Adaptive Scheduling for QoS Virtual Machines under Different Resource Availability—First Experiences

Angela Sodan

University of Windsor, Windsor ON N9B 3P4, Canada
acsodan@uwindsor.ca

Abstract. The current trend in CPU design is toward many-core CPUs which will create SMP servers with large numbers of cores and require jobs to be parallel. To provide performance guarantees (QoS) and support functional separation, virtual machines are an important approach in such environments. Very promising for job scheduling in such environments is adaptive scheduling which can adjust sizes of parallel jobs to consider different load situations and different resource availability. If applied to virtual machines and resource partitioning on many-core-CPU / multiple CPU servers, the resource allocation can be determined at virtual-machine level and propagated down to the job sizes. The paper investigates job re-shaping and virtual-machine resizing, the effects which the efficiency curve of the jobs has on performance, and the potential for predictability of performance under different resource allocation.

Keywords: adaptive job scheduling, molding, prediction, utilization

1 Introduction

Multi-core CPUs have become the current approach in CPU design to reduce power consumption and to continue performance growth of CPUs in spite of physical limits imposed on the performance improvement of individual cores [16]. The trend is likely toward many-core CPUs with many more than just 2 or 4 cores per CPU. To exploit the performance of many-core CPUs per individual program, parallelization of applications will be a must [23]. (System) virtual machines are already now an important approach to safely share servers among different applications or user groups, and are likely to become more important on many-core servers with large numbers of cores. Virtual machines provide functional separation (with potentially even different operating systems) but also a framework for performance guarantees (QoS) if controlling resource allocation among virtual machines.

If resource allocation per virtual machine changes over time, this has implications on suitable sizes of parallel jobs. Adaptive job scheduling which can adjust job sizes according to the current load and resource availability is already well investigated in the literature and will likely become an important approach for scheduling on many-core/virtual-machine servers. Adaptive job scheduling may mean deciding the job size at start time (molding jobs) or adjusting it at runtime (malleable jobs). Though the latter is more flexible it requires special support in the runtime system of the application, whereas start-time adaptation was found to be applicable to the majority of jobs [3]. In this paper, we therefore only consider size adaptation at job start-time.

The presented work studies the effects of changing the (virtual) machines size and the size allocation of the jobs, including the effects of the jobs' efficiency/scalability curve. The evaluations are mainly done by keeping the original scheduler unchanged and adjusting all job sizes equally, i.e. without looking into specific scheduling contexts. This permits investigation of effects independent of the scheduling algorithm, whereas previous research mixed efficiency and scheduler considerations. Thus, as one of the contributions, this paper looks separately into the benefits obtained from size changes under constant efficiency (work-conserving adaptation) and different levels of efficiency changes (non-work-conserving adaptation). The experiments were performed with 1) a standard scheduler and FCFS and priority policies and 2) with our Scojo-PECT preemptive scheduler [5][19][18] which can assign different priorities to different job type (currently defined on the basis of runtimes) via different time shares but is FCFS per job type. Though benefits may be expected from mere reduction in sizes by easier fitting of jobs and smart scheduling approaches, the results demonstrate that the change in efficiency and subsequently work load constitutes the dominating performance factor.

2 Related Work

Most adaptive approaches apply molding only. The approach of Cirne and Berman [3] molds jobs at the time of job submission, whereas later research showed performance improvements [22] by setting limits for the maximum job size in dependence on the current system load and on the job's size requests and by making decisions at job start time rather than submission time. Both approaches applied the Downey scalability model [4] which describes typical application scalability/efficiency curves and includes the possibility to model different scalability by modifying the corresponding scalability factor σ . In [22], the evaluations of the proposed adaptive scheduling algorithm were done with different σ values. However, in our work, we not only test different efficiency but separate efficiency effects from the scheduling algorithm. Most of these approaches exploit adaptation with the goal to adapt to varying system load. The approach by Naik et al. [12] also adapts resource allocation at runtime and attempts to schedule all jobs from the queue, though setting a limit for medium and long jobs to keep space for short jobs. Other approaches apply limits on job sizes in relation to machine size to keep space for future arrivals [8][14]. In addition to load adjustments, some approaches additionally try to reduce fragmentation in dependence on the specific resource/scheduling situation [13][21].

The two basic approaches to decide about the job sizes are resource-based partitioning and efficiency-based partitioning [6]. Resource-based partitioning typically comes in the form of EQUI partitioning which means assigning the same number of resources to each job. This approach yields suboptimal performance in the general case as it does not consider how well the jobs use the resources [2][11]. Efficiency-based partitioning exploits the efficiency characteristics of the applications and allocates more resources to jobs that make better use of them, which typically leads to the overall best results [2][11]. Similar to resource-based partitioning, efficiency-based partitioning may be applied in the form of providing equal efficiency to all jobs in the system (EQUI-EFF).

3 Work-Conserving and Non-Work-Conserving Job-Size Adaptation—Myths and Reality

3.1 Space Sharing and Scojo-PECT Time Sharing

For our experiments, we use a standard space-sharing scheduler which employs either a FCFS policy or priority scheduling. Priorities are based on runtime classes, and classes with shorter runtime receive higher priority. To avoid starvation, the implementation which is used here ages jobs to the next higher priority level if their wait time exceeds 10 times their runtime.

Scojo-PECT [5] employs preemption to support scheduling of shorter jobs in the presence of longer-running jobs. Scojo-PECT preempts jobs to swap space which is easy to support in the machine environment. This avoids the memory pressure which gang scheduling imposes and the hard-to-support checkpointing which is necessary for migration. However, Scojo-PECT subsequently imposes the constraint that preempted jobs are later restarted on the same resources. To make preemption to disk affordable and to avoid that jobs are delayed because of problems in getting access to their resources again, Scojo-PECT employs coarse-grain time slices that preempt all jobs. Jobs are sorted per job class/type, and slices associated with job types. The slice time for each job type is determined on the basis of typical job-type mixes and the administrator's policies and can be recalculated in regular time intervals. One slice for each type is scheduled per interval (since short jobs backfill into other slices in most cases, their slice is only scheduled if short jobs are waiting), and the slice times can be decided at the beginning of each interval. This permits controlling the resource allocation via different policies at different times of the day or via adaptive allocation which considers the current load of the machine [18]. In the context of this paper, the relative slice times of different job classes are kept static. Jobs per job type are scheduled in FCFS, and either EASY or conservative backfilling is applied. Since the separation of jobs into different types is likely to increase fragmentation because job sizes and job runtimes tend to be correlated, Scojo-PECT employs additionally safe non-type slice-backfilling. This means that preempted or waiting jobs of a different type may be backfilled into a slice-with this backfilling only being valid until the end of the slice-if they do not delay any job of the slice type or any of their own type jobs according to the backfilling approach applied. In [5] and [19], this optimization is shown to be crucial for good performance.

