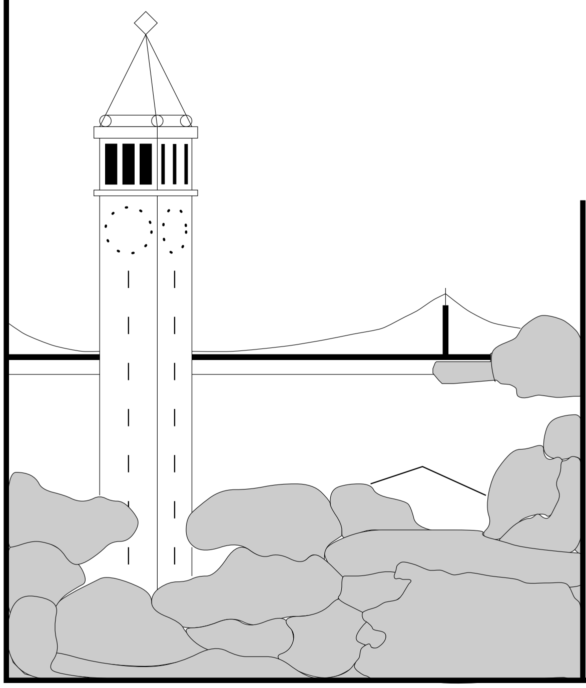
A parallel workload model and its implications for processor allocation

Allen B. Downey



Report No. UCB/CSD-96-922

November 1996

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A parallel workload model and its implications for processor allocation

Allen B. Downey *

November 1996

Abstract

We develop a workload model based on the observed behavior of parallel computers at the San Diego Supercomputer Center and the Cornell Theory Center. This model gives us insight into the performance of strategies for scheduling malleable jobs on space-sharing parallel computers. We find that Adaptive Static Partitioning (ASP), which has been reported to work well for other workloads, is inferior to some FIFO strategies that adapt better to system load. The best of the strategies we consider is one that explicitly restricts cluster sizes when load is high (a variation of Sevcik's *A+* strategy [13]).

Keywords: parallel, space-sharing, partitioning, scheduling, allocation, malleable, multiprocessor.

1 Introduction

Space-sharing, distributed-memory multiprocessors, like the Intel Paragon, the Cray T3E and the IBM SP2, are often used in supercomputing environments to support scientific applications. These environments typically have the following characteristics:

- For batch processing, jobs do not share processors, but rather allocate a *cluster* of processors exclusively and *run to completion.* Many of these machines also have an interactive partition that uses timesharing, but this paper only addresses scheduling strategies for batch partitions (pure space-sharing). In the environments we have observed, the vast majority of computation is done in batch mode.

*EECS — Computer Science Division, University of California, Berkeley, CA 94720 and San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186. Supported by NSF grant ASC-89-02825 and by Advanced Research Projects Agency/ITO, Distributed Object Computation Testbed, ARPA order No. D570, Issued by ESC/ENS under contract #F19628-96-C-0020. email: downey@sdsc.edu, http://www.sdsc.edu/~ downey

- Many jobs on these systems are *malleable*, meaning that they are capable of running on a range of cluster sizes. On the other hand, the programming models used for scientific applications usually do not generate jobs that can change cluster sizes *dynamically*. Thus, once a job begins execution, its cluster size is fixed.

In current systems, users choose cluster sizes for their jobs by hand, and the system does not have the option of allocating more or fewer than the requested number of processors. The factors that should influence the choice of a cluster size include the characteristics of the application (resource requirements), the load on the system (resource availability) and the performance requirements of the user (deadlines, desired throughput, *etc.*). But users generally do not have the information, tools, or inclination to weigh all of these factors accurately. Allowing the system to make this decision has the potential to improve system utilization and reduce users' wait times.

Toward this end, prior studies have proposed *allocation strategies* that choose automatically how many processors to allocate to each job. Most of these studies evaluate the proposed strategies with analysis and simulation based on hypothetical workloads. Each of the strategies we consider in this paper has been evaluated, under different workload assumptions, in at least three of [3] [13] [11] [12] [14] [5] [1] [9]. Chiang *et al.* [1] compare the performance of these strategies over a wide range of workload parameters, and argue that the discrepancies among various studies are due to differences in the hypothesized workloads.

The goal of this paper is to focus this debate by constructing a new workload model based on observations of space-sharing systems running scientific workloads. We intend this model to cover the range of workload parameters most relevant to current workloads and hence most applicable to real systems. By narrowing the range of workload parameters, we are able to examine the proposed strategies in more detail and gain insight into the reasons for their success or failure.

Section 2 presents the speedup model we use for our workload. The model defines a relationship between the speedup of a job and the mean and variance of its parallelism profile, making it possible to evaluate the importance of these parameters for scheduling. Section 3 explains the distributions that make up our workload model. Section 4 describes the simulation we used to evaluate the scheduling strategies in Section 5. Section 6 presents the results of our scheduling simulations. Section 7 summarizes our findings and proposes future work.

2 The speedup model

In [2] we developed a model of parallel speedup that estimates the speedup of a program as a function of its average parallelism and its variance in parallelism. The intent of this model is to find a family of speedup curves that are parameterized by the average parallelism of the program, A, and the variance in parallelism, V. To do this, we construct a hypothetical *parallelism profile*[1] with the desired values of A and V, and then use this profile to derive speedups. We use two families of profiles, one for programs with low V, the other for programs with high V. In [2] we show that this family of speedup profiles captures, at least approximately, the behavior of a variety of parallel scientific applications on a variety of architectures.

2.1 Low variance model

Figure 1a shows a hypothetical parallelism profile for an application with low variance in parallelism. The potential parallelism is A for all but some fraction σ of the duration ($0 \leq \sigma \leq 1$). The remaining time is divided between a sequential component and a high-parallelism component. The average parallelism of this profile is A; the variance of parallelism is $V = \sigma(A-1)^2$.

A program with this profile would have the following speedup as a function of the cluster size n:

$$S(n) = \begin{cases} \frac{An}{A+\sigma(n-1)/2} & 1 \leq n \leq A \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)} & A \leq n \leq 2A-1 \\ A & n \geq 2A-1 \end{cases} \tag{1}$$

2.2 High variance model

Figure 1b shows a hypothetical parallelism profile for an application with high variance in parallelism. This profile has a sequential component of duration σ and a parallel component of duration 1 and potential parallelism $A + A\sigma - \sigma$. By design, the average parallelism is A; by good fortune, the variance of parallelism is $\sigma(A-1)^2$, the same as that of the low-variance model. Thus for both models σ is approximately the square of the coefficient of variation, CV^2. This approximation follows from the definition of coefficient of variation, $CV = \sqrt{V}/A$. Thus, CV^2 is $\sigma(A-1)^2/A^2$, which for large A is approximately σ.

