Processor Saving Scheduling Policies for Multiprocessor Systems

Emilia Rosti, *Member*, *IEEE*, Evgenia Smirni, *Member*, *IEEE Computer Society*,
Lawrence W. Dowdy, *Member*, *IEEE Computer Society*,
Giuseppe Serazzi, *Member*, *IEEE Computer Society*,
and Kenneth C. Sevcik, *Member*, *IEEE*

Abstract—In this paper, processor scheduling policies that "save" processors are introduced and studied. In a multiprogrammed parallel system, a "processor saving" scheduling policy purposefully keeps some of the available processors idle in the presence of work to be done. The conditions under which processor saving policies can be more effective than their greedy counterparts, i.e., policies that never leave processors idle in the presence of work to be done, are examined. Sensitivity analysis is performed with respect to application speedup, system size, coefficient of variation of the applications' execution time, variability in the arrival process, and multiclass workloads. Analytical, simulation, and experimental results show that processor saving policies outperform their greedy counterparts under a variety of system and workload characteristics.

Index Terms—Multiprocessor systems, processor scheduling, processor saving algorithm, work conserving, Markov analysis, performance evaluation.

1 INTRODUCTION

PARALLEL systems consisting of large numbers of processors are readily available in production and research environments. In general, however, few single applications can fully exploit the considerable computational power offered by these systems due to factors such as: diminishing return from the assignment of additional processors to parallel applications, limited maximum application parallelism, and fluctuations in the submission frequency and in the execution time of applications. Such factors provide the motivation for many parallel systems to allow multiprogramming to improve their performance.

A common goal of processor scheduling policies is to maximize system throughput or to minimize response time. In uniprocessor systems, this is accomplished by allocating the processor as soon as possible. Such an approach is optimal for general purpose multiprogrammed uniprocessor systems where jobs have nonpreemptive priorities, since keeping the processor idle when there is unfinished work degrades the average system performance.[1] In multiprocessor systems, different approaches are viable. The class of policies investigated in this paper, that we call "processor saving" (`p_sav`) policies, deliberately keeps some of the available processors unassigned in the presence of unfinished work for unexpected future events, e.g., sudden bursty arrivals, unusually long execution time, irregular arrival behavior. A preliminary study concerning the potential benefits of `p_sav` policies has appeared in [24]. These policies can be effective because most parallel programs cannot take full advantage of the computational power, due to hardware and software constraints, e.g., system architecture characteristics and limited application parallelism [2]. For some parallel programs, the potential benefit of using extra processors is less than the potential benefit of saving the processors for future arrivals.

In this paper, a hierarchy of processor saving policies for multiprocessor systems is constructed based on the amount of information included in the policy. It is shown that `p_sav` policies may counter-intuitively yield better performance in terms of average response time than their greedy counterparts, i.e., policies that assign all available processors as soon as possible. Conditions under which it is beneficial to use processor saving policies are explored. Workload characteristics, such as fluctuations in the arrival and service processes, are investigated. Other scheduling policies that leave some of the processors idle have appeared in the literature, [5], [21], [1]. However, in these policies, saving processors is not deliberate but, rather, a side-effect of the allocation strategy: Processors are left unassigned when their number is either less than or more than some target allocation.

- *E. Rosti is with the Dipartimento di Scienze dell'Informazione, Università di Milano, Via Comelico 39/41, 20135 Milano, Italy. E-mail: rose@dsi.unimi.it.*
- *E. Smirni is with the Department of Computer Science, College of William and Mary, P.O. Box 8795, Williamsburg, VA 23187-8795. E-mail: esmirni@cs.wm.edu.*
- *L.W. Dowdy is with the Department of Computer Science, Vanderbilt University, P.O. Box 1679, Station B, Nashville, TN 37235. E-mail: dowdy@vuse.vanderbilt.edu.*
- *G. Serazzi is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20131 Milano, Italy. E-mail: serazzi@elet.polimi.it.*
- *K.C. Sevcik is with the Computer Systems Research Institute, University of Toronto, 6 King's College Road, Toronto, Ontario, Canada M5S 1A1. E-mail: kcs@csri.utoronto.edu.*

Manuscript received 19 Feb. 1996; revised July 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 106069.

1. In the presence of real-time or other types of constraints, optimality may be achieved using different strategies. In this paper, we consider only general purpose systems.

The policies considered in this paper are nonpreemptive adaptive policies. This type of policy allows for processor redistribution only at application scheduling time, when the number of allocated processors is computed. Each program is assigned a set of processors on which it runs exclusively until it completes. Nonpreemptive policies are also called static [5], [10], adaptive [21], [16], [19], or run-to-completion [26], [1]. Alternatively, preemptive policies allow executing programs to be interrupted and dynamically reallocated a larger or smaller set of processors. Examples of preemptive policies include gang scheduling [17], [4], [6], time-sharing [11], [25], [9], and dynamic space-sharing [9], [18], [3], [26], [12], [13], [15]. Preemptive policies are optimal from an allocation point of view, since they allow for better resource utilization and can adapt to sudden changes in the workload intensity. However, the complexity of the run time environment for their implementation on an actual system and the overhead of dynamic process and data reconfiguration may outweigh the benefits of a better resource allocation. Nonpreemptive adaptive policies react to sudden workload changes in a slower fashion, thus reducing the potential for thrashing when allocations change frequently. They represent a viable solution with negligible overhead that is easily implemented in actual systems. Moreover, they offer a compromise between the simple, easy to implement, but rigid, static partitioning schemes and the flexible, but overhead prone and complex to implement, dynamic ones. We focus on nonpreemptive schemes because we are interested in investigating policies that can be effectively implemented on actual systems.

This paper is organized as follows. In Section 2, the concept of processor saving is illustrated and a hierarchy of `p_sav` policies presented. These policies are modeled analytically using Markov chains and their performance trade-offs are investigated. A generalization of these policies is presented in Section 3 and performance results obtained from simulation are discussed. Section 4 presents the results of experimentation on an Intel Paragon multiprocessor system under single and multiclass real workloads. Section 5 summarizes the findings and concludes the paper.

2 Markov Analysis

In this section, a technique to uniformly construct and compare `p_sav` policies with incremental amounts of processor saving is presented. The policies are modeled using continuous time Markov chains and performance results are obtained by solving the global balance equations. In order to allow for the analytical solution, we restrict here the partition space of the policies presented in Section 3 to either one-half or all of the system processors. The policies obtained are simple, yet they provide valuable insights about system performance. The general version of the `p_sav` policies is investigated in Section 3 via simulation.

2.1 The Workload

Differences in parallel applications are taken into consideration by means of workloads with different speedups or, equivalently, execution signatures. Several functional forms have been proposed as application execution signatures [18], [23], [1], from which speedup functions can be derived. Let p be the number of processors assigned to an application and $S(p)$ the speedup achieved when the application is executed on p processors. The functional form adopted here for $S(p)$ is

$$S(p) = \frac{1-\alpha^{p}}{1-\alpha}, \tag{1}$$

where

$$\alpha = \frac{S(p+1)-S(p)}{S(p)-S(p-1)} \qquad \alpha \in [0,\ 1) \tag{2}$$

is the ratio of the speedup gain when the number of allocated processors is increased from p to $p + 1$ over the speedup gain when the number of allocated processors is increased from $p - 1$ to p. The ratio α is constant and characterizes the concavity of the speedup curve.

