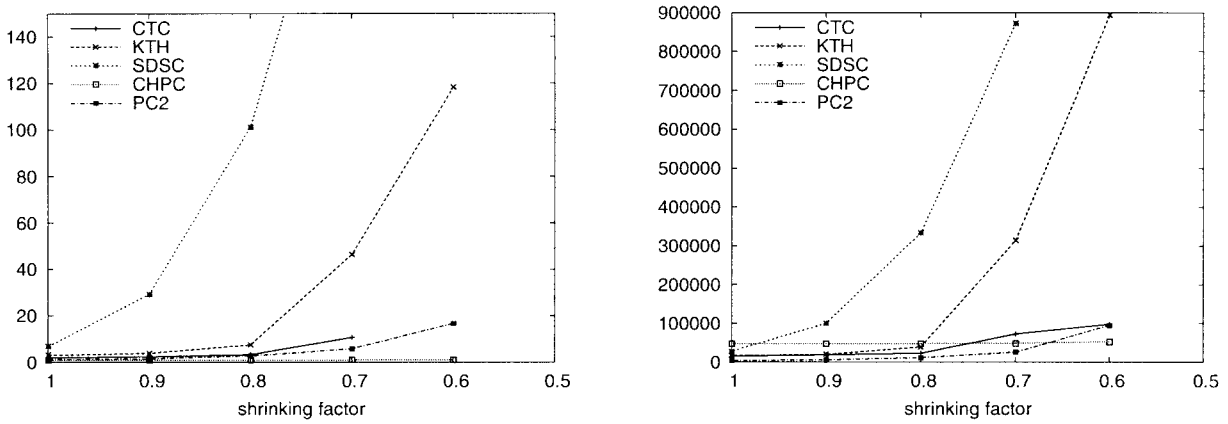

Figure 2. Slowdown weighted by job area (left) and average response time of all traces.

up to 122 jobs using the moldable job model, whereas it reaches zero when only malleable jobs are scheduled. The strong increase when going from 75% to 100% moldable jobs gives a reason for the higher response times. The longer queue lengths cause longer waiting times and therefore higher response times.

The average values of Fig. 3 are calculated over all jobs, the rigid and the non-rigid. Whereas, in Fig. 5 the graphs are drawn separately for the rigid and moldable jobs and rigid and malleable jobs, respectively, for the KTH tracefile and a shrinking factor of 0.75. In the left graph of Fig. 6 the average response time of moldable and rigid jobs is shown. The initial response time is 101 381 seconds. With only 5% moldable jobs the response time falls about 57% down to 43 558 seconds, it increases to 47 133 seconds while the fraction of moldable jobs is raising up to 75%. For the rigid jobs the corresponding value is 65 085 seconds, which is about 65% of the original value and therefore still better than before. When malleable jobs are used instead, as shown in the right graph of Fig. 6, the effect is even more dramatic. Starting with the same values, the response time for malleable jobs drops down to 33 431 seconds with a fraction of 5% of the jobs being non-rigid, which is only about a third of the initial value. If more malleable jobs are scheduled it decreases even further. The minimum of 13 472 seconds is reached with half of the job being malleable. Nearly synchronously the response time of the rigid jobs is getting shorter, it goes down to 28 683 seconds when rigid jobs are half of the jobs. The presence of flexible jobs in the schedule gives shorter response times for all jobs, not only for the non-rigid ones. Especially if malleable jobs are scheduled, the benefit is obvious.

If the fraction of moldable or malleable jobs is fixed to 75% and the shrinking factor is varied we obtain