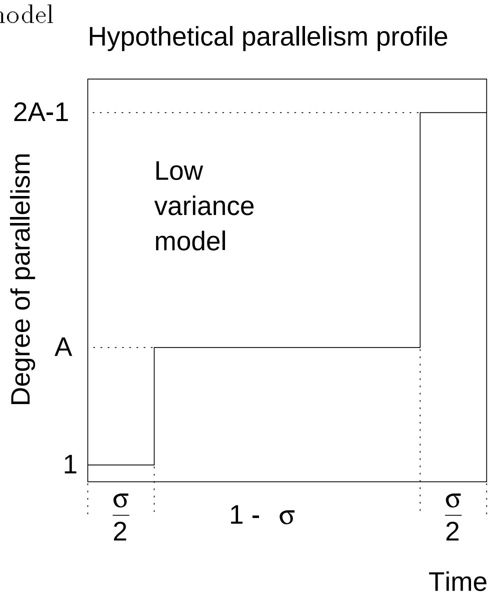
A program with this profile would have the following speedup as a function of cluster size:

$$S(n) = \begin{cases} \frac{nA(\sigma+1)}{A+A\sigma-\sigma+n\sigma} & 1 \leq n \leq A + A\sigma - \sigma \\ A & n \geq A + A\sigma - \sigma \end{cases} \tag{2}$$

Figure 2 shows a set of speedup curves for a range of values of σ (and $A = 64$). When $\sigma = 0$ the curve matches the theoretical upper bound for

[1] The parallelism profile is the distribution of potential parallelism of a program[13].

a) Low-variance model



b) High-variance model

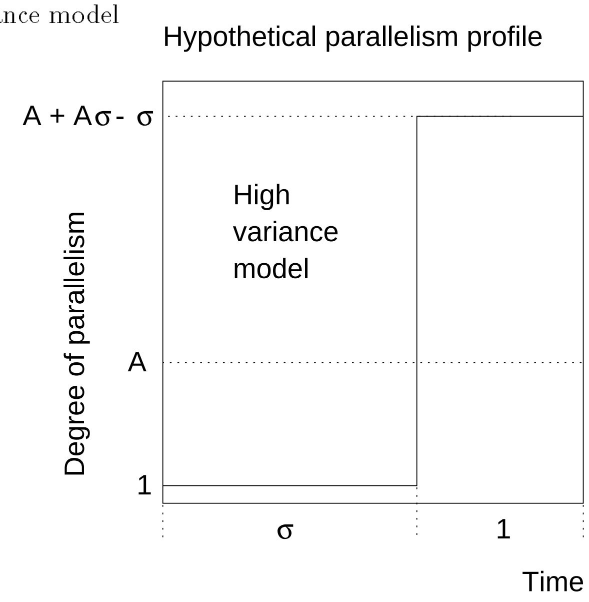


Figure 1: The hypothetical parallelism profiles we use to derive our speedup model.

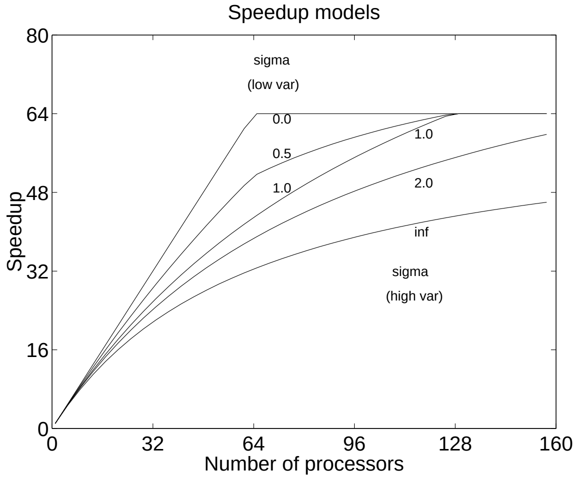


Figure 2: Speedup curves for a range of values of σ.

speedup—bound at first by the "hardware limit" (linear speedup) and then by the "software limit" (the average parallelism A). As σ approaches infinity, the curve approaches the theoretical lower bound on speedup derived by Eager *et al.* [3]:

$$S_{min}(n) = An/(A + n - 1) \tag{3}$$

When $\sigma = 1$ the two models are identical.

3 Distribution of parameters

In order to evaluate potential scheduling strategies, we would like to construct a workload model that is as realistic as possible, given the available information about real workloads. In this section we present our observations of the Paragon at SDSC and the SP2 at CTC, and use these observations to derive a workload model. This workload model will generate, for purposes of simulation, a set of jobs with distributions of lifetimes and parallelism parameters (A and σ) similar to those of real workloads.

3.1 Distribution of lifetimes

Ideally, we would like to know the distribution of L, the *sequential lifetime*, for a real workload. Sequential lifetime is the time a job would take on a single processor, so if we knew L, A and σ, we could calculate the speedup, $S(n, A, \sigma)$, on n processors and the *run time*, L/S. But the accounting data we have from real systems does not contain sequential lifetimes; in general L is not known, and often it is not even defined, because memory requirements prevent some jobs from running on a single processor. On the other hand, we do know the *total allocated time*, T, which is the product of wall clock lifetime and cluster size. For programs with linear speedup, T equals L, but for programs with sublinear speedups, T can be much larger than L. Keeping this overestimation in mind, we will use observed distributions of T to construct the distribution of L for our workload.

We have examined accounting logs from the Intel Paragon at the San Diego Supercomputer Center (SDSC) and the IBM SP2 at the Cornell Theory Center (CTC). Figure 3 shows the distribution of total allocated time for these machines. On both machines, the distribution is approximately linear in log space, which implies that the cumulative distribution function has the form:

$$cdf_T(t) = Pr\{T \leq t\} = \beta_0 + \beta_1 \ln t \qquad t_{min} \leq t \leq t_{max} \tag{4}$$

where β_0 and β_1 are the intercept and slope of the observed line. The upper and lower bounds of this distribution are $t_{min} = e^{-\beta_0/\beta_1}$ and $t_{max} = e^{(1.0-\beta_0)/\beta_1}$.

Since this distribution is uniform in log space, we call it a *uniform-log distribution*. We know of no theoretical reason that the distribution should have this shape, but we believe that it is pervasive among batch workloads, since we have observed similar distributions on the Cray C90 at SDSC, and other authors have reported similar distributions on other systems [4][15].

For the SDSC workload, we estimated the parameters $\beta_0 = -0.14$ and $\beta_1 = 0.073$ by linear regression. The fitted line, shown in Figure 3a, is a good match for the observed data, except for the shortest 5% of jobs. This discrepancy is negligible, though; the only effect is that a few jobs that we expect to run 10 seconds would run only one second instead.

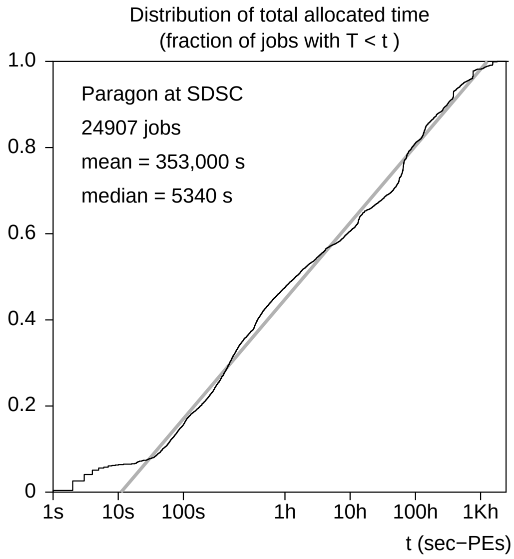
For the CTC workload, we estimated an alternate model based on a two-stage uniform-log distribution. In this model there are two classes of jobs: short jobs, from a uniform-log distribution between $e^{2.0}$ and $e^{11.1}$ seconds, and long jobs, from a uniform-log distribution between $e^{11.1}$ and $e^{14.5}$ seconds. The multistage model, shown in Figure 3b, is visually a good fit for the measured data.

Since T overestimates the sequential lifetimes of jobs, our workload model uses a distribution of L with a somewhat lower maximum than the distributions we observed. In our simulations, L is distributed in a single uniform-log stage between e^2 and e^{12} seconds. The median of this distribution is 18 minutes; the mean is 271 minutes.