All schedulers are configured to support 3 job classes based on their runtime: short (S), medium (M), and long (L) jobs. The original classification is kept if resizing the jobs. We use A to denote results for all jobs. The backfilling approach applied is for all schedulers conservative backfilling.

3.2 Test Setup

Thus, the following schedulers were tested:

– Standard space-sharing with FCFS ($FCFS$)
– Standard space-sharing with priorities ($Prio10$)
– Scojo-PECT coarse-grain time sharing with separation of job types ($PECT$)

The parameters of Scojo-PECT were set to 30% relative time share for M jobs and 70% for L jobs, 60 sec overhead per time slice for preemption/resumption of the jobs, and 1 h time intervals for scheduling one S (optional), one M, and one L time slice. Jobs are classified as S if they run $\leq$ 10 minutes, as M if they run $\leq$ 3 hours, and as L otherwise.

The schedulers were tested with the Lublin-Feitelson workload model [9], setting the original machine size to 128 nodes, and with the CM5 trace from the Feitelson workload archive [7]. The Lubin-Feitelson model creates one process per node. The CM5 trace has 32 CPUs per node and sizes only come in multiple of 32 (i.e. do not differentiate the use per node)—thus, all sizes (including the original machine size measured in 1024 CPUs) were divided by 32 to map them to the model of one process per node. In each test run, 10,000 jobs were simulated.

Job size was modified by certain factors F_A in the range between 1.5 and 0.3. For the basic tests without modification of the scheduling algorithm, the same F_A was applied to all jobs per experiment. Job sizes were always rounded up. Note that the Lublin-Feitelson model creates about 25% serial jobs which never change their size. The jobs with maximum size never grow beyond this size and can only become smaller. The job sizes and corresponding runtimes created by the Lublin-Feitelson model or the trace were taken as the original sizes/runtimes with F_A=1.0. Runtimes are considered to be correct estimates.

An efficiency model was built on top of the created workload and the following efficiency parameters tested:

– Equal efficiency for all resource allocations, which means that the jobs can be size adapted in a work-conserving manner (WC)
– Efficiency described via typical efficiency speedup curves, which were modeled with a simple phase-wise linear approximation between pairs of 4 sizes: a) 1, b) 0.5 * original size, c) original size, and d) 2 * original size, using efficiencies of
  1. 1, 0.8, 0.65, and 0.4 ($E1$)
  2. 1, 0.75, 0.65, and 0.5 ($E2$)
  3. 1, 0.7, 0.65, and 0.6 ($E3$)

This is similar to the idea of Secvik's modeling approach presented in [15] and sufficient for the purpose of our experiments to only study the principle effects of different efficiencies. With the same argument, the efficiency is assumed to be the same for all jobs. In other work, we develop efficiency/scalability models for applications on multi-core CPUs [10].

For tests with different resource allocation to virtual machines, the original machine size is modified by a factor F_{VM}, while still using the job sizes and runtimes generated for the original machine sizes, adjusted by specified F_A factor. Resizing of virtual machines is assumed to be done without any performance impact (slowdown) from other virtual machines which may share the server.

Throughout the paper, the evaluation uses average response times R and average bounded relative response times (bounded slowdowns) RR. RR which relates response times to runtimes is calculated vs. the pure runtime of the job (without time slicing) and always vs. the original runtime without size modification. The graphs evaluate relative worsening, i.e. $RR_A/RR - 1$, if $F_A > 1$, and relative improvement, i.e., $RR/RR_A - 1$, if $F_A < 1$ to obtain a balanced presentation (rather than one end providing results in the range [0,1] which are hard to differentiate).

3.3 Formal Results for Work-Conserving and Non-Work-Conserving Job Re-Shaping

Before presenting the results of the experiments, below some simple theoretical considerations are presented to help explain some effects in the experiments.

Theorem 1: We assume that all job runtimes T are equal, have equal original size Sz with $Sz = Mz$ (machine size), and constant and equal efficiency under different sizes. Additionally, we assume off-line scheduling of a fixed set of jobs. This means that jobs can be reshaped in a work-conserving manner. N_J is the number of jobs resized to fit together into memory rather than being scheduled serially, and N is the overall number of jobs in the system, $N_G = N/N_J$ the number of groups under re-shaping, then serial scheduling gives average response times ϕR of

$$\phi R_{Serial} = T * (\sum_{i=1,N} i)/N = T * (N * (N+1)/2)/N = T * (N+1)/2 \qquad (1)$$

and re-shaped-job adaptive scheduling gives

$$\phi R_{Adaptive,WC} = N_J * T * (\sum_{i=1,N_G} i)/N_G \qquad (2)$$

If rewriting (1) for better comparison, this gives:

$$\phi R_{Serial} = T * (\sum_{i=1,N_J} i)/N_J + N_J * T * (\sum_{i=1,N_G-1} i)/N_G \qquad (3)$$

Proof: Under serial scheduling, the first job's response time is T, the second one has a wait time W of T and a runtime of T, i.e. a response time of $R = T +$

$W = 2T$, etc., which means that $\phi R_{Serial} = (\sum_{i=1,N}(i*T))/N = T*\sum_{i=1,N} i/N$ (q.e.d.)

Under reshaping, the runtime per job changes to $N_J * T$. The response time is equal for all the jobs in the group, with the first group having an average response time of $N_J * T$, the second group a wait time of $N_J * T$ and a runtime of $N_J * T$, i.e. an average response time of $2 * N_J * T$, etc., which means that $\phi R_{Adaptive,WC} = (\sum_{i=1,N_G}(iN_J * T))/N_G = N_J * T * \sum_{i=1,N_G} i/N_G$ (q.e.d.)

However, the runtime of the overall group is equal to the sum of the serial runtimes of those jobs, i.e. the average wait time for the next group of jobs is the same under both approaches. Thus, (1) can be transformed into (3).

Comparing (2) and (3) shows that the term $N_J * T * (\sum_{i=1,N_G-1} i)/N_G$ is common. Thus, for large N_G, the difference becomes small. However, the difference can matter if N_G is very small. This means high load makes the difference insignificant, whereas low load makes it relevant. In regards to the detailed difference, the remaining term in (2) is $t_{r,Serial} = T * (\sum_{i=1,N_J} i)/N_J$ and the remaining term in (3) is $t_{r,Adaptive,WC} = N_J * T$. Thus, $t_{r,Serial} < t_{r,Adaptive,WC}$ for $N_J > 1$ since $(N_J + 1)/2 < N_J$ for $N_J > 1$. This means that serial scheduling performs typically slightly better and that, for small N_G, the difference between serial and adaptive scheduling is more significant if N_J is larger.