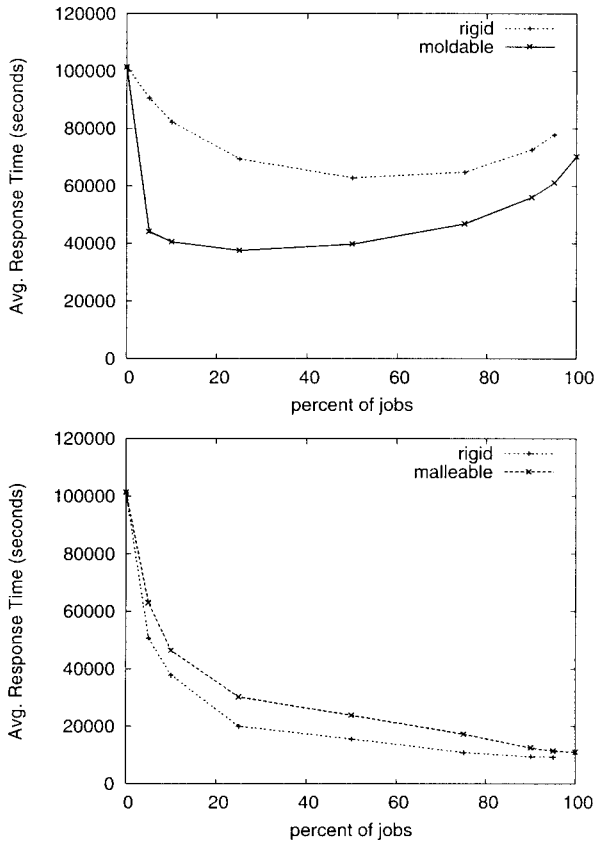

Figure 5. Average response time of moldable (left) and malleable jobs.