3.2 Distribution of average parallelism

For our workload model, we would like to know the parallelism profile of the jobs in the workload. But the parallelism profile reflects *potential* parallelism, as if

a) SDSC workload



b) CTC workload

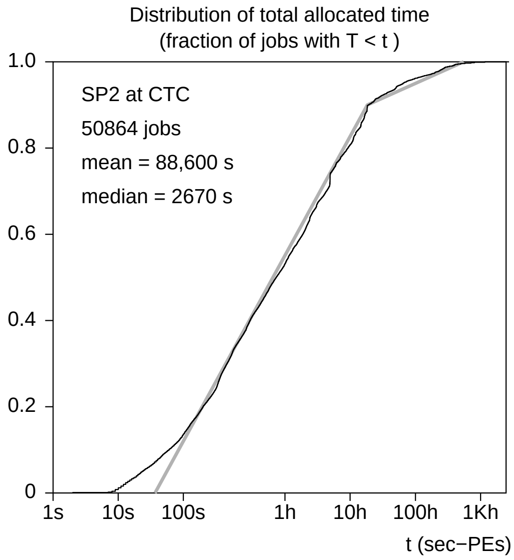


Figure 3: Distribution of total allocated time (wall clock time multiplied by number of processors) for 24907 batch jobs from the Intel Paragon at the San Diego Supercomputer Center (SDSC) and for 50864 batch jobs from the IBM SP2 at the Cornell Theory Center (CTC). The gray lines show the model used to summarize each distribution.

there were an unbounded number of processors available, and in general it is not possible to derive this information by observing the execution of the program. Kumar [7] has instrumented Fortran programs to perform run-time data flow analysis and approximate their potential parallelism. But this analysis requires programs to have a certain structure—it cannot analyze existing applications. In [2] we proposed a technique for estimating the average parallelism of a program by observing its speedup on a range of cluster sizes. But this approach is based on simplifying assumptions about the nature of parallel overheads, so the estimated values may or may not truly reflect the internal structure of the programs.

In the accounting data we have from SDSC and CTC, we do not have information about the average parallelism of jobs. On the other hand, we do know the cluster size the user chose for each job, and we hypothesize that these cluster sizes, in the aggregate, reflect the parallelism of the workload. This hypothesis is based on the following justification:

- The parallelism of a job is usually a function of the problem size. Larger problems generally have more available parallelism.
- Users often submit jobs with problem size and cluster size chosen such that the turnaround time is less than some constant time limit. Thus, the cluster size and the problem size tend to increase together [6].
- In the aggregate, we expect the shape of the distribution of chosen cluster sizes to reflect the shape of the distribution of average parallelism.

Figure 4 shows the distribution of cluster sizes for the workloads from SDSC and CTC. In both cases, almost all jobs have cluster sizes that are powers of two. Neither the Intel Paragon nor the IBM SP2 *require* power-of-two cluster sizes, but in both cases the interface to the queueing system suggests powers of two and few users have any incentive to resist the combination of suggestion and habit. We hypothesize that the step-wise pattern in the distribution of cluster sizes reflects this habit and not the true distribution of A. To derive a smoother distribution more suitable for our workload model, we estimated linear approximations of the observed distributions. The gray lines in the figure are the fitted lines.

Based on these observations, the distribution of A in our workload is uniform-log with parameters $min = 1$ and $max = N$, where N is the number of processors in the system. This model fits the observed data well, except that both workloads contain significantly more sequential jobs than the model. But this surplus is disappearing as the workloads mature; in the past two months, the fraction of sequential jobs at SDSC was 21%, down from over 40% a year ago. At CTC this fraction is also falling as more parallel codes stabilize and begin production runs. Thus more recent workloads match our model well (the fraction of sequential jobs in our simulations is $\sim 17\%$).

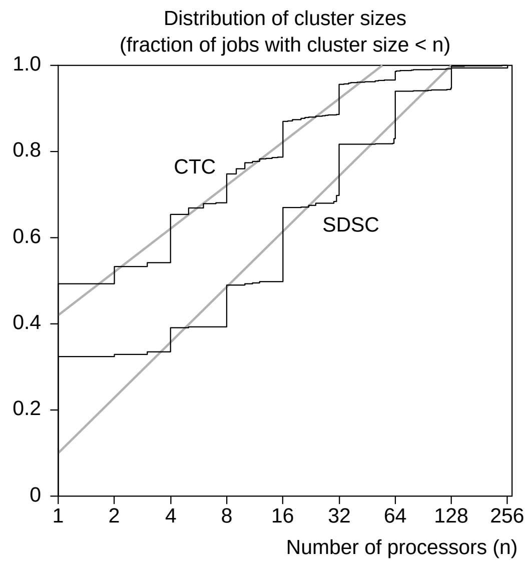


Figure 4: Distribution of cluster sizes for the workloads from SDSC and CTC. The gray lines are the fitted distributions we used for our workload model.

3.3 Distribution of variance in parallelism

In general there is no way to measure the variance in potential parallelism of existing codes explicitly. In [2] we proposed a way to infer this value from observed speedup curves. To test this technique, we collected speedup curves reported for a variety of scientific applications running on a variety of parallel computers. We found that the parameter σ, which approximates the coefficient of variance of parallelism, was typically in the range 0–2, with occasional higher values.

Although these observations provide a range of values for σ, they do not tell us its distribution in a real workload. For this study, we use a uniform distribution between 0 and 2. We found that the specific shape of this distribution has no effect on the relative performance of the allocation strategies we evaluate.

4 Simulations

To evaluate scheduling strategies for parallel applications, we used the models in the previous section to generate workloads, and then used a simulator to construct schedules for each workload according to each strategy. We compare these schedules using several summary statistics as performance metrics (see Section 4.1).

Queueing models and simulations are often used to study systems that are in *steady state* or *statistical equilibrium*; in other words, systems in which the arrival rate equals the departure rate. Our simulation is intended to study transient effects; that is, the behavior of systems during changes in either the arrival or service rate. The transient behaviors we are interested in have been observed in several supercomputing environments:

- In the early morning there are few arrivals, system utilization is at its lowest, and queue lengths are short.
- During the day, the arrival rate exceeds the departure rate and jobs accumulate in queue. System utilization is highest late in the day.
- In the evening, the arrival rate falls but the utilization stays high as the jobs in queue begin execution.

Previous work has described these daily patterns on the Intel iPSC/860 at NASA Ames and the Paragon at SDSC [4] [15].

To model these variations, we divided each simulated day into two 12-hour phases: during the "day-time" phase, jobs arrive according to a Poisson process and either begin execution or join the queue, depending on the state of the system and the current scheduling strategy. During the "night-time" phase, no new jobs arrive, but the existing jobs continue to run until all queued jobs have been scheduled.

Figure 5 shows one of the schedules generated by our simulator. Hour 0, at the top of the page, is roughly 8am; time proceeds down the page. The length of each job indicates its duration. The width of each job is proportional to its cluster size. The jobs have been arranged for visual clarity rather than according to the topology of their allocations; thus, the width of the figure is greater than the number of processors in the system, and empty spaces do not necessarily indicate idle processors. Jobs which were queued have a black tail representing their queue time. The night-time interval, during which there are no additional arrivals, is shaded.

We choose the day-time arrival rate in order to achieve a specified offered load, ρ. We define the offered load as the total sequential load divided by the processing capacity of the system: $\rho = \lambda \cdot E[L]/N$, where λ is the arrival rate (in jobs per second), $E[L]$ is the average sequential lifetime (271 minutes in our simulations), and N is the number of processors in the system (64 in our simulations). We observe that the number of jobs per day is between 80 (when $\rho = 0.5$) and 160 (when $\rho = 1.0$).

4.1 Metrics

As jobs enter and leave the system, we collect the following statistics:

Figure 5: A typical daily schedule, with offered load $\rho = 0.6$, using the ASP strategy (see Section 5.4.

Load average : The nominal load in the system at any time is the fraction of processors that are busy (allocated to a job). The load average is the nominal load averaged over time. Because the jobs in this workload have sublinear speedups, the total allocated time, T, exceeds the sequential lifetime, L, whenever $\sigma > 0$ and the cluster size is greater than 1. Thus, the measured load may exceed the offered load.

Utilization : Utilization takes into account not only how many processors have been assigned to jobs, but also the efficiency with which those jobs are running. Efficiency is the ratio of speedup to cluster size; utilization is efficiency averaged over processors and time. In most real systems the efficiency of jobs, and therefore the utilization of the system, are not known.