Equation (1) offers a close fit to the experimental curves derived on the Intel Paragon system used for the experimentation in Section 4 and is a convenient analytic form. The speedup curves derived from (1) are monotonically increasing, while, in real systems, speedup may drop or remain flat after a given number of processors. However, the applications considered here are assumed to have sufficient parallelism for the system on which they execute. Reasonably interesting applications for parallel systems are expected to have increasing speedups at least for the system sizes used here.

In the following analysis, highly parallel workloads with linear speedups (i.e., where $\mu_p = p\mu_1$, μ_p being the execution rate when p processors are allocated) and poorly parallel workloads with flat speedup (i.e., where $\mu_p = \mu_1$) are considered. In spite of their limited interest for actual systems, highly and poorly parallel workloads are considered as they represent the best and worst case of application scalability, respectively. Performance bounds are obtained on such workloads. Henceforth, intermediate speedups will be referred to as *concave*m%, where m is the attained percentage of the maximum achievable speedup (linear speedup) for a given system size. Poorly parallel and highly parallel workloads are indicated by *concave*0% and *concave*100%, respectively. The speedup curves used in the following analysis (see Section 2.3) are plotted in Fig. 1 and comprise the two bounding cases, i.e., *concave*100% and *concave*0%, and an intermediate case, i.e., *concave*50%.

Performance trade-offs are investigated with respect to workload characteristics, by changing the speedup concavity across the range of offered load, defined as $\lambda/(P\mu_1)$, where λ is the workload arrival rate, P is the number of system processors, and $1/\mu_1$ is the amount of work required by an application when executed on a single processor. The offered load is changed by varying the job arrival rate λ over the range $[0, P\mu_1]$. Exponential interarrival and service times are assumed as a baseline case (see Section 2.3). Results for nonexponential arrivals are presented in Section 2.4.

The focus of this paper is the relative performance of `p_sav` policies compared with their work-conserving counterparts. Therefore, the performance metric adopted is the average *response time ratio*, that is, the ratio of the average

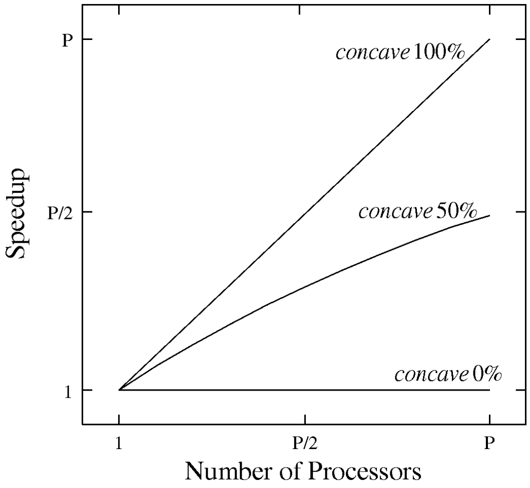


Fig. 1. Speedup curves for the highly parallel *concave*100%, medium parallel *concave*50%, and poorly parallel *concave*0% workloads considered in the Markovian analysis.

response time under a given `p_sav` policy to the average response time under the baseline work-conserving policy.

2.2 The Policies

The allocation policies considered in this paper are nonpreemptive. The decision on how many processors to assign to a job is based only on information available at scheduling time. Jobs are treated as statistically identical and scheduled FCFS, since workload characteristics are assumed to be unknown to the scheduler. Considerable benefits could be derived from knowing parameters such as the job execution time or the number of processors that optimizes a given performance metric since optimal queuing disciplines (e.g., Shortest Job First) or allocation policies could be used. However, in real environments where multiprogrammed parallel systems are operational, little, if any, a priori knowledge about the submitted jobs is available at execution time. Therefore, any implementable allocation policy for such systems cannot rely on information that is usually unavailable. Since we are interested in policies that may be effectively implemented, in this study, we assume no knowledge of workload parameters.

Markovian models of the policies are constructed and solved. To keep such models tractable, the general `p_sav` policies are restricted here to two partitions, the entire system or one-half of it. The Markovian models of the unrestricted version can not be solved analytically, so results of simulation analysis are presented in Section 3.

Processor saving decisions are based on system history. Under certain conditions, the scheduler keeps some of the available processors unassigned anticipating future arrivals based on the recent past behavior. The amount of past system behavior remembered by the scheduler is the basis for `p_sav` decisions and determines the `p_sav` level of each policy. Starting from a policy with processor saving level 0 (i.e., a greedy policy that assigns all available processors as soon as possible), a hierarchy of `p_sav` policies with increasing `p_sav` level is constructed. Higher level policies are based upon additional past system history and tend to save more processors.

The level 0 `p_sav` policy is the greedy (or work-conserving) baseline version for the entire hierarchy. The policy is given in Algorithm 2. By setting the variable `system_size` to two, the simple Processor Saving Policy 0, PSP0, is obtained. PSP0 assigns the entire system to a job if it is the only job in queue waiting for service and the entire system is free. If two or more jobs are waiting for service, and the currently finishing job was allocated the entire system, the first two jobs in the FIFO queue are each allocated half of the processors. When a job that executes with half of the processors completes and there are jobs waiting for service, the first job in the queue is scheduled on the released partition.

A second policy, PSP1 (Processor Saving Policy 1), is considered that introduces one `p_sav` level. It is obtained from Algorithm 1 by setting `system_size` to two, thus forcing the partition sizes to either the entire system or one-half of it, as with PSP0. A first level of processor saving is implemented by assigning only one-half of the system to the next incoming job, when the last job to finish had been allocated one-half of the processors and there are no jobs waiting for execution. That is, the system "remembers" that, before emptying out, both partitions were busy in the recent past, since the job that finished last had been allocated only one-half of the system. This is regarded as an indication of the load being sufficiently high to keep allocating only half of the system. Therefore, the system saves half of the processors for anticipated future arrivals. If no further arrivals occur during the execution of this newly arrived job on half of the system, the next incoming job will be assigned the entire system.

An additional `p_sav` level is introduced if the system remembers a longer period of the system's history, e.g., the states when both partitions were busy and at least one job was waiting for execution. The new policy, PSP2 (Processor Saving Policy 2), behaves like PSP1 except when the system has reached a state where there are jobs waiting for execution and both partitions are busy before emptying out for the first time. In this case, two `p_sav` decisions are applied before returning to the baseline behavior. The first `p_sav` decision, i.e., assigning half of the available processors instead of the entire system to the first incoming job, is applied when the system empties out for the first time. If no new arrival occurs during this job's execution, then the second `p_sav` decision, again assigning only half of the system, is applied. The system assigns half of the processors twice since it remembers a "high load" state, i.e., two jobs executing and at least one waiting, the rationale being that it is likely that several jobs may arrive in the near future because such a behavior has been observed in the recent past. If, again, no new arrival occurs during the second job's execution, the next incoming job will be assigned the entire system, thus returning to the PSP0 baseline behavior. PSP1 resets after *one* erroneous decision, i.e., when one-half of the processors were reserved for an anticipated arrival that did not occur, while PSP2 resets only after *two* such consecutive erroneous decisions.