Theorem 2: If reshaping is non work-conserving, i.e. jobs run with better efficiency if becoming smaller or lower efficiency if becoming larger, the runtimes are affected by the change in efficiency E, with $E_{F_A=1.0}$ being original and $E_{F_A=X}$ the efficiency after adaptation:

$$\phi R_{Adaptive,E} = N_J * T * E_{F_A=1.0}/E_{F_A=X} * (\sum_{i=1,N_G} i)/N_G \tag{4}$$

$$\begin{aligned}\phi R_{Adaptive,E} = {} & N_J * T * E_{F_A=1.0}/E_{F_A=X} + \\ & N_J * T * E_{F_A=1.0}/E_{F_A=X} * (\sum_{i=1,N_G-1} i)/N_G\end{aligned} \tag{5}$$

If comparing (3) and (4), $t_{r,Serial} = T*(\sum_{i=1,N_J} i)/N_J$ is not necessarily less than $t_{r,Adaptive,E} = N_J*T*E_{F_A=1.0}/E_{F_A=X}$ since $(\sum_{i=1,N_J} i)/N_J = (N_J+1)/2$ may not be less than $E_{F_A=1.0}/E_{F_A=X} * N_J$ if N_J is small and $E_{F_A=X}$ is much better than $E_{F_A=1.0}$. Thus, as can be easily seen from the formulation in (5), with larger N_G, adaptation reduces average response time by approximately $E_{F_A=1.0}/E_{F_A=X}$.

Thus, if assuming $E_{F_A=0.5} = 0.8$ and $E_{F_A=1.0} = 0.65$ (which corresponds to $E1$), $N_J = 2$, and $N = 2$ ($T_G = 1$)), $\phi R_{Adaptive,E} = E_{F_A=1.0}/E_{F_A=0.5} * N_J = 1.625$, whereas $\phi R_{Serial} = 1.5$. ϕR_{Serial} can be expected to be better than $\phi R_{Adaptive,E}$ under low load and to be increasingly worse with increasing load which is shown with the following calculations:

- $N_J = 2$ and $N = 4$ ($T_G = 2$): $\phi R_{Serial} = 2.5$ and $\phi R_{Adaptive,E} = 1.625*1.5 = 2.4375$ which means that adaptation is already slightly better

- $N_J = 2$ and $N = 6$ ($T_G = 3$):
  $\phi R_{Serial} = 3.5$ and $\phi R_{Adaptive,E} = 1.625 * 2 = 3.25$
- $N_J = 2$ and $N = 100$ ($T_G = 50$):
  $\phi R_{Serial} = 50.5$ and $\phi R_{Adaptive,E} = 1.625 * 25.5 = 41.4$ which means their ratio (1.21) is already close to the ratio of the efficiencies (1.23).

The writing as presented in (5) shows average runtime ϕT as the first term and average wait time ϕW as the second term. As can be seen from comparison to (2), with reshaping, ϕT becomes longer by $N_J * E_{F_A=1.0}/E_{F_A=X}$ and ϕW is reduced by $E_{F_A=1.0}/E_{F_A=X}$. The latter dominates for larger N_G.

In realistic scheduling, we face additionally packing problems, different runtimes and sizes, and potentially different efficiency. Moreover, submissions are dynamic. However, the above considerations give some basic clues about the expected performance behavior.

3.4 Experimental Results for Work-Conserving and Non-Work-Conserving Job Re-shaping

Before showing adaptation results which mostly look into relative performance under different adaptation, a note about absolute performance under $F_A = 1.0$. If setting resource shares for equal service, Scojo-PECT provides similar service to M and L jobs as the priority scheduler but improves overall response times by about 45% by serving S jobs better. The simple FCFS scheduler performs by about 60% worse for all jobs and by about 20% worse if only considering M and L jobs.

In the following, we investigate performance under different job re-shaping, while keeping the machine size at original size. Note that only the sizes of the jobs are changed and no special adaptive scheduler is used.

In regards to re-shaping, we can expect that

- If decreasing jobs sizes, the smaller sizes might provide better options for packing. If increasing job sizes, packing possibilities might worsen.
- A counter effect under work-conserving re-shaping as per (3) shows in phases of low load, while performance is approximately the same if the load is very high.
- Improvements under non-work-conserving re-shaping according to (4) show if reducing the job sizes, while performance is worsened if increasing the job sizes.

In our experiments, we explored the corresponding aggregate effects on RR for the 3 schedulers $FCFS$, $Prio10$, and $PECT$. The results are shown in Figure 1, Figure 2, Figure 3, and Figure 4, differentiated for M, L, and A jobs.

As we can see, under work-conserving re-shaping, the sizes make no relevant difference in regards to RR if scheduling with $FCFS$ or $PECT$ (though RR is shown, the same applies to R). Obviously any effects from better packing and extended wait times under low load cancel each other out, though improvements

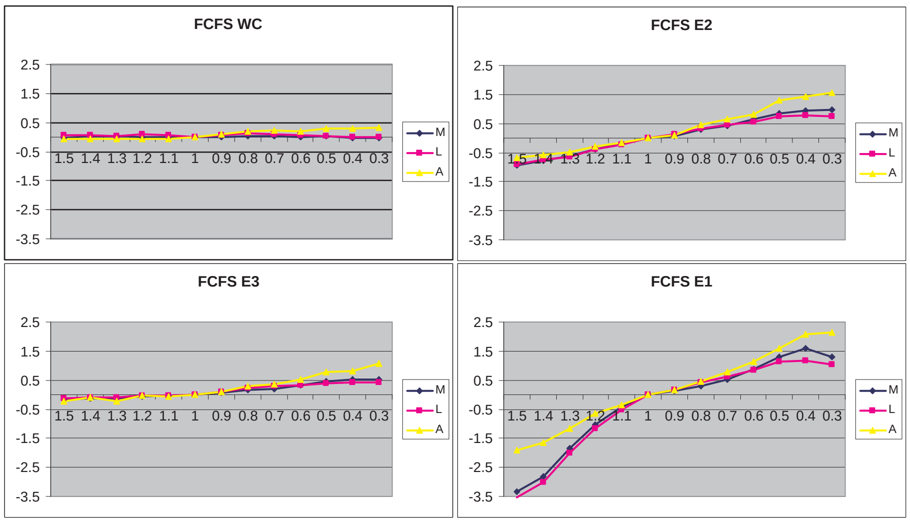

Fig. 1. RR improvement for Lublin-Feitelson workload under $FCFS$ with different efficiency, shown over different size-modification factors.