Average turnaround time : The turnaround time is the time between the arrival and completion of a job; *i.e.* the sum of its queue time and its run time.

Average slowdown : Slowdown is the ratio of turnaround time to the shortest possible turnaround time, as if the job had run on a dedicated machine. In other words,

$$\text{slowdown} = \frac{\text{queue time} + R(n)}{R(N)} \tag{5}$$

where $R(n)$ is the run time on the allocated cluster size, n, and $R(N)$ is the hypothetical run time on all N processors. Slowdown is a useful performance metric because it gives equal weight to all jobs regardless of length, whereas average turnaround time tends to be dominated by long jobs. Also, slowdown may better represent users' perception of system performance, since it measures delays relative to job duration. For example, a long queue time is more acceptable for a long job than for a short one. Slowdown captures this implicit cost function.

5 Allocation Strategies

Scheduling strategies consist of a *queueing strategy* that chooses which queued job to run and an *allocation strategy* that chooses how many processors to allocate to each job.

The queueing strategy we consider is first-in-first-out (FIFO). The advantages of FIFO are predictability (it is easier to estimate when a queued job will begin execution) and avoidance of starvation (there is no danger of stranding a large job in queue while smaller, later arrivals run). The disadvantages are possibly lower utilization (a job at the head of the queue might leave processors idle waiting for a large cluster) and large slowdowns (giving priority to short jobs would reduce the average slowdown over all jobs).

In future work we would like to evaluate some of the non-FIFO strategies that have been proposed. For example, Ghosal *et al.* [5] propose a "First Fit" strategy, in which the system chooses the first queued job whose chosen cluster size is smaller than the number of idle processors. Chiang *et al.* [1] have shown that a similar strategy is most effective if there is a correlation between duration and cluster size; in this case giving priority to small jobs tends also to give priority to short jobs. The workloads we observed show a strong correlation between cluster size and total allocated time: at SDSC the coefficient of correlation is 0.6; at CTC it is 0.3. This correlation does not affect the FIFO strategies we are evaluating, but it suggests that non-FIFO strategies are promising.

The following sections describe the allocation strategies we consider. None of these strategies is work-conserving, meaning that in some circumstances processors may be left idle that could have been allocated to a job. Rosti *et al.* [9] show that non-work-conserving strategies are best if the workload contains jobs with limited parallelism, if the arrival process is bursty, or if the distribution of job lifetimes is highly variable. The workloads we observed meet all these criteria.

5.1 PWS

Several authors [3][5] have proposed the idea that the optimal cluster size for a job maximizes the power, Φ, defined as the product of speedup and efficiency. Since efficiency, e, is defined as S/n, Φ is S^2/n. The cluster size that maximizes Φ is called the *processor working set*, and hence this strategy is called PWS.

Most systems do not have enough information about applications to find their processor working set. But for our hypothesized speedup model (Equations 1 and 2) we can find the point of maximum power analytically. For the low-variance model,

$$pws = \begin{cases} A & \sigma \leq 2A/(3A-1) \\ \frac{\sigma(A-1/2)}{1-\sigma/2} & \sigma \geq 2A/(3A-1) \end{cases} \tag{6}$$

For the high variance model $pws = (A\sigma + A - \sigma)/\sigma$. Figure 6 shows pws for a range of values of σ (with A fixed at 64). For values of σ less than $\sim 2/3$, the point of maximum power is $pws = A$, which is in accord with the heuristic that the number of processors allocated to a job should be equal to its average parallelism. It also agrees with the colloquial interpretation of a "knee," *i.e.* a change in the slope of the speedup curve (see Figure 2).

But as the value of σ approaches 1, pws increases quickly to $2A-1$. For $\sigma > 1$, pws decreases and approaches $A-1$ asymptotically. This result is surprising because it violates the intuition that the optimal allocation for a job

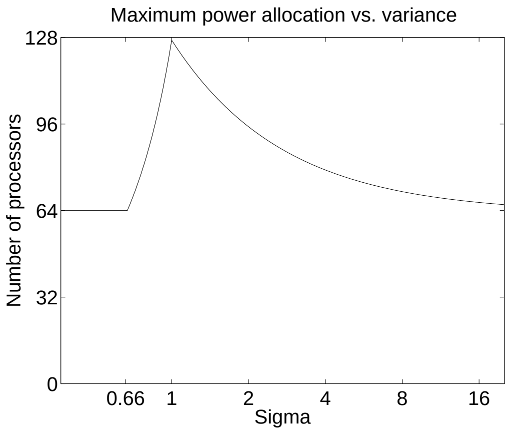


Figure 6: The optimal allocation for a range of values of σ. The average parallelism, A, is 64.

should decrease as the variance in parallelism increases, which is one of the assumptions of Sevcik's allocation strategy (see below). Our results (Section 6) show that neither intuition is correct: the pws is not an optimal allocation and decreasing allocations for jobs with high σ is not useful.

Eager *et al.* [3] prove that for any parallelism profile, pws is bounded by $A/2 \leq pws \leq 2A - 1$. Within our speedup model, pws never approaches the lower bound, but it reaches the upper bound when $\sigma = 1$, a value we have found is not uncommon for real applications.

5.2 AVG and MAX

Several authors have proposed the idea that a job should be allocated a number of processors equal to A, the average parallelism. Eager *et al.* [3] suggest that this strategy should do well because A is always within a factor of two of pws. Interestingly, we find that this strategy, AVG, performs *better* than PWS, which it is supposed to approximate.

Another strategy suggested by the speedup curves in Figure 2 is MAX, which allocates enough processors to achieve the maximum speedup for the job; in other words, it allocates the minimum n such that $S(n) = A$. For our speedup model:

$$max = \begin{cases} A & \sigma = 0 \\ 2A & 0 < \sigma \leq 1 \\ A + A\sigma - \sigma & \sigma \geq 1 \end{cases} \tag{7}$$

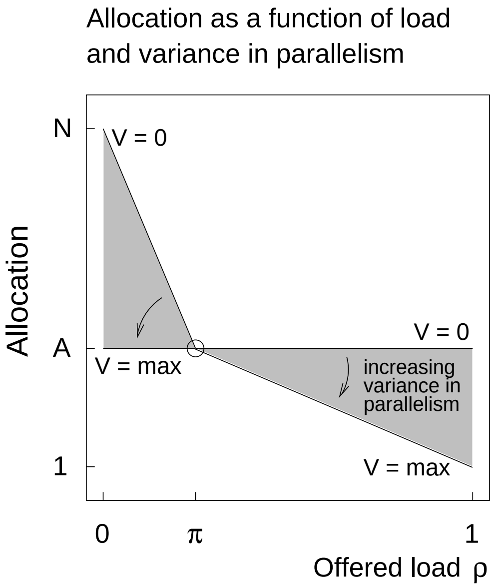


Figure 7: The relationship proposed by Sevcik between average parallelism, A, variance of parallelism, V, system load, ρ and cluster size. The value of V_{max} is specific to the parallelism model used by Sevcik; in general, the variance in parallelism is not bounded.

5.3 Sevcik's allocation strategy

Sevcik [13] has proposed an allocation strategy that chooses cluster sizes as a function of the average parallelism of the job, A, the variance in parallelism, V, and the offered load, ρ. This strategy is based on the intuition that cluster sizes should be large when A is large, but should get smaller as V or ρ increases. Sevcik calls this strategy $A+$; we call our variation of it SEV. The primary distinction is that we use the parameter σ where Sevcik uses V. The reason we use σ as the measure of variance of parallelism is that V increases with A; σ, which approximates the coefficient of variation, is independent of A.

Figure 7 shows this allocation strategy graphically: (1) for very low loads, cluster sizes vary from N for jobs with low V to A for jobs with high V, (2) at some moderate load π, all jobs are allocated A processors, and (3) for high loads, cluster sizes vary from A for jobs with low V to 1 for jobs with high V.