A hierarchy of `p_sav` policies, with respect to the `p_sav` level, can be constructed by systematically increasing the amount of recent past history kept by the system,[2] i.e., the number of consecutive p_sav assignments attempted before returning to the baseline greedy assignment. Thus, policy PSPn is "less" p_sav than policy PSP$n + 1$.

The PSP policies are modeled with continuous time first order Markov chains. The Markov diagrams of the PSP0, PSP1, and PSP2 policies are depicted in Fig. 2. The parameters λ, μ_h, and μ_e represent the job's arrival rate, average execution rate when allocated half or the entire system, respectively. The state notation (q, r_p) indicates that q jobs are waiting for execution and r jobs are executing, each on a partition of size p. For the simple policies considered here, $p = h$ or $p = e$ for half or the entire system, respectively. As an example, state $(3, 1_e)$ indicates that three jobs are waiting for service and one job is executing on the entire system. State 0 indicates a completely idle system. Shaded states indicate p_sav states, that is, states where a processor saving decision has been or will be made. A superscript in a state notation distinguishes the p_sav level to which a state belongs (it is omitted for p_sav level 0 states). For example, the shaded state $0'$ indicates an empty system in a p_sav state of level 1, i.e., a state where a p_sav decision is applied. Introducing states to remember the recent past behavior is equivalent to constructing higher order Markov chains while preserving the simple solution of first order Markov chains.

The higher the policy p_sav level, the longer the policy will be in p_sav states. The concept of p_sav level is quantified by the total steady state probability of being in a p_sav state, which measures how long the return to the basic work-conserving behavior is delayed. The probability of being in a p_sav state is plotted in Fig. 3 as a function of the offered load for the PSP1 and PSP2 policies under three workload types. The analytical results support the intuitive observation that the total probability of being in a p_sav state increases inversely proportionally to the workload parallelism.

2.3 Performance Results

The performance figures derived from solving the global balance equations of the Markovian models [20] of the allocation policies are reported in this section. The goal of this analysis is to investigate the performance trade-offs of the p_sav policies with respect to their greedy, or work-conserving, counterparts. Therefore, the ratio of the average response time under a given p_sav policy (PSP1, PSP2) to the average response time under the baseline work-conserving policy (PSP0) is considered. The impact of different workloads is investigated by changing the speedup concavity and the job arrival rate λ. Baseline results are given for exponential interarrival and service times. Nonstationary arrival processes arel also considered.

In Fig. 4, the response time ratios for the PSP1 (solid line) and PSP2 (dashed line) policies to PSP0 are plotted against the offered load. The horizontal performance line at 1 represents the performance of the baseline policy, PSP0. Performance above the horizontal line at 1 indicates a loss, i.e., higher response time, with respect to PSP0, while performance below 1 (shaded area) indicates a gain, i.e., lower response time, from using a processor saving policy. Ratios equal to 1 for offered load equal to 0 percent and 100 percent are assumed in all cases, since all policies behave similarly at low and high load, i.e., they all allocate the largest or smallest feasible partition, respectively. As expected, on a

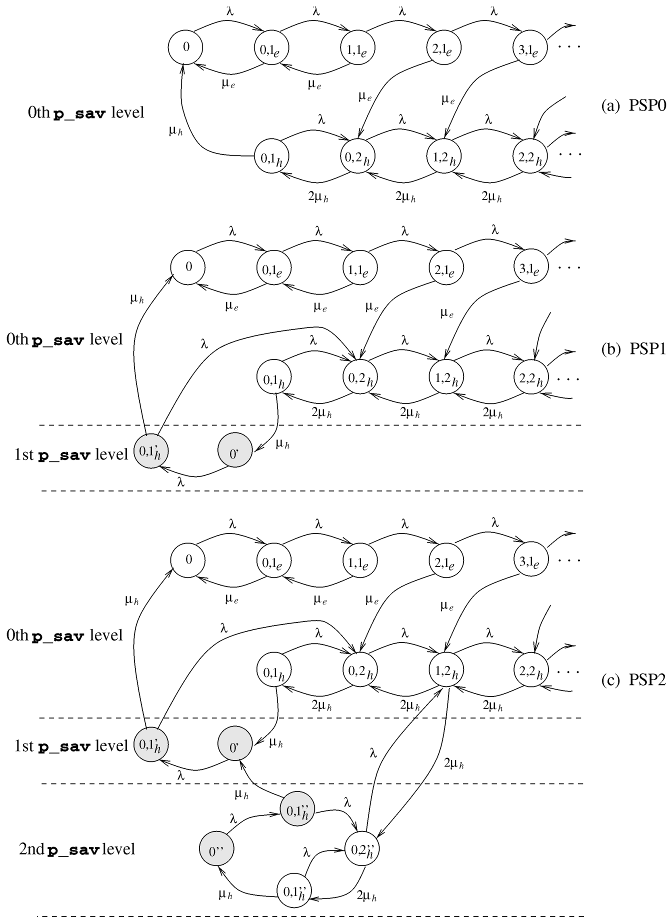


Fig. 2. Markov diagrams of the basic policy PSP0 (a), PSP1 (b), and PSP2 (c) with p_sav level 0, 1, and 2, respectively.

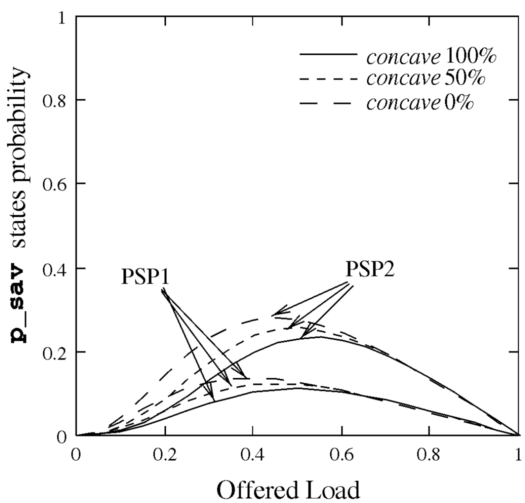


Fig. 3. Probability of processor saving states with policies PSP1 and PSP2 for various workloads.

2. Other policies and relative hierarchies are possible by changing the way transitions between p_sav and regular states occur and/or the way history of previous states is kept by the system.