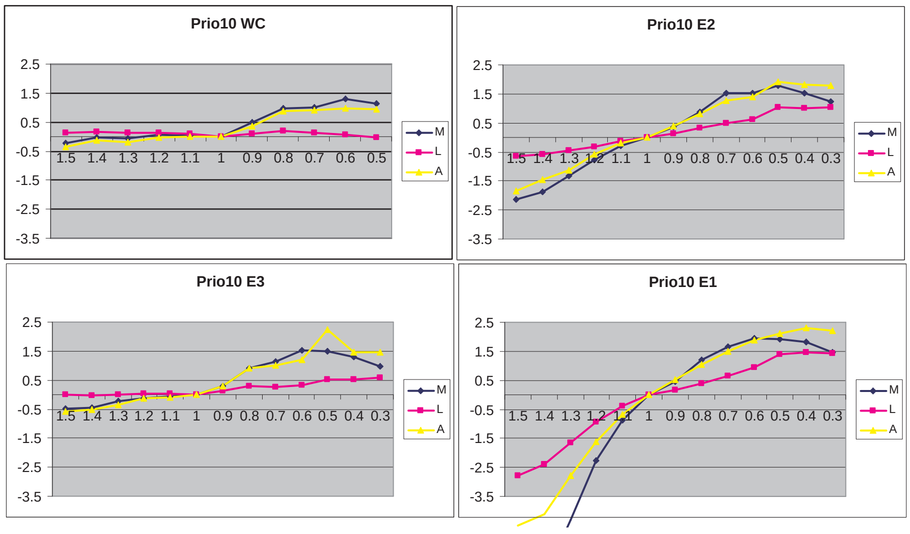

Fig. 2. RR improvement for Lublin-Feitelson workload under $Prio10$ with different efficiencies, shown over different size-modification factors.

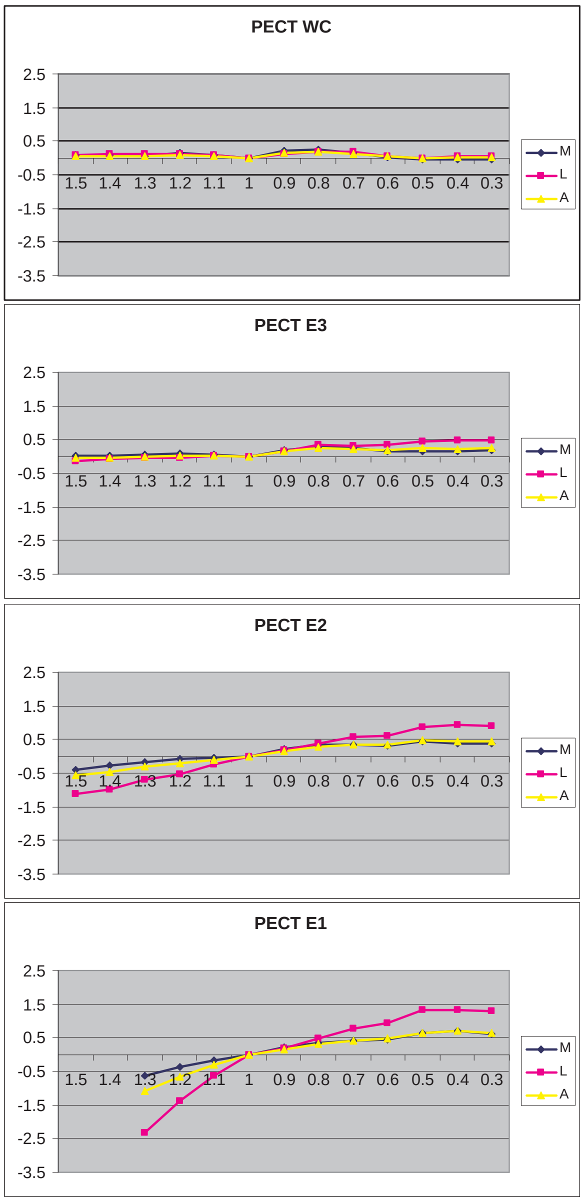

Fig. 3. RR improvement for Lubin-Feitelson workload under $PECT$ with different efficiencies, shown over different size-modification factors.

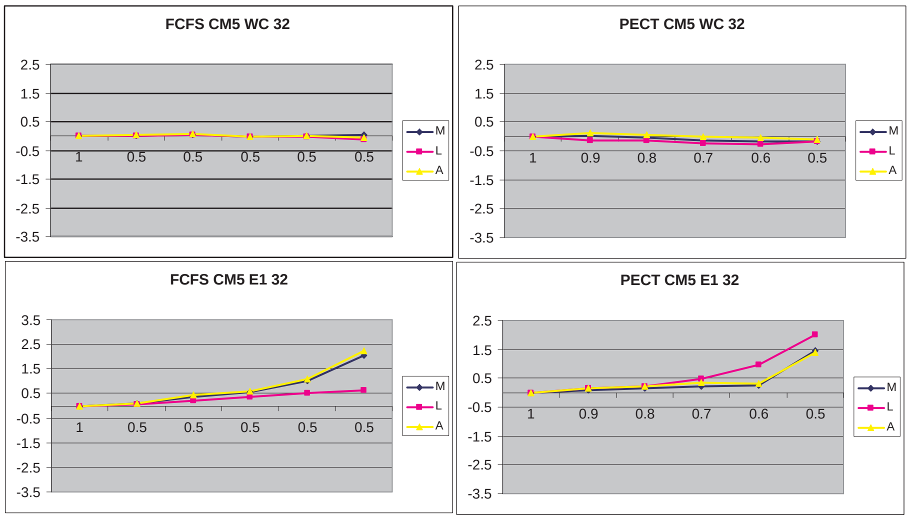

Fig. 4. RR improvement for CM5 trace under $FCFS$ and $PECT$ with work-conserving and with $E1$ efficiency, shown over different size-modification factors.

in packing might have been expected to make more of a difference. For $Prio10$, smaller sizes provide a benefit of up to a factor of 1.4.

The likely explanation is that packing and backfilling do not behave significantly different and that mostly the load of the jobs in the system matters (which is further investigated in Section 6.1.

Since, however, the overall load and the length of the queue may matter, Figure 5 shows results for the original load L_1 with $U = 0.76$, L_2 with $U = 0.66$ and L_3 with utilization $U = 0.56$ (loads are modified by increasing interarrival times via the α parameter from 10.33 to 10.65 and 10.9). Figure 5 also includes results for the interarrival times being changed to exponential without daily cycles which significantly reduces average queue lengths ϕQ_{system} of jobs in the system (waiting or running) though they are almost independent of F_A (ϕQ_{system} is about 13 for M and 31 for L with $L1$; about 5 for M and 15 for L with $L3$; and reduces to about half if interarrival times are exponential without daily cycles). However, the RR results are all similar. Even if queue lengths become shorter, F_A only changes the number of jobs fitting into the machine by maximally a factor of 2 ($N_J = 2$), i.e. $(N_J + 1)/2 = 1.5$ which means that the difference for small N_G is low. Thus, the results behave as expected for the theoretic consideration that were done for the simple off-line case with equal job sizes.