The value of π is the load at which the optimal allocation per process is equal to A. Sevcik chooses the value of π based on a queueing model with an exponential distribution of job lifetimes ($CV = 1$); for this workload π is roughly 0.25. As CV increases, we expect π to decrease, reducing cluster sizes in order to lessen the danger of assigning a large cluster to a very long job. Some studies

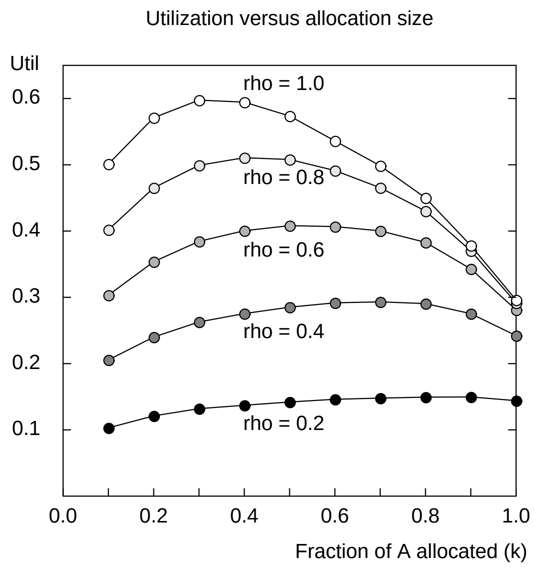


Figure 8: Each curve shows the average utilization of a simulated 64-processor system with offered load ρ. The parameter k is the fraction of A processors assigned to each job. Each data point is the average of 60 simulated days.

have used the same value of π for workloads with $CV > 1$. Part of the reason these studies report poor performance for this strategy may be that π is not calibrated.

To find the value of π for our workload, we ran our simulator with a range of offered loads ρ and with a range of values of a parameter, k, which is the fraction of A processors allocated to each job. Thus, the cluster size for each job is kA. We expect π to be the value of ρ for which the optimal allocation is A; in other words, the offered load for which $k = 1$ yields the best performance.

Figure 8 shows the average utilization of a simulated 64-processor system with values of $\rho = 0.2$, 0.4, 0.6, 0.8 and 1.0, and values of k between 0.1 and 1.0. Figure 9 shows the optimal value of k chosen for each value of ρ. The gray line indicates the hypothetical linear trend.

We conclude: (1) one of Sevcik's assumptions—that the optimal allocation size should decrease linearly as the load increases—is at least approximately correct, and (2) the value of π is at or near zero. It is difficult to make the latter claim precise, since for low loads ($\rho < 0.4$) the value of the parameter has almost no effect on performance, and hence "optimal value" has little meaning.

With $\pi = 0$, the allocation strategy in Figure 7 can be simplified as in Figure 10. But before we can implement this method, there is one other free parameter that needs to be resolved—the maximum variance. In Sevcik's work-

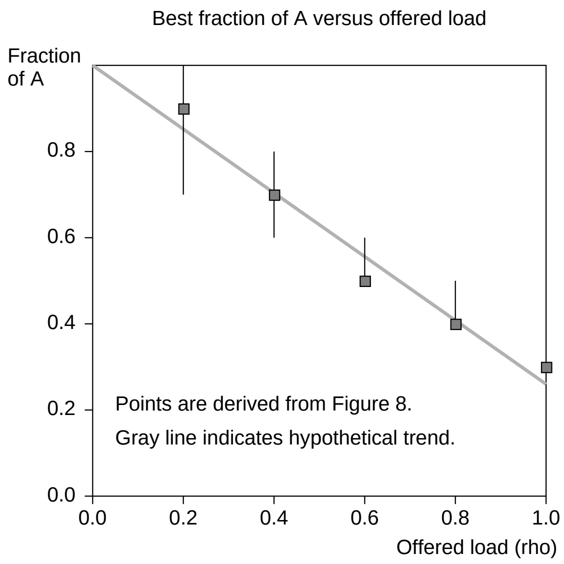


Figure 9: For each value of ρ we have chosen the value of k that maximizes utilization; the vertical error bars show the range of values where this maximum may fall. The gray line extrapolates until $k = 1$, which is approximately at $\rho = 0$.

load model there was an upper bound on a job's variance in parallelism. For our workloads, the parameter we use to measure variance, σ, is unbounded in theory, but we have imposed the bound $\sigma \leq 2$. This bound is based on our estimates of σ for a variety of speedup curves reported by application implementors.

Sevcik also proposes a variant strategy, $A+\&mM$, which imposes a maximum and minimum cluster size on each job. In our implementation, the minimum size is 1 and, since $\pi = 0$, the maximum cluster size is A. In Section 6.4 we discuss the impact of jobs with minimum cluster sizes greater than 1.

5.4 ASP

The strategies we have described so far have been based on the idea that the optimal allocation for a job depends on the characteristics of the application. For purposes of comparison, we also consider a form of *adaptive static partitioning* (ASP) based on the notion that it is preferable to have as many jobs running as possible, rather than waiting in queue, even if they run on small cluster sizes.

Toward this goal, the ASP strategy takes the number of free processors (at the time of an arrival or completion) and divides them evenly among the jobs in queue. In our implementation, no job is allocated more processors than it

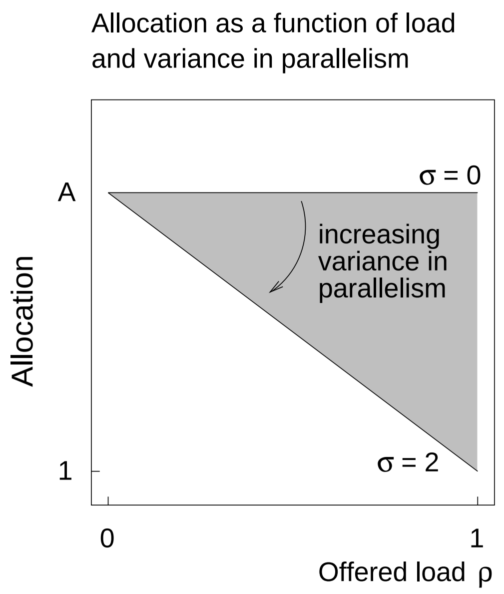


Figure 10: A simplified version of Figure 7 with $\pi = 0.0$. For high loads, the cluster size varies from A to 1, depending on the coefficient of variation of parallelism, σ.

can use (see Equation 7). Chiang *et al.* [1] study a similar strategy, called ASP-max, but their maximum cluster size is a system parameter that does not depend on application characteristics. For workloads in which parallelism does not vary greatly from job to job, they report that ASP-max is better than PWS, AVG, or SEV. For workloads with large variance in A, though, their results are consistent with ours — the performance of ASP is relatively poor. We explain this result in more detail in Section 6.3.

6 Results

In this section, we evaluate the performance of the proposed allocation strategies. We consider two variations of each strategy:

Stubborn strategies : These strategies choose an "ideal" number of processors for each job and will allocate no fewer. If there are not enough free processors, the job waits in queue.

Greedy strategies : These jobs treat the "ideal" cluster size only as an upper bound; if fewer processors are available, the job begins execution immediately on the smaller cluster.

We find that greedy strategies are better than stubborn strategies by all metrics. In Section 6.4 we consider hybrid strategies that can allocate fewer than the ideal number of processors (greedy), but which nevertheless impose some lower bound on cluster sizes (stubborn). We find that the pure greedy strategy is better than hybrid strategies with even moderate minimum cluster sizes.

Since ASP is inherently a greedy strategy, there is no stubborn version. We include ASP in both groups to make it easier to compare their performance.