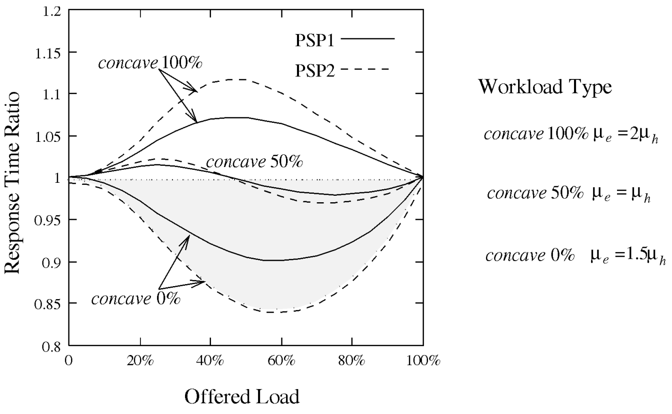


Fig. 4. Response time ratio of PSP1 (solid lines) and PSP2 (dashed lines) to PSP0 (horizontal line) for three workloads. The shaded area indicates the performance gains with respect to PSP0.

poorly parallel workload, both PSP1 and PSP2 outperform PSP0. With such a workload, the maximum performance gain is 10 percent for PSP1 and 16 percent for PSP2. PSP2 outperforms PSP1 across the entire range of offered load because it delays its return to the work-conserving behavior more than PSP1 does. With a poorly parallel workload, a fixed equipartitioning policy that assigns the smallest possible partition to each job optimizes performance. On the contrary, with a highly parallel workload, there is no advantage in saving processors. In this case, PSP0 outperforms both PSP1 and PSP2 at all offered loads. Performance degradation is more serious with PSP2 since it is "more" `p_sav` than PSP1.

For intermediate workloads, particularly under medium to high loads, `p_sav` policies outperform their work-conserving counterpart. The wave-shaped curves of Fig. 4 are typical of various intermediate speedup workloads. For intermediate speedups the "less" `p_sav` policy PSP1 performs better at low load than the "more" `p_sav` policy PSP2. Their relative performance is reversed at high load. The workload speedup characteristics determine the offered load level, i.e., the crossover point, where performance of the `p_sav` policy and of the work-conserving policy are equal. For a given workload, the crossover point [20] is given by $\lambda^*/(P\mu_1)$, where λ^* satisfies the following equation:

$$RT_{\mathrm{PSP0}} = RT_{\mathrm{PSPn}} \tag{3}$$

for n = 1 and 2. RT_{PSP0} is the response time of PSP0 and is given by:

$$RT_{\mathrm{PSP0}} = P_0^{\mathrm{PSP0}} \frac{\lambda + \mu_e}{\lambda^2 + \lambda\mu_e + \mu_e^2} \cdot \left\{ \frac{(\lambda + \mu_e)^2}{\mu_e^2} + \frac{\lambda^2}{2\mu_h - \lambda} \left[\frac{\lambda(2\mu_h - \lambda) + \mu_e(4\mu_h - \lambda)}{\mu_e} + \frac{4\mu_h}{\lambda + \mu_e} \right] \right\}, \tag{4}$$

where

$$P_0^{\mathrm{PSP0}} = \frac{\mu_e(2\mu_h - \lambda)(\lambda^2 + \lambda\mu_e + \mu_e^2)}{\mu_e(2\mu_h - \lambda)(\lambda^2 + \lambda\mu_e + \mu_e^2) + 2\mu_e\lambda^3 + \lambda(\lambda + \mu_e)[\mu_e(2\mu_h - \lambda) + 2\lambda\mu_h]}.$$

RT_{PSP1} is the response time of PSP1 and is given by:

$$RT_{\mathrm{PSP1}} = P_0^{\mathrm{PSP1}} \cdot \frac{(\lambda + \mu_e)^3(2\mu_h - \lambda)^2 + \lambda^2\{[\lambda\mu_e + (2\mu_e + \lambda)(2\mu_h - \lambda)](\lambda + \mu_e) + 4\mu_e^2(\lambda + 2\mu_h)\}}{\mu_e^2(2\mu_h - \lambda)^2(\lambda^2 + \lambda\mu_e + \mu_e^2)}, \tag{5}$$

where

$$P_0^{\mathrm{PSP1}} = \left\{ 1 + \left[\frac{\lambda(\lambda^3 + \lambda^2 + 3\lambda^2\mu_h + 3\lambda\mu_h + \mu_h^2)}{\mu_h^2(\lambda + \mu_e)} + \frac{\lambda^2 + (\lambda + \mu_e)(2\mu_h - \lambda)}{\mu_e(2\mu_h - \lambda)} \right] \frac{\lambda(\lambda + \mu_e)}{\lambda^2 + \lambda\mu_e + \mu_e^2} \right\}^{-1}.$$

Similar expressions can be derived for RT_{PSP2}. Given the forms of (4) and (5), solving (3) for λ^* explicitly is not feasible, only numerical solutions are possible. λ^* is a function of the speedup concavity, i.e., the workload type. As speedup decreases, the intersection point moves leftward, reaching zero in the limit when the workload is poorly parallel, where it is always advantageous to make `p_sav` decisions. As speedup increases, the intersection point moves rightward, reaching 100 percent in the limit when the workload is highly parallel, where it is never advantageous to save processors.

2.4 Arrival Bursts

In this section, the performance trade-offs of the `p_sav` policies in the presence of bursty arrivals are explored. An arrival burst consists of two or more jobs arriving at the system simultaneously. They introduce noise in a stationary Poisson arrival process. Arrival bursts are modeled as bulks [8] of a given size and arrive at the system with a certain probability γ. Bulk arrivals of fixed size are selected as an example of nonstationary arrival process that can be solved analytically. In Section 3.2, other examples of nonstationary arrival processes are considered and analyzed via simulation.

With bulks of size two, two jobs arrive at the system simultaneously with probability γ. Single arrivals occur with probability $1 - \gamma$. In Fig. 5, the Markov diagrams of PSP0 and PSP1 are plotted with bulk arrivals of size two (bold arcs labeled $\lambda\gamma$). The Markov diagram for the PSP2 policy can be constructed in a similar fashion and is not reported for the sake of conciseness. The underlying global balance equations are solved analytically [20] and performance measures are derived for the entire range of offered load.

The response time ratios of PSP1 with respect to PSP0 are plotted in Fig. 6a for various burst probabilities (i.e., $\gamma = 0$, the base case with no bursts, and $\gamma = 0.5$, 0.9) and workload types. As the figure shows, the presence of bursts in the arrival process emphasizes the previously observed behavior. Bulk arrivals tend to minimize the chances for wrong processor saving decisions and tend to maximize the potential utilization of reserved processors for future arrivals. Fig. 6a indicates that, as the workload speedup decreases and as the arrival rate increases with increasing bulk probabilities, the relative performance of PSP1 improves.

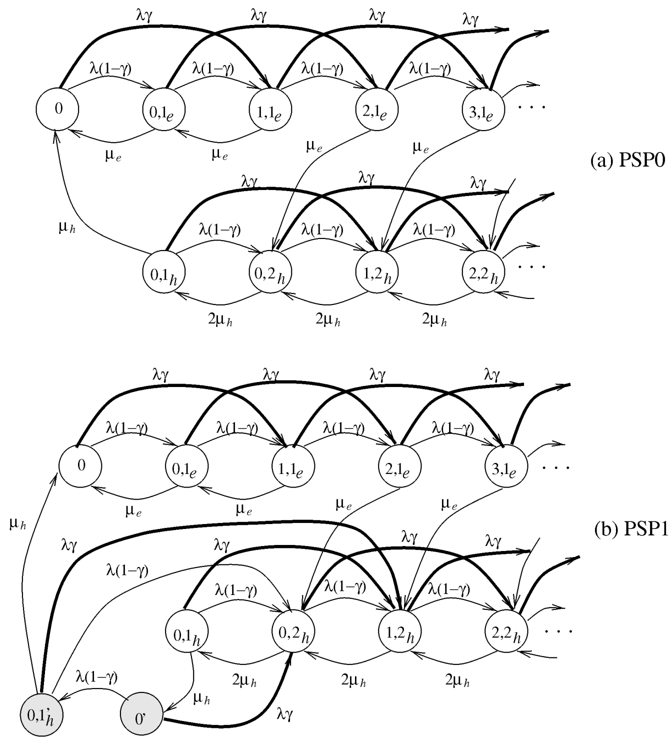


Fig. 5. Markov diagrams of PSP0 and PSP1 with bulk arrivals (bold arcs labeled $\lambda\gamma$) of size two.