Under non-work-conserving job re-shaping, all factors $F_A < 1.0$ would suggest an improvement in R and RR which is indeed the case. The improvements are higher if the efficiency improvement is higher, i.e. best for $E1$ and lowest for

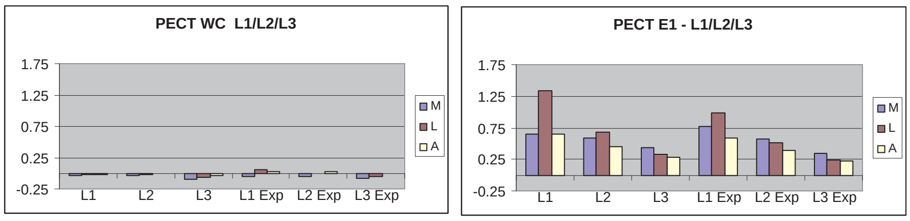

Fig. 5. Improvement for $F_A = 0.5$ vs. $F_A = 1.0$ with different loads, and either standard interarrival times of model or exponential interarrival times without daily cycles (*Exp*).

$E3$. If looking at R from the theoretic consideration in (5), $E_{1,F_A=0.5}/E_{1,F_A=1.0} = 0.8/0.65 = 1.23$. The real improvements due to higher efficiency should be lower since not all jobs change sizes (such as serial jobs always remaining serial). However, the measured R_L improves by a factor of 2.28 and the measured R_M by a factor of 1.67. Correspondingly, the measured RR_L improves by a factor of 2.34, RR_M by a factor of 1.65, and RR_A by a factor of 1.65. Thus, the changes in R and RR are almost identical. The higher-than-expected benefits may be partially due to better packing but likely also to less queue-up of work in the systems, as will be discussed in Section 6.1. To check the effect of the load on the improvements, again lower Loads $L2$ and $L3$ were tested (see Figure 5). Load $L2$ and $L3$ also show improvements though they are relatively lower than for $L1$ This can be explained by less difference in the work queuing up. Indeed, if looking at average wait times, they are—if changing from $F_A = 1.0$ to $F_A = 0.5$ and using $L1$—reduced from 9.3h to 2.4h for M jobs and from 33h to 7.4h for L jobs, which is much more than the predicted factor of 1.23 from (4).

Looking at the effect on runtimes, the change from F_A is limited: with $F_A = 1.0$, ϕT is 4.55h for M and 7.3h for L jobs and, with $F_A = 0.5$ and $E1$, ϕT is 5.9h for M and 10.3h for L jobs—which is a factor of 1.3 for M and a factor of 1.41 for L jobs. Predicted would be a factor of 1.625 for both (from $2 * E_{F_A=1.0}/E_{F_A=0.5}) = 2 * 0.65/0.8 = 1.625$) for adaptable jobs. With 25% serial jobs this reduces to a factor of 1.47. However, the lower load, especially for M jobs, also makes more non-type slice backfilling possible which improves service and reduces runtimes more than expected.

4 Adaptive Scheduling with Efficiency and Load Considerations

4.1 Scheduling with Adaptation to Load

As seen in Section 3.4, non-work-conserving scheduling provides significant benefits already from improved efficiency. Note that the scheduler simply re-shaped all jobs. In the following, we put more intelligence into the scheduler (Scojo-PECT) to consider different load situations and create a truly adaptive scheduler. The

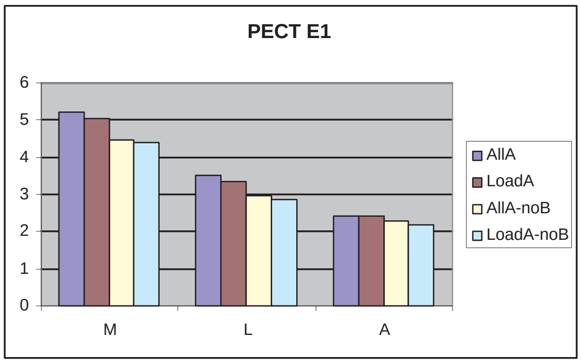

Fig. 6. RR for $PECT$ with static $F_A = 0.5$, with (*AllA*) and without (*AllA_noB*) application of the same factor to backfilled jobs, and for adaptive $PECT$ with dynamic F_{dynA} between $F_{A,min} = 0.5$ and $F_{A,max} = 1.0$, with (*LoadA*) and without (*LoadA_noB*) application to backfilled jobs.

adaptive scheduler is applied separately and independently per job type M and L. Since $F_A = 0.5$ performed best in the basic experiments, we use this factor in the following experiments with an adaptive scheduler. Adaptation is performed according to the following steps:

1. The resource needs are calculated as the sum of the sizes of all waiting jobs (assuming that ideally all jobs should be allocated). The factor used for adaptation $F_{dyn,A}$ in the dynamic scheduling context is then adjusted as

$$max(F_{try,A}, F_{A,min}) \leq F_{dyan,A} \leq min(F_{try,A}, F_{A,max}) \qquad (6)$$

   with $F_{A,min} = 0.5$ and $F_{A,max} = 1.0$ in the following.
2. All jobs of a specific job type which the scheduler tries to fill into the machine, are then reshaped by $F_{dyan,A}$ unless the jobs are relatively long-running in their own jobs class. Thus, if the original job runtime is > 45 min (M jobs) or > 7 h (L jobs), the factor $F_{A,min}$ is used for reshaping.

The result of applying this approach (called *LoadA*) are shown in Figure 6. The improvement is relatively low. However, not adapting backfilled jobs made a difference and best results were obtained if combining both, load adaptive resizing and keeping the original size for backfilling (*LoadA_noB*). However, the improvements are still moderate. (Note that further improvements may be obtained by looking at each scheduling situation in detail to reduce fragmentation. However, previous work [21] suggests that the improvements would be minor.)

The obtained moderate additional improvements with dynamic job-size adaptation suggest that a substantial part of the benefits obtained in previous research with adaptive schedulers may been due to improved efficiency.

5 Job Re-Shaping for Virtual Machines of Adaptive Size

In the following, we show results from changing the size of the machine by a factor F_M which corresponds to different resource allocation to virtual machines if partitioning core numbers among them. At the same time, the sizes of the jobs per virtual machines are adjusted by a static factor F_A. We tested resulting virtual-machine sizes of 96, 112, 144, and 160 nodes, using the Lublin-Feitelson workload for 128 nodes and $E1$ efficiency. The results are shown in Figure 7.

The sizes delivering similar results as $F_A = 1.0$ does for 128 nodes are $F_A = 0.5$ (still somewhat higher) for 96 nodes, $F_A = 0.75$ for 112 nodes, $F_A = 1.07$ for 144 nodes, and $F_A = 1.21$ for 160 nodes. The results demonstrate that certain performance (QoS) can be kept over different resource allocations per virtual machine if re-shaping the jobs accordingly.