6.1 Stubborn strategies

Figure 11a shows the utilization of a simulated 64-node system with the stubborn version of the allocation strategies. We consider only offered loads greater than 0.5 because as we saw in Figure 8, for lower loads allocation strategy has little effect on performance. Also, the systems we observed operate at an average load (fraction of busy processors) between 0.6 and 0.8.

The performance of ASP is reasonably good; utilization ranges from 0.4 to 0.6 and mean turnaround times range from 5000 seconds when $\rho = 0.5$ to 9000 seconds when $\rho = 1.0$.

The performance of the stubborn strategies is abysmal. Even SEV, which achieves reasonably good utilization, yields turnaround times twice as long as ASP's. The other methods are even worse: turnaround times under AVG are 5–8 times longer than ASP's, under PWS 13–17 times longer, and under MAX 17–21 times longer. The dominant term in the excess is queue time; jobs are spending too much time in queue waiting for their "ideal" cluster sizes, rather than running immediately on what's available.

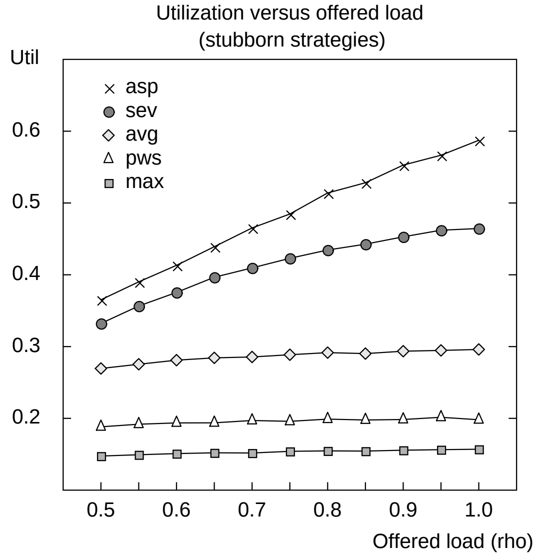
We conclude that is that it is infeasible to choose a cluster size for a job solely on the basis of its parallelism profile. The allocation strategy must be flexible enough to adapt to system load by allocating smaller-than-ideal cluster sizes when necessary.

6.2 Greedy strategies

In this section, we consider strategies that use application characteristics to choose a *maximum* cluster size, but that require no *minimum* cluster size. These strategies are *greedy* in the sense that if there are any idle processors, the job at the head of the queue will allocate them immediately and begin execution, even if the number of processors is much less than the application's available parallelism.

The advantages of greedy strategies are higher utilization and lower queue times. The potential disadvantage is that long jobs with high parallelism might be allocated far too few processors, resulting in high turnaround times. Figure 11b shows that the short queue times that result from small cluster sizes clearly outweigh the longer run times. The greedy strategies are better than the

a) stubborn strategies



b) greedy strategies

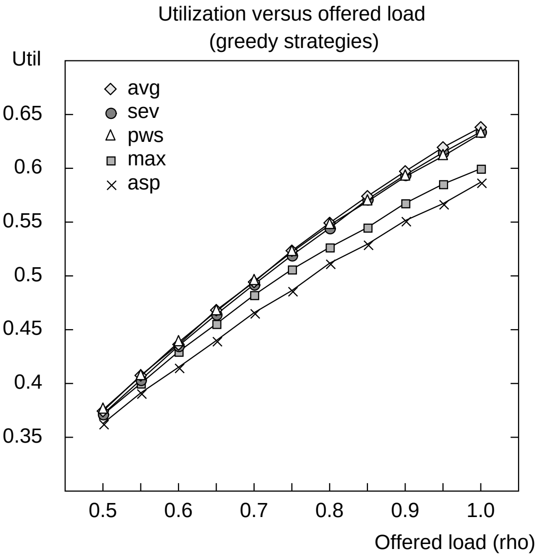


Figure 11: Utilization as a function of offered load for (a) the stubborn version and (b) the greedy version of each strategy. Each data point is the average of 1200 day-long simulated runs. The size of the data markers (squares, circles, *etc.*) is approximately two standard deviations. Thus, non-overlapping data markers indicate statistically significant variations.

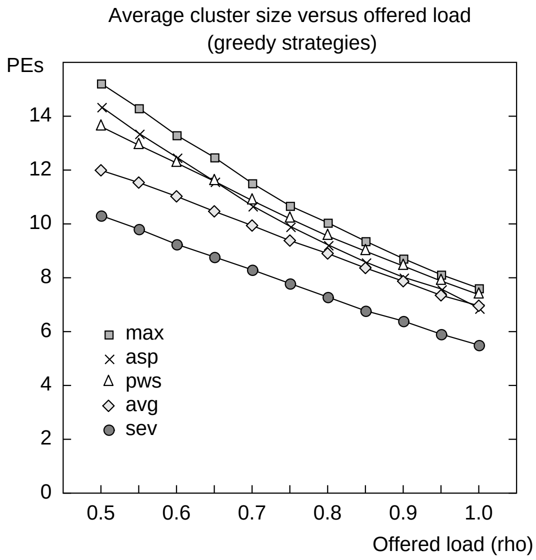


Figure 12: Average cluster size for each strategy for a range of offered loads. In each case, the cluster size decreases as load increases. Each data point is the average of 120 day-long simulated runs.

stubborn strategies by every metric. All of the greedy strategies perform *better* than ASP, as opposed to the stubborn versions, all of which are significantly worse.

The primary reason for this improvement is that these strategies are load-sensitive; that is, as the offered load increases, the average cluster size decreases. Figure 12 shows the average cluster size allocated by each strategy for a range of loads. ASP's curve is steepest; *i.e.* it is most sensitive to load. In the next section, we suggest that they reason for ASP's relatively poor performance is that it is *too* load-sensitive.

Although the differences among the greedy strategies are small, it is interesting to see that the performance of PWS is marginally worse than that of AVG (the differences in turnaround times and slowdowns are more pronounced). Thus we conclude that the point of maximum power (the "knee" of the speedup curve) is not an optimal allocation.

6.3 A closer look

In order to investigate the relationships among various performance metrics, we set $\rho = 0.75$ and examine the proposed strategies in more detail. Table 1 shows several summary statistics for each strategy, averaged[2] over $\sim$ 15000 jobs (120 simulated days).

Table 1

	measured load average	util-ization	average turnaround time	average queue time	average cluster size (CV)	90th percentile slowdown
SEV	.64	.52	5858	204	7.8 (1.04)	11.6
AVG	.70	.52	5782	372	9.4 (1.07)	35.3
PWS	.73	.52	6017	566	10.2 (1.10)	77.8
MAX	.81	.51	6597	1115	10.7 (1.11)	249
ASP	.77	.49	7510	402	9.9 (1.24)	63.6

Strategies that allocate large cluster sizes tend to have high measured loads—they leave fewer idle processors. But load only measures how many processors are busy, not how effectively they are being used. Since utilization also considers the efficiency of running jobs, it is a better indicator of performance than load.

It is often observed that there is a potential conflict between maximizing system utilization and minimizing average turnaround time. For the strategies we considered, though, the two metrics are consistent; whatever strategy resulted in the highest utilization also yielded the lowest average turnaround times. On the other hand, slowdown and turnaround time are not always consistent; for example, SEV suffers somewhat longer turnaround times than AVG, but yields much better slowdowns. Since slowdown reflects users' perception of system performance, it might be preferable to choose a strategy that achieves minimal slowdowns, even with (moderately) longer turnaround times.

There is a clear relationship between average cluster size and average queue time. The strategies that allocate the fewest processors per job have the shortest queue times. But the tradeoff is that smaller cluster sizes result in longer run times. The proposed strategies operate at different points along this tradeoff:

- SEV allocates the smallest cluster sizes, resulting in the shortest queue times, but incurs the longest run times.
- AVG and PWS reduce run times by allocating more processors per job, but this benefit is almost exactly balanced by greater queue times; thus, the utilization of the three strategies is the same and the average turnaround times are similar.
- On the other hand, MAX allocates so many processors per job that the additional queue time overwhelms the saved run time. Utilization for this strategy is lower and turnaround times are higher.