The impact on performance of the burst size has been investigated by fixing the probability of a burst and varying the size of the bursts. In Fig. 6b, the response time ratios of PSP1 to PSP0 are plotted for a *concave*50% workload with bulk arrivals of size two and three for bulk probability equal to 0.5 and for the base case with single arrivals only. The results indicate a clear advantage in using processor saving policies as the bulk size increases. The trade-off between gains and losses, corresponding to the intersection point of the PSP1 curve with the reference line at 1, moves leftward as the bulk size increases. The λ^* value corresponding to the crossover point is identified as illustrated in (3). For a bulk of size three, a performance improvement is achieved under PSP1, even at offered load close to 0 percent (the sharp drop in the response time ratio), since a bulk of size three fully utilizes the system and leaves one job waiting in the queue. On the other hand, at high load, the performance improvement with bulks of size three is less than with bulks of size two because of saturation effects.

These results indicate that the more bursty the arrival process, or the larger the expected arrival bursts, the better it is to anticipate future arrivals by saving some processors.

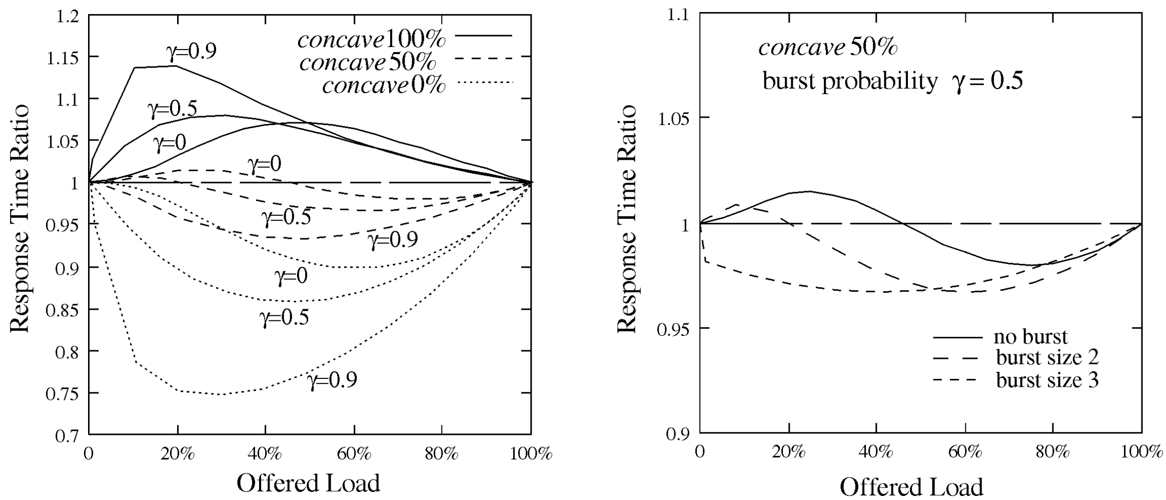
3 Simulation Analysis

The general version of the policies examined in the previous section is investigated by simulation in this section. The whole spectrum of possible assignments is allowed, thus providing for greater system flexibility. Because of the variety of feasible allocations combined with the number of `p_sav` decisions, the `p_sav` policies examined in this section are expected to adapt well to unpredictable workload behavior.

Due to the size and complexity of the underlying Markovian models, when more than two partitions are allowed, the evaluation study is conducted via simulation. All simulation estimates have 95 percent confidence intervals. A wide range of parameters is investigated via simulation, namely,

- offered load (the entire range),
- workload speedup (from *concave*20% to *concave*94%),
- system size (32, 64, 128, and 256 processors),
- coefficient of variation of the job execution time ([0, 10]),
- size ([2, 8]) and probability ([0.1, 0.9]) of arrival bursts, and
- instantaneous arrival rate.

The speedup curves of the workloads considered were derived analytically, using (1), so as to fit the experimental curves measured on the Paragon illustrated in Section 4. One additional curve is considered, namely *concave*94%, in order



(a) (b)

Fig. 6. Response time ratio of PSP1 to PSP0 for bursty arrivals (a) of size two for various burst probabilities (γ = 0, 0.5, 0.9) for three workload types and (b) of various sizes (two and three) with probability γ = 0.5 on a *concave*50% workload.

to consider highly parallel workloads. The speedups analyzed span from *concave*20% to *concave*94%. Response time ratios with respect to the greedy counterpart are plotted as a function of the offered load, except when sensitivity analysis to the parameters listed above is performed. In such cases, the offered load is set to 70 percent and results are plotted as a function of the investigated parameter.

3.1 The PSA Policy

The policy presented in this section is the general version of PSP1 with respect to the number and size of partitions allowed in the system. To emphasize the variability of the possible partition sizes, it is denoted as the Processor Saving Adaptive (PSA) policy. Partitions of all sizes, ranging from one to the entire system, are possible. No additional overhead is introduced for the allocation of partitions of various sizes. PSA treats all jobs as statistically identical as workload characteristics are assumed to be unknown to the scheduler. It implements one `p_sav` level by saving processors once before returning to the baseline work-conserving behavior after making one mistake. The policy target is an adaptive equipartitioning scheme that adjusts the partition size according to the workload intensity indicated by the queue length seen at each scheduling instant. The recent past system behavior is the basis for `p_sav` decisions. When the system becomes idle after a fragmentation period, i.e., a period when the last allocated partition is smaller than the entire processor set, the system remembers that it comes from such a period, i.e., that a "high load" case recently occurred. In this case, the next incoming job is assigned only half of the available processors: the system "remembers" that it has been busy in the recent past executing more than one job simultaneously and, based on this history, it saves some processors for anticipated future arrivals. If no anticipated future arrival occurs during the newly arrived job's execution, the next incoming job will be assigned the entire system. As in other nonpreemptive adaptive policies, when the number of available processors is smaller than the current partition size, no job is scheduled and the processors are left idle until the next system state change, i.e., job departure or completion. The pseudocode for the PSA policy is reported in Algorithm 1.

The pseudocode for the work-conserving version of PSA is reported in Algorithm 2. As mentioned in Section 2.2, PSP0 is obtained from this algorithm by setting the variable `system_size` to two. Like PSA, its work-conserving version allows for partitions of any size, between one and P, the system size.