The sizes delivering similar results as $F_A = 1.0$ does for 128 nodes are $F_A = 0.5$ (still somewhat higher) for 96 nodes, $F_A = 0.75$ for 112 nodes, $F_A = 1.07$ for 144 nodes, and $F_A = 1.21$ for 160 nodes. The results demonstrate that certain performance (QoS) can be kept over different resource allocations per virtual machine if re-shaping the jobs accordingly.

6 Predicting *R* under Varying Resource Allocation

6.1 Dependence of *R* and *RR* on Utilization

Trying to generalize the performance in dependence on different virtual machine sizes, different job-reshaping factors, and different efficiencies, Figure 8 and Figure 10 plot RR and Figure 9 and Figure 11 plot R in dependence on the measured utilization for the corresponding test runs. Though M and L jobs perform differently, we observe a clear correlation between relative response times and utilization per job type. The correlation is stronger for L jobs but still reasonably clear for M jobs. This permits the conclusion that utilization changes from running jobs at better efficiency are the major source of improvements in adaptive job scheduling and that utilization makes a good predictor for performance if changing virtual-machine and job sizes (cf. (5)).

This is a nice property since utilization is easy to predict if knowing the efficiency changes and the mix of the sizes (especially the percentage of serial jobs), as utilization corresponds to the submitted load (work over time).

The explanation of the response times not matching the simplified off-line case is that we are dealing with dynamics in interarrival times, daily cycles, job runtimes, and job sizes which can temporarily lead to very long queues. Thus, the proper approach is a queuing model which, however, is hard to establish due to the many contributing statistical distributions.

6.2 A Simple Predictive Model

If knowing queue lengths, response times can be predicted via Little' Law which is independent of the specific statistical distributions and the scheduling policies

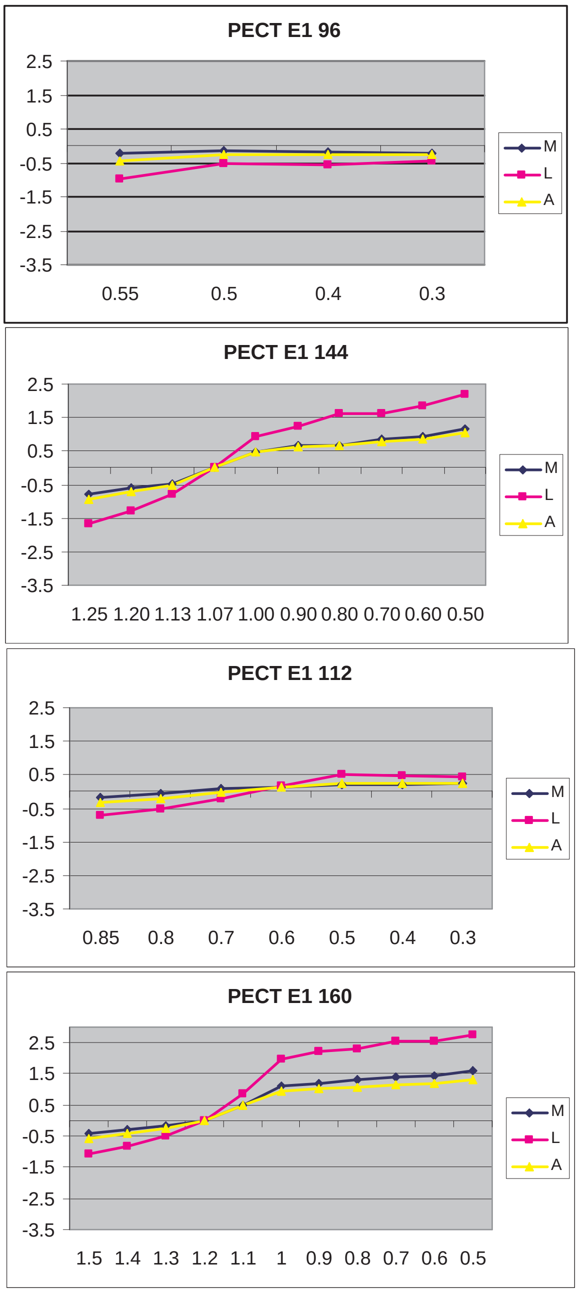

Fig. 7. RR with different resizing of the machine combined with different re-shaping of the workload, using $PECT$ and $E1$.

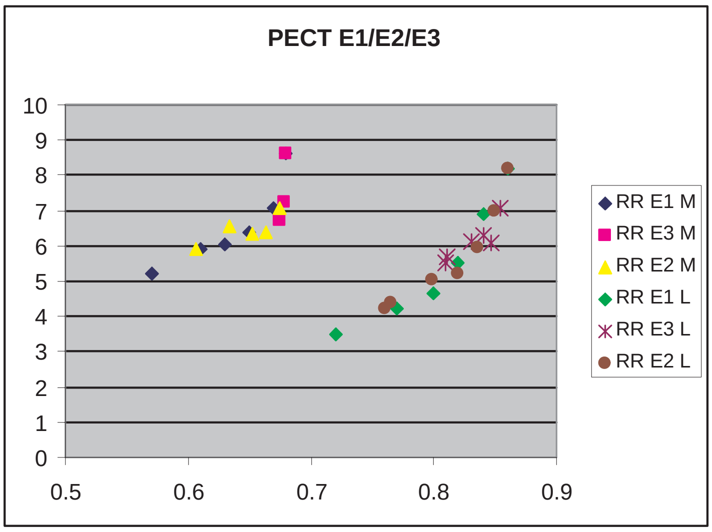

Fig. 8. RR for $PECT$ and $E1$, $E2$, and $E3$ with different F_A, shown over corresponding measured utilization.

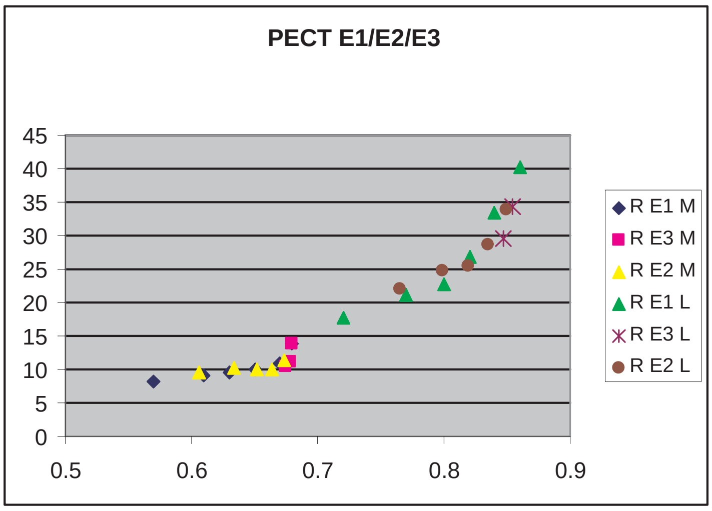

Fig. 9. R for $PECT$ and $E1$, $E2$, and $E3$ with different F_A, shown over corresponding measured utilization.