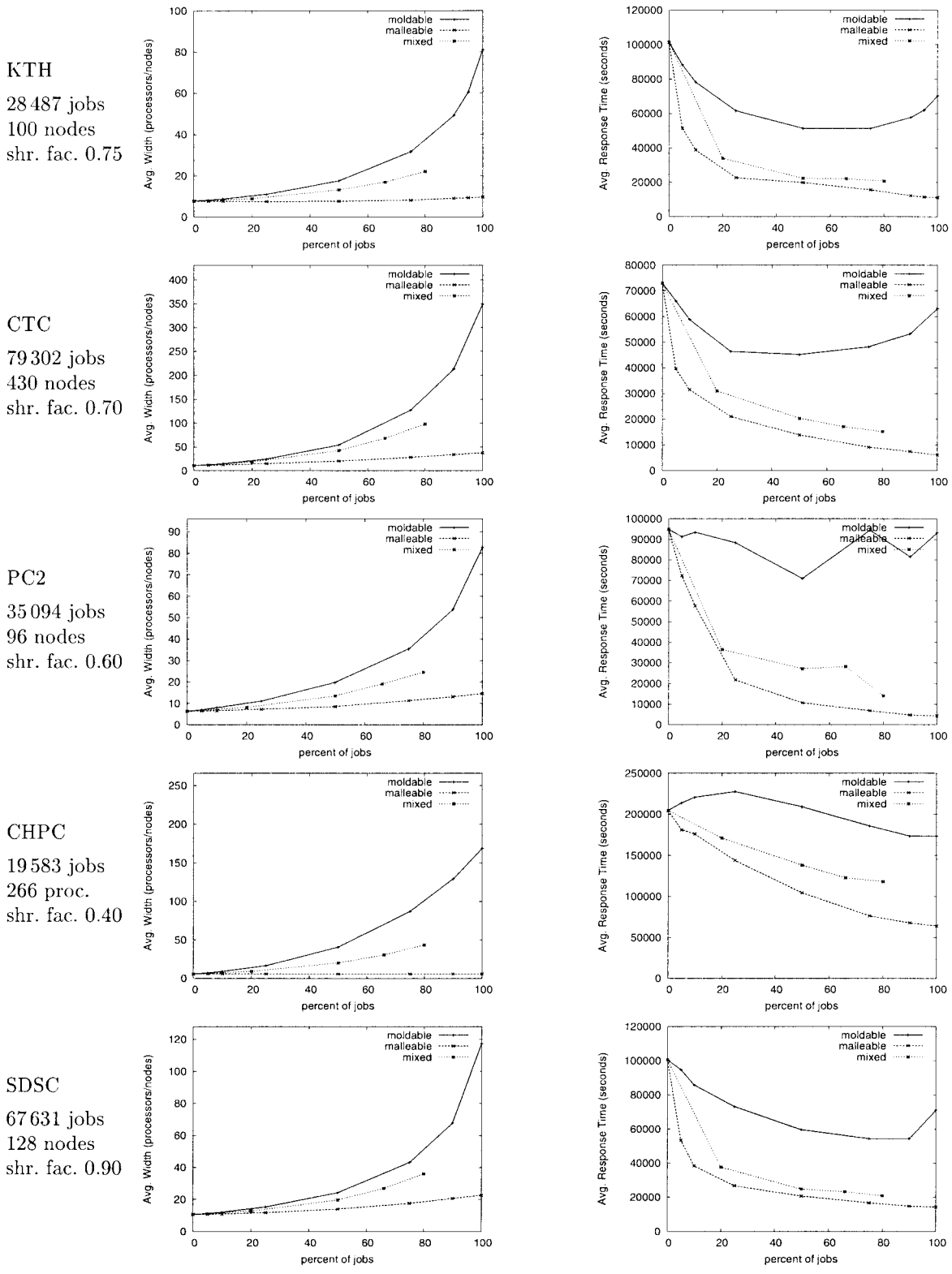

Figure 3. Average job width (left) and average response time (right) of tracefiles.

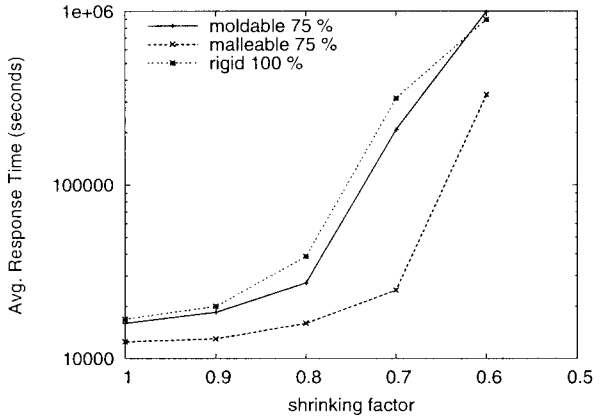

Figure 6. Average response time of KTH tracefile.

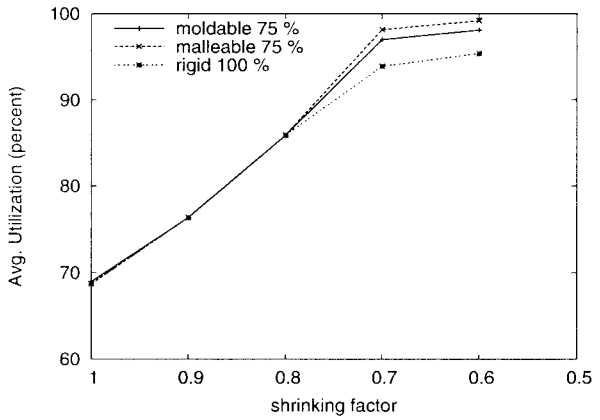

Figure 7. Utilization of KTH tracefile.

the graph of Fig. 6, for example. It shows the average response time for the KTH job set plotted over the shrinking factor on the x-axis. The response time increases with decreasing shrinking factor. Using malleable jobs the response time is about 330 000 seconds at a shrinking factor of 0.6, whereas both other runs, moldable and rigid, raise to more than 890 000 seconds.

Slowdown and response time are user centric metrics whereas the system utilization is an owner centric measure which focuses on the efficient usage of the computing system. The plot in Fig. 7 shows the system utilization for the KTH job set. There are three lines in the graph: for 75 % of moldable or malleable jobs and for rigid-only runs, respectively. The original utilization is about 68 %. It increases with the scheduler's workload as the shrinking factor gets smaller and approaches a saturated state of nearly 100 % at a shrinking factor of 0.6. Again, the simulation run with the malleable jobs outperforms the utilization of the moldable jobs.

From the two diagrams in Figures 6 and 7 we can read that using malleable jobs we reach the same average response time with a higher shrinking factor, than with rigid-only schedules. Concurrently, we get a higher system utilization. While the first fact is of main interest for the system user, the latter one causes a higher contentment of the system maintainers.

6.1. Reconfiguration Penalty

Besides the standard scenario measurements with a reconfiguration penalty for the malleable jobs have been made. For each reconfiguration a penalty of x seconds extra run time has been applied. The value of x has been varied between 1 and 20 seconds. If we take the KTH trace for example and set the penalty to 20 seconds, the relative amount of applied penalty is below 2 % of the total job area for the analyzed tracefiles. The average response time is about 6 % larger for the malleable jobs compared to zero penalty. The number of resource changes per job is about 11 in average and about 1000 at maximum.