The general version of PSP2 is derived from PSA by adding extra state variables to remember longer periods of system history, similar to the way PSP2 was obtained from PSP1. For the sake of simplicity, only the results of the investigation of the PSA policy are presented here.

3.2 Performance Results

As in Section 2, the ratio of the average response time of the PSA policy to its work-conserving counterpart is considered. The interarrival and execution times are assumed to be exponentially distributed. Sensitivity analysis with respect to the system size and the distributions of the arrival and service processes is presented.

```
PSA policy
if executed_alone(last_executed_job) then
    last_released_partition := system_size
job_in_queue := length(waiting_jobs_queue)
if (job_in_queue > 0) then {
    if (job_in_queue = 1) and
    ((executing_jobs = 0 and last_released_partition ≠ system_size) or
    (executing_jobs = 1 and partition_size = system_size)) then {
        if (partition_nb > 2) then
            partition_nb := partition_nb – 1
        else partition_nb := 2
    }
    else {
        if (job_in_queue ≥ partition_nb) then
            partition_nb := min(system_size, job_in_queue)
        else if ((partition_nb – executing_jobs) > 1) then
            partition_nb := partition_nb – 1
    }
    partition_size := [system_size/partition_nb + 0.5]
    while (free_processors > 0 and job_in_queue > 0) do {
        partition := find(system, partition_size)
        schedule(job, partition, partition_size)
    }
}
```

Algorithm 1. Pseudocode for the Processor Saving Adaptive policy.

```
work-conserving policy
job_in_queue := length(waiting_jobs_queue)
if (job_in_queue > 0) then {
    partition_nb := min(system_size, job_in_queue)
    partition_size := [system_size/partition_nb + 0.5]
    while ((free_proc > 0) and (job_in_queue > 0)) do {
        if (partition_size > free_proc) then
            partition_size := free_proc
        partition := find(system, partition_size)
        schedule(job, partition, partition_size)
    }
}
```

Algorithm 2. Pseudocode for the work-conserving version of the Processor Saving Adaptive policy.

The response time ratios for the PSA policy under exponential interarrivals and execution times as a function of the offered load are plotted in Fig. 7. Results are reported for all workload types considered, spanning from *concave*20% to *concave*94%. As in the analytical case (see Fig. 4), the curves follow a similar wave-shaped trend. With highly parallel workloads, performance losses result up to medium load. As the offered load increases, the policies' performance becomes equivalent until the relative performance switches and performance gains are observed at medium to high load. At low load, it is better not to reserve processors for anticipated future arrivals since they are unlikely to occur. The executing jobs can take advantage of all the available processors. As the workload concavity diminishes, `p_sav` performance gains become more apparent. With poorly parallel workloads, the `p_sav` policy outperforms its work-conserving counterpart across the entire range of offered load. A maximum gain of about 30 percent is achieved in the range of 20 percent to 40 percent of the offered load. In general, both losses and gains are more consistent than in the analytical case. The larger system size considered here accounts for the observed differences. As previous studies show, e.g., [22], [11], [12], due to software inefficiencies, performance improves by executing several jobs simultaneously on smaller partitions. The reduced waiting time in the

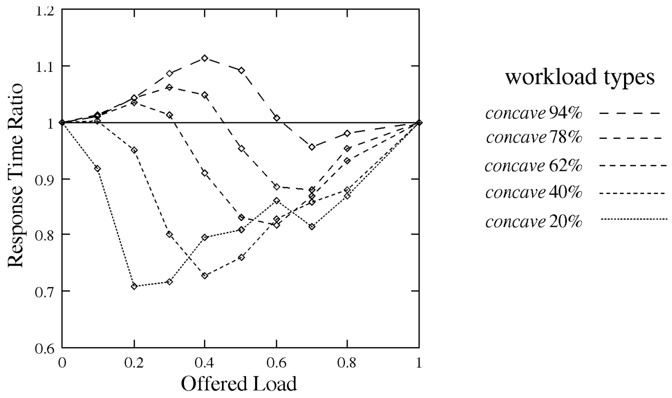


Fig. 7. Response time ratio of the PSA policy to the corresponding work-conserving policy under exponential interarrival and execution times for various workloads in a system with 64 processors.

queue prior to scheduling seems to compensate for a possibly increased execution time.

3.2.1 Sensitivity w.r.t. System Size

In Fig. 8a, the response time ratios of the PSA policy are reported as a function of the system size ($\log_2$ scale) at offered load 70 percent. Systems of 32, 64, 128, and 256 processors are considered. Exponential interarrival and service times are assumed. A common trend is observed: As the system size grows, the advantage from using `p_sav` policies increases, regardless of the workload type. The relative ranking of the curves with respect to the workload type is preserved. Larger gains are derived from less scalable workloads. However, for large systems, performance gains also result for highly parallel workloads. With larger systems, unutilized processors reserved in anticipation of future arrivals that results in wrong `p_sav` decisions, do not impact on performance as in smaller systems.

3.2.2 Sensitivity w.r.t. Execution Time Distributions

The analysis of Section 2 suggests that processor saving policies tend to work well under irregular workload behavior. To