[2] We report the 90th percentile of slowdown rather that its mean because the distribution of slowdowns is long-tailed, and for such distributions order statistics (median and other percentiles) are more robust, and hence more meaningful, than moment statistics (mean, variance, *etc.*).

Of the methods with the highest utilization, SEV has by far the lowest slowdowns. One of the reasons SEV does well is that it takes load into account explicitly and reduces cluster sizes when load is high. The other strategies achieve a similar effect implicitly, by using the number of free processors as a proxy for the current load, but this implicit load-sensitivity is less precise—these methods sometimes allocate too many processors during a momentary lull. Figure 5 provides an example. At Hour 10, a large job arrives during a lull and allocates a large fraction of the machine. While that job runs (for over two hours) 16 smaller jobs arrive and wait in queue for the large job to complete.

Of the strategies proposed, only SEV would avoid this mistake by restricting the cluster size of the large job in anticipation of future arrivals. Of course, the disadvantage of SEV is that it is necessary to know the offered load on the system *a priori*, or estimate it dynamically. Our simulation of this strategy assumes optimistically that the offered load is known exactly.

We have discussed how a momentary lull causes some of the strategies to allocate too many processors, but there is also the danger that a momentary surge in queue length will cause the system to allocate too few processors. Most of our strategies avoid this type of error by allowing the job at the head of the queue to choose a cluster size regardless of queue length. ASP is the only strategy that takes queue length into account, and it suffers for it.

The reason ASP performs relatively poorly is that it is most likely to make extreme allocations (very large or very small) during short-term variations in load (lulls and surges). To support this claim we examined the variance in cluster size for each strategy. As expected, ASP has a higher CV of cluster size than any of the other strategies (1.24 compared to a range of 1.04 to 1.11). We conclude that the success of a scheduling strategy lies in its ability to adjust to long-term changes in load (as in the daily cycle we model) without over-reacting to short-term variations.

6.4 Hybrid strategies

We have presented several ways to use application characteristics (A and σ) to derive cluster sizes, and suggested two ways to use those cluster sizes as part of an allocation strategy. In the stubborn version, the derived cluster size is both the minimum and maximum number of processors a job can allocate. In the greedy version, it only determines the maximum size; most of the time the cluster size depends on the number of free processors in the system. We have shown that the greedy versions, which impose no minimum on cluster size, yield much better performance by all metrics.

It is natural to wonder whether there is a compromise between these extremes that might do even better, perhaps by imposing a minimum cluster size for each job. To evaluate these hybrid strategies, we introduce a parameter, c, which is the fraction of the maximum cluster size that is guaranteed. In other words, if the maximum cluster size for the job at the head of the queue is n, then the job will not run until at least cn processors are available. Thus, when $c = 0$ the strategies are greedy and when $c = 1$ they are stubborn.

Figure 13 shows that imposing a minimum cluster size does not improve any of the proposed strategies. For some of the strategies, a low minimum improves utilization somewhat, but not significantly. Imposing a large minimum degrades performance greatly, reducing utilization and increasing turnaround times. We conclude that the greedy strategies are best.

Real applications that use a lot of memory may require a minimum cluster size because the performance of the memory hierarchy degrades as memory demand per processor increases. Our results suggest that the presence of jobs with moderate processor requirements does not degrade the performance of the system, especially with SEV or AVG, both of which can tolerate $c = 0.4 - 0.6$ before their performance deteriorates significantly. On the other hand, Setia [10] has examined the effect of memory constraints on cluster size, and finds that wide-ranging minimum cluster sizes significantly impair performance. Parsons and Sevcik [8] propose a model for these memory bounds and evaluate scheduling strategies that coordinate the allocation of memory and processors.

6.5 Sensitivity to variance

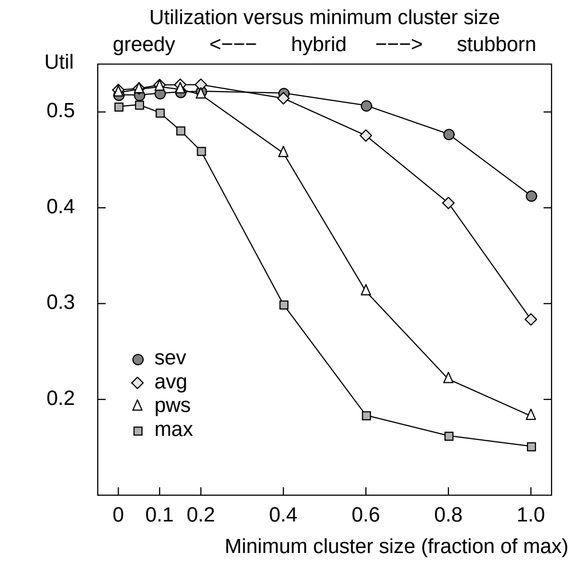
In Section 6.2 we argued that sensitivity to load is more important than sensitivity to application characteristics, but that application characteristics were useful nevertheless. In this section, we show that it is primarily A that is useful in choosing cluster sizes, and that sensitivity to variance is not important.

Figure 14 shows the relationship between cluster size and σ for each strategy (with A fixed). The two strategies that are most sensitive to variance are SEV, which decreases the cluster size linearly as σ increases, and PWS, which has the counter-intuitive relationship derived in Section 5.1.

We would like to investigate whether the performance of these strategies depends on these particular relationships, or whether similar, variance-insensitive strategies might do as well. We consider two new strategies, 3/2 AVG and Simplified SEV, that are insensitive to variance (σ), but which are designed to allocate the same number of processors per job on average. 3/2 AVG is the same as AVG except that the maximum cluster size for each job is $3/2A$. Simplified SEV is the same as SEV except that for all jobs σ is considered to be 1. Figure 15a shows that these strategies allocate the same cluster sizes, on average, as their variance-sensitive counterparts.

Figure 15b shows that the average turnaround time for Simplified SEV is not significantly different from that of SEV; in other words, variance-sensitivity has no effect on the performance of SEV. This contradicts one of the underlying assumptions of Sevcik's method, that cluster size should decrease as variance increases.

On the other hand, the strange relationship between σ and cluster size used by PWS *does* have a significant impact on performance. The variance-insensitive a)



b)

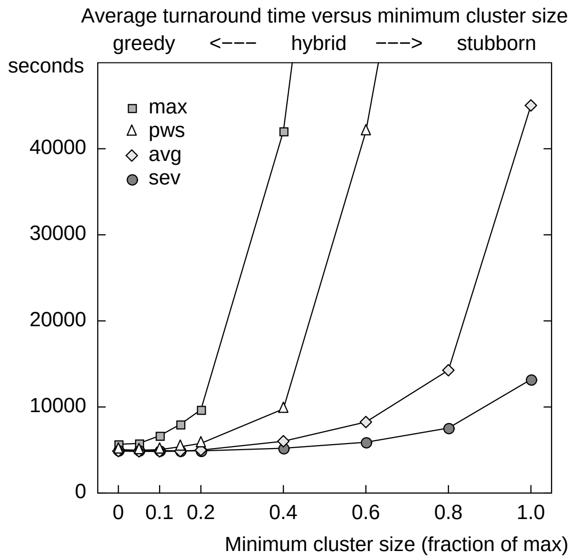


Figure 13: System utilization (a) and average turnaround time (b) as a function of the minimum cluster size. Each data point is the average of 120 simulated days, with offered load $\rho = 0.75$.