Compared with the run time of the jobs the accumulated penalty times are still quite small. Therefore, we conclude that the decision to omit this parameter for the standard measurements was right.

7. Conclusion

In this paper we have presented first results on the influence of job flexibility on the quality of schedules, based on the FCFS policy. We compared traces of moldable, malleable, and mixed job sets with standard, rigid jobs. A set of real tracefiles from supercomputer installations has been modified and virtually scheduled within a simulation tool. The resulting schedules have been analyzed statistically.

The job sets containing a reasonable number of moldable or malleable jobs perform better with respect to the metrics average response time and utilization. Due to their higher flexibility moldable and malleable jobs can provide significantly shorter response times. If only a fraction, e. g., 25 %, of all jobs is non-rigid the overall response time, including the rigid jobs, can be reduced about 50 %. The benefit of using this kind of flexible jobs is immediate for both, the users and the system maintainer.

Summarizing the results shown above we propose the usage of flexible job models, especially malleable jobs. Our experiments show that much better schedules can be found by utilizing the higher flexibility of malleable jobs. We are of the opinion that the advantages in many cases pay off the higher effort to write malleable parallel application.

References

[1] Computing center software (CCS), Apr. 2004. http://www.upb.de/pc2/projects/ccs/.

[2] HPC workload/resource trace repository, Apr. 2004. http://supercluster.org/research/traces/.

[3] J. Błażewicz, M. Machowiak, G. Mounié, and D. Trystram. Approximation algorithms for scheduling independent malleable tasks. In R. Sakellariou, editor, *Proc. of Euro-Par 2001*, volume 2150 of *Lecture Notes in Computer Science*, pages 191–197. Springer, Aug. 2001.

[4] W. Cirne. *Using Moldability to Improve the Performance of Supercomputer Jobs.* PhD thesis, University of California, San Diego, Feb. 2001.

[5] W. Cirne and F. Berman. A model for moldable supercomputer jobs. In *Proc. of 15th Intl. Parallel & Distributed Processing Symp*, 2001.

[6] W. Cirne and F. Berman. Using moldability to improve the performance of supercomputer jobs, 2001.

[7] P.-F. Dutot and D. Trystram. Scheduling on hierarchical clusters using malleable tasks. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 199–208. ACM Press, 2001.

[8] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems. IBM Research Report, Hebrew University, Aug. 1997. Second Revision (1997).

[9] D. G. Feitelson. Parallel workloads archive, Apr. 2004. http://www.cs.huji.ac.il/labs/parallel/workload/.

[10] D. G. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, and P. Wong. *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, chapter Theory and Practice in Parallel Job Scheduling, pages 1–34. Springer, 1997.

[11] 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*, pages 1–20. Springer, 2003.

[12] J. Hungershöfer, J.-M. Wierum, and H.-P. Gänser. Efficient shared memory parallelisation and resource management of explicit finite element codes. Technical Report TR-001-03, Paderborn Center for Parallel Computing, Dec. 2003.

[13] J. Hungershöfer, J.-M. Wierum, and H.-P. Gänser. Resource management for finite element codes on shared memory systems. In M. L. Gavrilova, V. Kumar, P. L'Ecuyer, and C. J. K. Tan, editors, *Proc. of Intl. Conf. on Computational Science and Its Applications (ICCSA)*, volume 2667 of *Lecture Notes in Computer Science*, pages 927–936. Springer, May 2003.

[14] S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proc. ACM SIGMETRICS Conf. on Measurement and Modelling of Computer Systems*, pages 226–236, 1990.

[15] D. A. Lifka. The ANL/IBMSP scheduling system. In D. G. Feitelson and L. Rudolph, editors, *Proc. of 1st Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 295–303. Springer, 1995.

[16] C. McCann, R. Vaswani, and J. Zarhojan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 11(2):146–178, May 1993.

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

[18] A. Streit. *Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids.* PhD thesis, Faculty of Computer Science, Electrical Engineering and Mathematics, Paderborn University, 2003.

[19] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the twelfth ACM Symposium on Operating systems principles*, pages 159–166. ACM Press, 1989.