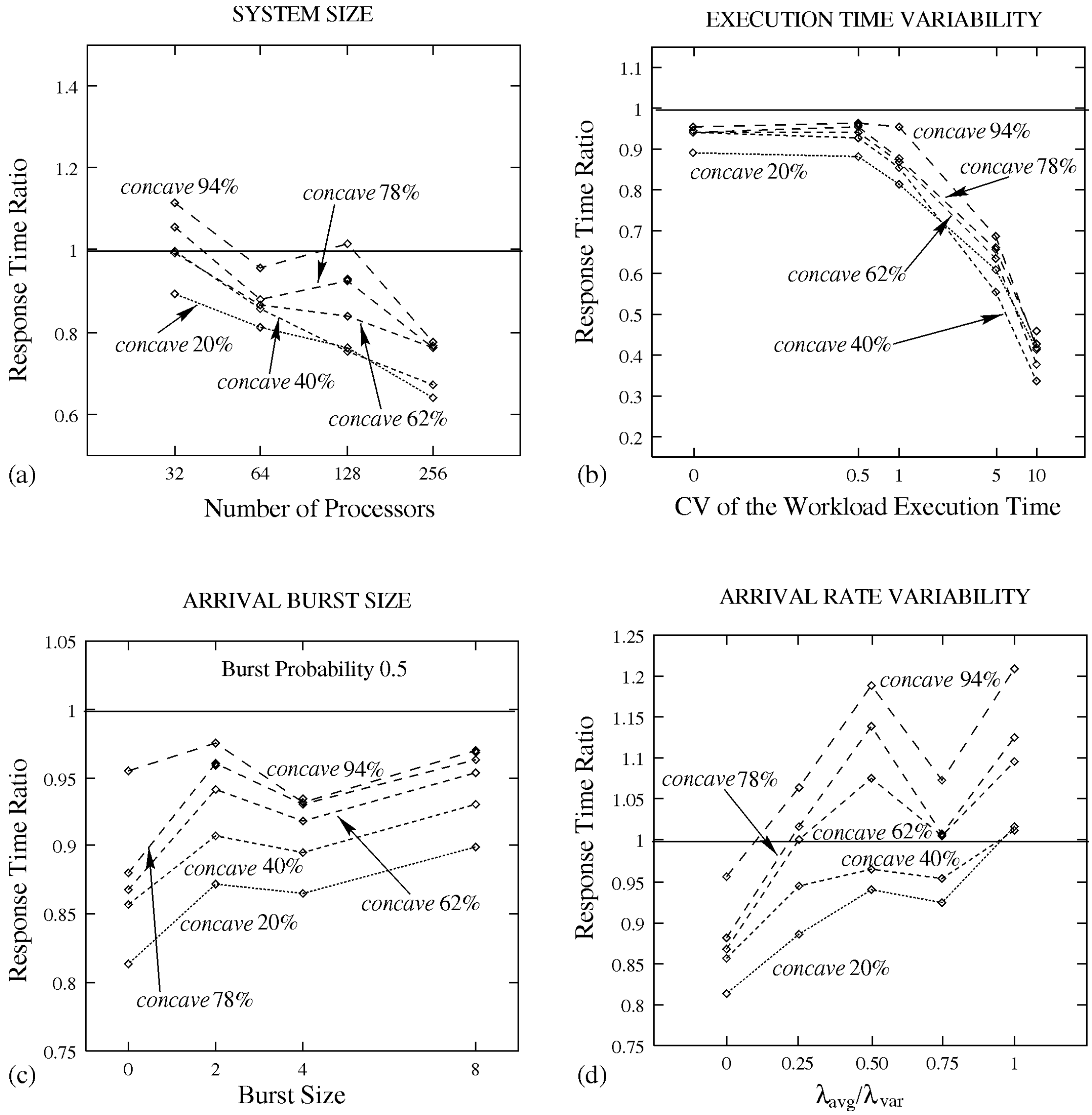


Fig. 8. Response time ratio for the PSA policy at 70 percent offered load, with respect to various sensitivity parameters: (a) system size (log scale), (b) execution time coefficient of variation (log scale), (c) burst size of the arrival process, and (d) instantaneously varying arrival rate.

validate such a hypothesis, the coefficient of variation (CV) of the execution time distributions is varied over the range [0, 10] under a Poisson arrival stream for a system of 64 processors. The entire range of offered load is considered. Results are reported in Fig. 8b as a function of the CV of the job execution time distribution ($\log_2$ scale) at 70 percent offered load. The PSA policy performance improves as the CV increases. Thus, as the probability of a long service time increases, it is better to save processors to guard against such an anomalous occurrence. As the figure shows, performance is insensitive to workload speedup. Another situation where `p_sav` policies may prove valuable is when jobs have long execution times, regardless of the distribution. In this case, saving processors for future arrivals can decrease the job waiting time for a partition on which to execute. For a given speedup and a given offered load, jobs with long execution time benefit more than those with short execution time, as the latter have a faster turnaround.

3.2.3 Sensitivity w.r.t. Nonstationary Arrival Processes

Two types of nonstationary arrival processes are considered, under the assumption of exponential execution times for a system with 64 processors. Bursty arrivals and instantaneous arrival rate that varies sinusoidally are investigated. With bursty arrivals, as in the analytical models of Section 2, at each arrival instant, a burst arrival of a given size occurs with probability γ and a single arrival occurs with probability $1 - \gamma$. Bursts of size two, four, and eight, each with probability 0.1, 0.5, and 0.9, are considered. Results are reported for all burst sizes at probability 0.5. The times between two consecutive arrivals (either single or bursty) are exponentially distributed.

In Fig. 8c, the response time ratios for the PSA policy are plotted as a function of the burst size for probability 0.5 for the various workload types at 70 percent offered load. Burst size zero represents the base case with no bursts where only single arrivals are allowed. Trade-offs between performance, burst size, and burst probability are observed. Consistent with the Markovian analysis of Section 2.4, the PSA policy outperforms the greedy counterpart in all cases.

The arrival process with sinusoidally varying instantaneous arrival rate is obtained by considering exponential interarrival times with instantaneous arrival rate $\lambda(t)$ given by

$$\lambda(t) = \lambda_{avg} + \lambda_{var}\sin(t/k) \qquad 0 \le \lambda_{var} \le \lambda_{avg}, \tag{6}$$

where λ_{avg} is the constant arrival rate of the basic Poisson process, λ_{var} is the fraction of the base arrival rate carrying the sinusoidal noise into the arrival process, and k is the scaling factor for the argument of the sin function. k is defined as $\frac{duration\ of\ run}{10\pi}$, so that five cycles are simulated in a run. When the instantaneous arrival rate varies, as in (6), periodic cycles of light load and heavy load result from the sinusoidal variation of the arrival rate. Exponential interarrival times with instantaneous arrival rate given by (6) are generated. As in the previous case, workload execution time is assumed exponential. The results of the simulations for $\lambda_{var}/\lambda_{avg}$ in [0, 1], i.e., the two extremes of pure Poisson for $\lambda_{var}/\lambda_{avg} = 0$ and pure sinusoidal for $\lambda_{var}/\lambda_{avg} = 1$, are plotted in Fig. 8d for various workload types for a system with 64 processors at 70 percent offered load. Under the nonstationary arrival process described above, the relative performance of the PSA policy is more sensitive to the workload parallel characteristics than under bursty arrivals.

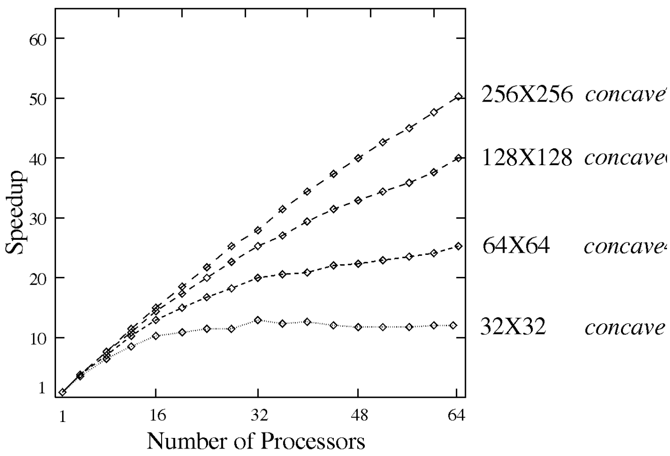


Fig. 9. Experimental speedup curves for LU decomposition obtained on the Intel Paragon.

4 Experimental Results

In this section, experimentation on a multiprocessor system using an actual workload is used to investigate policy performance and validate previous results. The results presented in Section 3 were derived for single class workloads, on a 64 processor system. In this section, these results are validated on actual workloads and system sizes, and extended by considering workload mixes comprising various classes of components. Experimentation is used to investigate the impact of single and multiclass workloads on the performance under `p_sav` policies.

In our experimental setting, a 512 node Intel Paragon, a message passing multiprocessor system with distributed memory [7], is the experimental platform. Experiments with 32, 64, and 128 processors were conducted and results are presented for 64 processors. The omitted cases exhibit similar behavior to the results described here. Actual applications are submitted to the scheduler according to a Poisson process. The application used as test workload is an LU decomposition kernel executed on matrices of different size, obtaining a range of speedup curves. Four matrix sizes, namely, 32×32, 64×64, 128×128, and 256×256, are considered. The speedup curves for the various matrix sizes on the Paragon are reported in Fig. 9.