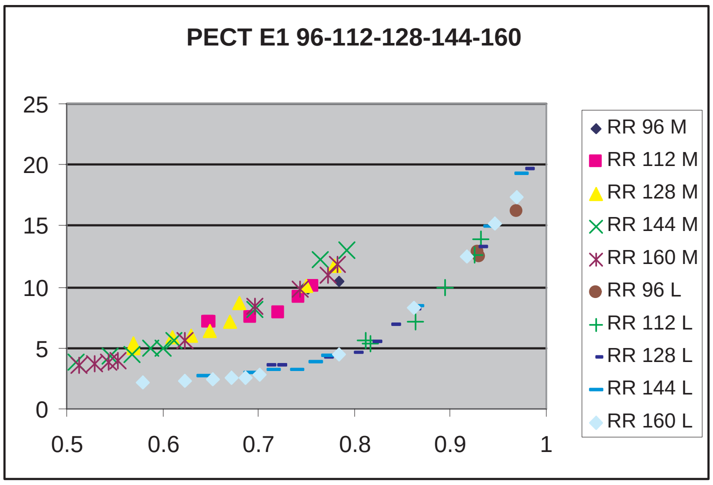

Fig. 10. RR for $PECT$ and $E1$ with different virtual-machine sizes and different F_A, shown over corresponding measured utilization.

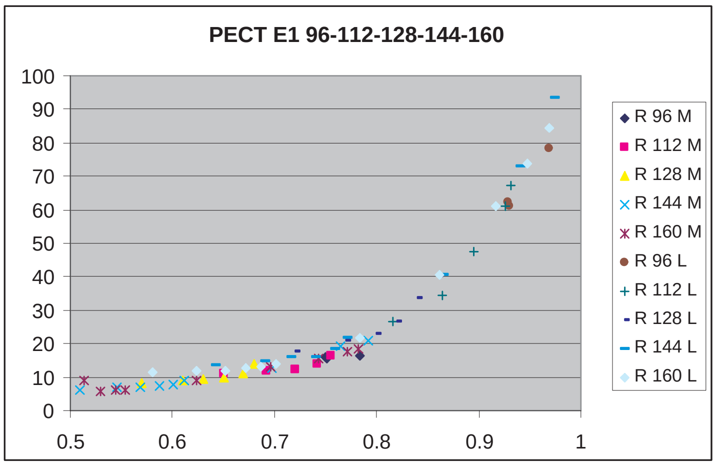

Fig. 11. R for $PECT$ and $E1$ with different virtual-machine sizes and different F_A, shown over corresponding measured utilization.

and says that

$$\phi Q_{system} = \lambda * \phi R \tag{7}$$

with λ being the average arrival rate and ϕQ_{system} the average number of jobs in the system (waiting and running). However, the law applies to a single-server system. We approximate the packing of multiple jobs into the machine as a variation of service time to obtain a single-server model. This simplification appears to be feasible, considering the predictions shown in Figure 12. The results show a few cases of prediction from the standard model and the exponential interarrival times without daily cycles. As we can see, the predictions are very accurate—a little too high for the standard interarrival model and a little too low for the exponential interarrival model.

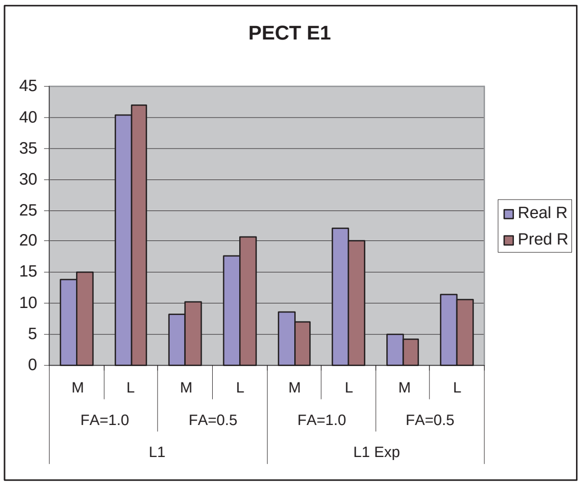

Fig. 12. Real and predicted R (from Q_{system}) for $PECT$ and $E1$ with different F_A. Both standard model and exponentially distributed interarrival times without daily cylces are shown.

Our final goal is predicting response times for different resource allocations F_A and different F_M from a base resource allocation which in our case is $F_A = 1.0$ and $Mz = 128$. As mentioned above, the change in utilization can be directly calculated from the workload. Since the queuing model can capture the multi-node server behavior, we use an M/G/1 model to predict R from the utilization, which means

$$\phi Q_{system} = U + U^2(1 + C^2S)/(2 * (1 - U))) \tag{8}$$

With this simplified model, all variations of daily cycles, sizes, backfilling, and runtimes are mapped to the variation coefficient of the service time C^2S.

The concrete parameter is obtained from fitting the curve for $F_A = 1.0$ which reflects predicting from a base-case allocation.

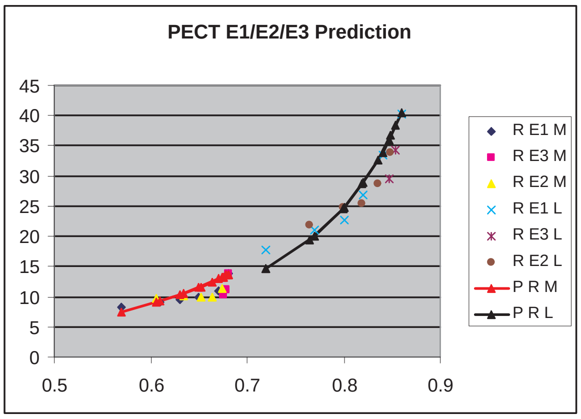

Fig. 13. Real and predicted R (from U) for $PECT$ and $E1$ with different F_A, shown over utilization.

The results from applying this simple prediction approach based on (7) and (8) to our workload are shown in Figure 13 and Figure 14. Considering that the actual R curves shows some irregularities, the prediction is a good fit (and worked better than all more detailed models tried). However, the curves match a little less well if utilization decreases and would be completely off if utilization increases beyond 0.9. The reason is that variation increases with lower utilization and decreases with very high utilization. For utilization beyond 0.9, the wait times are in the order of days, i.e. daily cycles have little impact, and backfilling opportunities can be expected to saturate. However, the extremely long wait times make such allocations anyway undesirable. Thus, the simple prediction model works well for practically relevant cases.