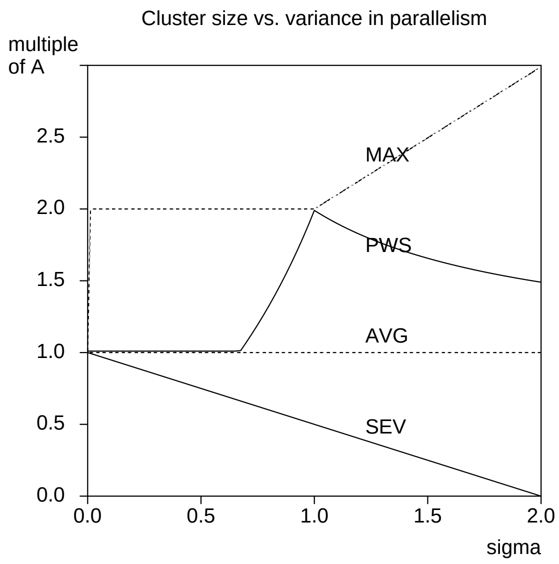


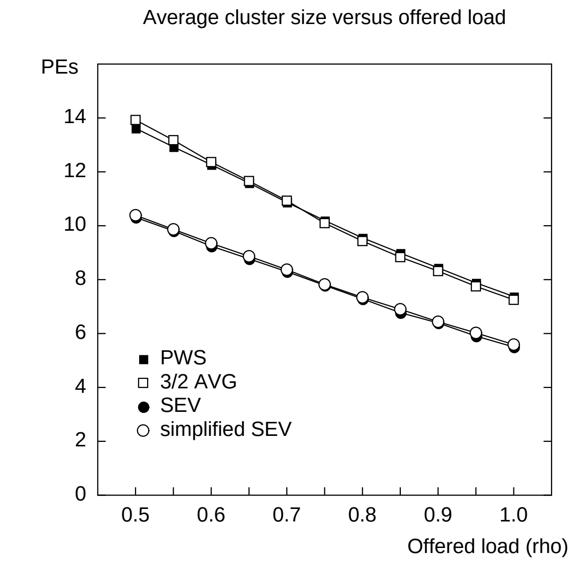
Figure 14: The relationship between cluster size and variance in parallelism (σ) for each of the proposed strategies.

version of this strategy does significantly worse by all metrics. Although this result is surprising, it has no practical importance, since AVG, which is variance-insensitive, performs better than even the variance-sensitive version of PWS. Thus we conclude that knowing variance in parallelism for each job is not useful for allocation.

7 Conclusions

- One of the strategies recommended in other studies (ASP) did not perform well for our workload. We show that this policy is too sensitive to short-term variations in system load. It performs worse than simple FIFO strategies that use application characteristics to bound cluster sizes.
- The application characteristics we examined, average and variance of parallelism, are useful for choosing the upper bound on cluster size (and thereby imposing a lower bound on efficiency). We found, though, that strategies that considered variance of parallelism were no better than those that considered only average parallelism.
- The processor working set, or "knee" of the speedup curve, is not an optimal processor allocation.

a)



b)

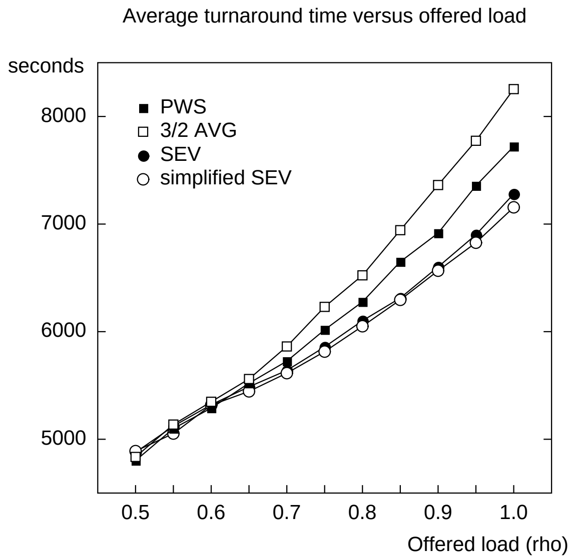


Figure 15: Average cluster size (a) and turnaround time (b) as a function of offered load, comparing two of the proposed strategies with variance-insensitive versions. Each data point is the average of 120 simulated days.

- It is not necessary to set maximum cluster sizes precisely. There is a tradeoff between large clusters/long queues and small clusters/short queues, but within a wide range, system performance (measured by average turnaround time) does not vary greatly.

- Of several strategies with equivalent turnaround times, Sevcik's strategy, which uses *a priori* knowledge about system load to restrict cluster sizes, yielded the lowest slowdowns. Depending on what metric matters most to users, this strategy might be the best choice.

- One of Sevcik's underlying assumptions — that cluster sizes should decrease linearly as load increases — has been validated. The other underlying assumption — that jobs with a more variable parallelism profile should allocate fewer processors — has been contradicted.

- Lifetimes for batch jobs on supercomputers are distributed uniformly in log space. Our *uniform-log model* is useful for summarizing these distributions and generating simulated workloads. The observed distributions have coefficients of variation in the range 2–4.

7.1 Future work

When a job arrives at the head of the queue, there is a conflict between the system, which would like the job to begin execution as soon as possible, and the job, which might enjoy a shorter turnaround time by waiting until a larger cluster size is available. We would like to investigate this conflict and address these questions:

- How much do individual jobs benefit by leaving processors idle and waiting for larger clusters?

- How much do these stubborn jobs hurt the system as a whole by increasing the queue times of other jobs? This paper has shown that a naive stubborn policy can severely degrade system performance.

- In a real system, is it tenable for the scheduler to make decisions that are contrary to the immediate interests of users, or will users subvert such a system?

Our goal is to find an allocation policy that maintains acceptable overall performance without creating incentives for users to manipulate the system for their own benefit.

References

[1] Su-Hui Chiang, Rajesh K. Mansharamani, and Mary K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1994.

[2] Allen B. Downey. A model for speedup of parallel programs. Submitted for publication., 1996.

[3] Derek L. Eager, John Zahorjan, and Edward L. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, March 1989.

[4] Dror G. Feitelson and Bill Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 215–227, 1995.

[5] Dipak Ghosal, Giuseppe Serazzi, and Satish K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17(5):443–453, May 1991.

[6] John L. Gustafson. The consequences of fixed time performance measurement. In *Proceedings of the Twenty-Fifth Hawaii International Conference on system Sciences, Vol. III*, January 1992.

[7] Manoi Kumar. Measuring parallelism in computation-intensive scientific/engineering applications. *IEEE Transactions on Computers*, 37(9):1088–1098, September 1988.

[8] Eric W. Parsons and Kenneth K. Sevcik. Coordinated allocation of memory and processors in multiprocessors. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 57–67, May 1996.

[9] Emilia Rosti, Evgenia Smirni, Lawrence W. Dowdy, Giuseppe Serazzi, and Brian M. Carlson. Analysis of non-work-conserving processor partitioning policies. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 101–111, 1995.

[10] Sanjeev. K Setia. The interaction between memory allocation and adaptive partitioning in message-passing multicomputers. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 89–99, 1995.

[11] Sanjeev K. Setia and Satish K. Tripathi. An analysis of several processor partitioning policies for parallel computers. Technical Report CS-TR-2684, University of Maryland, May 1991.

[12] Sanjeev K. Setia and Satish K. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proceedings of the Internationsal Workshop on Modeling and Simulation of Computer and Telecommunications Systems (MASCOTS)*, January 1993.

[13] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. *Performance Evaluation Review*, 17(1):171–180, May 1989.

[14] Evgenia Smirni, Emilia Rosti, Lawrence W. Dowdy, and Giuseppe Serazzi. Evaluation of multiprocessor allocation policies. Technical report, Vanderbilt University, 1993.

[15] Kurt Windisch, Virginia Lo, Dror Feitelson, Bill Nitzberg, and Reagan Moore. A comparison of workload traces from two production parallel machines. In *6th Symposium on the Frontiers of Massively Parallel Computation*, 1996.