7 Summary and Conclusion

The experiments in this paper have investigated the effects on relative response times merely obtained from reshaping all jobs statically to run with smaller sizes (and correspondingly longer runtimes), with or without changing the (virtual) machine size. The results show that under constant machine size and FCFS scheduling, the results remain the same if reshaping the jobs in a work-conserving manner (no gains from efficiency). However, significant benefits are obtained if jobs reshaped to smaller size run at higher efficiency. Any further benefits obtained from an adaptive scheduler which decides job sizes dynamically at job

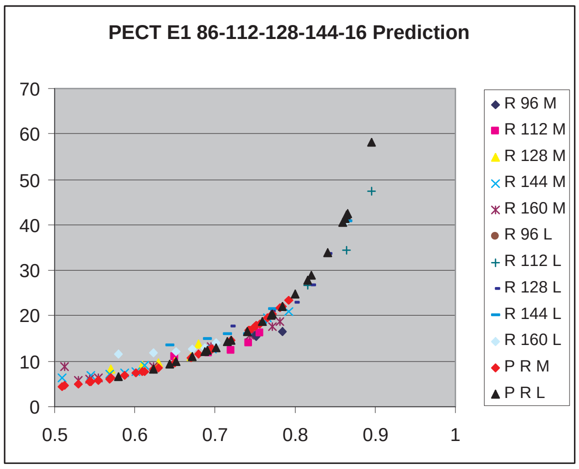

Fig. 14. Real and predicted R (from U) for $PECT$ with different virtual-machine sizes and different F_A, shown over utilization.

start time according to the machine load are relatively small. Simple formulas looking at the off-line schedules of a series of jobs can only partially explain the effects but dynamic measurement of utilization proves to show a strong correlation to the obtained relative response times under varying virtual-machine and job-size adjustments. Ongoing work is to establish a more detailed predictive model for response times under varying resource allocation via different job sizes and different virtual-machine sizes.

References

[1] Barsanti, L., Sodan, A.C.: Adaptive Job Scheduling via Predictive Job Resource Allocation. Proc. of JSSPP Workshop, Saint-Malo, France, Springer, LNCS 4376 (June 2006) 115-140

[2] Chiang, S.-H., Vernon, M.K.: Dynamic vs. Static Quantum-Based Parallel Processor Allocation. Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, LNCS 1162 (May 1996) 200-223

[3] Cirne , W., Berman, F.: When the Herd is Smart-Aggregate Behavior in the election of Job Request. IEEE Trans. on Par. and Distr. Systems, 14(2) (Feb. 2003)

[4] Downey, A.: A Model for Speedup of Parallel Programs. Technical Report CSD-97-933, Univ. of California Berkeley (Jan. 1997)

[5] Esbaugh, B., Sodan, A.C.: Coarse-Grain Time Slicing with Resource-Share Control in Parallel-Job Scheduling. Proc. High Performance Computing and Communication (HPCC), Houston, September, Springer, LNCS 4782 (2007)

[6] Feitelson, D.G., Rudolph, L., Schwiegelsohn, U., Sevcik, K.C., Parsons, W.: Theory and Practice in Parallel Job Scheduling. Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, LNCS 1291 (1997)

[7] Feitelson Workload Archive, available at http://www.cs.huji.ac.il/labs/parallel/workload/logs.html (ast retrieved January 2008)

[8] Ghosal, D., Serazzi, G., Tripathi, S.K.: The Processor Working Set and Its Use in Scheduling Multiprocessor Systems. IEEE Trans. Software Engineering, 17(5), (May 1991) 443-453

[9] Lublin, U., Feitelson, D.G.: The Workload on Parallel Supercomputers-Modelling the Characteristics of Rigid Jobs. Journal of Parallel and Distributed Computing, 63(11) (Nov. 2003) 1105-1122

[10] Machina, J., Sodan, A.C.: Predicting Cache Needs and Cache Sensitivity for Applications in Cloud Computing on CMP Servers with Configurable Caches. Workshop on System Management Techniques, Processes, and Services (SMTPS) of IPDPS, Proc. IPDPS, IEEE, Rome (May 2009)

[11] McCann, C., Zahorjan, J.: Processor Allocation Policies for Message Passing Parallel Computers. Proc. SIGMETRICS Conf. Measurement & Modeling of Computer Systems (May 1994) 208-219

[12] Naik, V.K., Setia, S.K., Squillante, M.K.: Processor Allocation in Multiprogrammed Distributed-Memory Parallel Computer Systems. Journal of Parallel and Distributed Computing, 46(1) (1997) 28-47

[13] Parsons, E.W., Sevcik, K.C.: Implementing Multiprocessor Scheduling Disciplines. Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, LNCS 1291 (1997)

[14] Rosti, E., Smirni, E., Serazzi, G., Dowdy, L.W.: Analysis of Non-Work-Conserving Processor Partitioning Policies. Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) (1995)

[15] Sevcik, K.C.: Characterization of Parallelism in Applications and Their Use in Scheduling. Performance Evaluation Review 17 (1989) 171-180

[16] Sodan, A.C., Deshmeh, A., Esbaugh, B., Machina, J.: Thread-Level Parallelism in Modern CPUs. Submitted to journal.

[17] Sodan, A.C.: Dynamic Job Scheduling for Computational Grids. Book chapter in Grid Computing Research Progress, Nova Science Publisher, Inc., Hauppauge, NY (August 2008)

[18] Sodan, A.C.: Autonomic Share Allocation and Bounded Prediction of Response Times in Parallel Job Scheduling for Grids. Workshop on Adaptive Grid Computing (NCA-AGC), Proc. IEEE Int. Symp. on Network Computing and Applications (NCA), Cambridge (July 2008) 307-314

[19] Sodan, A.C., Esbaugh, B.: Service Control and Service Prediction with the Preemptive Parallel Job Scheduler Scojo-PECT. Submitted to journal.

[20] Sodan, A.C., Lan, L.: LOMARC Lookahead Matchmaking for Multiresource Coscheduling on Hyperthreaded CPUs. IEEE Transactions on Parallel and Distributed Systems, 17(11) (Nov. 2006) 1360-1375

[21] Sodan, A.C., Huang, X.: Adaptive Time/Space Sharing for Workload Adaptation and Fragmentation Reduction. IJHPCN, 4(5/6) (2006) 256-269

[22] Srinivasan, S., Subramani, V., Kettimuthu, R., Holenarsipur, P., Sadayappan, P.: Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs. Proc. HiPC (2002)

[23] Sutter, H.: The Free Lunch is Over-A Fundamental Turn Toward Concurrency in Software. Dr. Dobb's Journal, 30(3) (March 2005)

[24] Weinberg, J., Snavely, A.: Symbiotic Space-Sharing on SDSC's DataStar System. Proc. of JSSPP Workshop, Saint-Malo, France, Springer, LNCS 4376(June 2006)