The scheduler implements the PSA policy and serves the applications in FIFO order. It computes the partition size to be allocated and then starts the application execution on the assigned partition.

4.1 Single Class Workload

Results for experiments with 64 processors are reported for the PSA policy with single class workloads. Each single class workload is obtained by using a different data set size for the LU decomposition algorithm (see Fig. 9 for the corresponding speedup curves). Since the measured execution

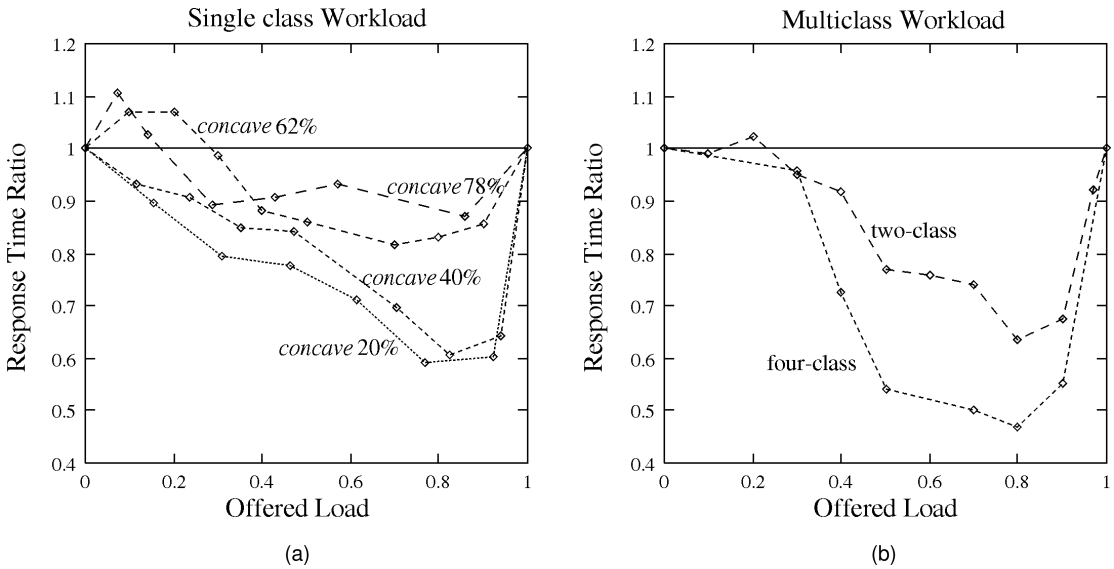


Fig. 10. Response time ratios of the PSA policy for experiments on the Intel Paragon with 64 processors with (a) single class and (b) multiclass workloads.

time for a given number of processors has negligible variance, the experiments are characterized by Poisson arrivals and a quasi-deterministic execution time distribution. Performance is studied across the range of possible offered loads by varying the job arrival rate. For each arrival rate considered, 4,000 jobs are submitted to the system and the average response time is measured.

Fig. 10a illustrates the response time ratios as a function of offered load for the four single class workload types given in Fig. 9 under the PSA policy. Each experiment was repeated multiple times. The average response times were measured and reported. As Fig. 10a shows, the response times for the PSA policy are generally better than those of their work-conserving counterpart. The exceptions occur when the load is light and the workload has high concavity. Relative performance rankings from poorly parallel to highly parallel workloads are preserved. The maximum PSA advantage is achieved with workloads that scale poorly (32×32 case). As the system size increases, the performance gains achieved with the PSA policy over its work-conserving counterpart become more significant. The intersection point of each curve with the horizontal work-conserving reference line moves leftward as the system size increases and the workload speedup decreases. With larger systems where processor scheduling policies allow for a wider variety of partition sizes and higher multiprogramming levels, the disadvantages of processor saving policies (i.e., potential waste of idle processors) is minimized.

4.2 Multiclass Workload

Multiclass workloads exhibit widely varying execution requirements, scalability, and communication characteristics. They are generally considered as a more representative model of the real workload executed on actual systems than single class workloads. In the presence of unpredictable workload behavior, such as that exhibited by multiclass workloads, `p_sav` policies are expected to perform better than work-conserving ones.

A Poisson arrival process for the multiclass workload is obtained by superimposing single class workloads. Let the quadruple (W percent, X percent, Y percent, Z percent) represent the percentages of each single class component in a multiclass mix. The single class components are given by the LU decomposition executed on different matrix sizes (see Fig. 9). Thus, W percent of the arriving jobs belong to class 1, i.e., perform LU decomposition on a matrix of size 32×32, X percent of the arriving jobs belong to class 2, i.e., a matrix size of 64×64, Y percent belong to class 3, and Z percent belong to class 4, i.e., matrices of size 128×128 and 256×256, respectively. In Fig. 10b, response time ratios for the PSA policy on 64 processors with two job mixes, namely, a four-class (25 percent, 25 percent, 25 percent, 25 percent) and a two-class (0 percent, 50 percent, 50 percent, 0 percent) workloads, are reported. The four-class mix represents a more heterogeneous workload than the two-class mix and, as expected, benefits more from a processor saving policy. As the workload becomes more homogeneous, i.e., approximating a single class workload, the benefits of `p_sav` policies decrease. These experiments provide evidence that, if the workload components vary considerably, performance may improve by saving some processors to act as a buffer against such variability.

5 Conclusions

In this paper, it has been shown that, in multiprogrammed multiprocessor systems, processor saving scheduling policies, i.e., policies that may keep some of the available processors idle in the presence of work to be done, may yield better performance than their corresponding work-conserving counterparts. Conditions under which this occurs have been investigated by varying the offered load, workload type, system size, coefficient of variation of the execution time distribution, and the arrival process.

In general, if the workload varies considerably, performance improvements may result from saving some processors as a buffer of computational power against such a variability. The main conclusions are:

- The more heterogeneous the workload is (i.e., multiclass), the better the performance of the processor saving policies.
- Workloads with irregular execution time distributions benefit from processor saving policies.
- The largest advantage of `p_sav` policies is observed at offered load in the range [30 percent, 80 percent].
- Processor saving policies are effective under nonstationary arrival processes, especially when bursty arrivals are considered.

The more unstable the conditions are with respect to the parameters examined, the greater the relative gains derived from using `p_sav` policies.

Acknowledgments

We gratefully acknowledge the support of Oak Ridge National Laboratories for providing access to their Intel Paragon systems for the experimental analysis. This work was partially supported by Italian M.U.R.S.T. 40% and 60% projects, and by subcontract 19X-SL131V from the Oak Ridge National Laboratory managed by Martin Marietta Energy Systems, Inc. for the U.S. Department of Energy under contract no. DE-AC05-84OR21400.

References

[1] S.-H. Chiang, R.K. Mansharamani, and M.K. Vernon, "Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies," *ACM SIGMETRICS*, pp. 33-44, 1994.

[2] D.L. Eager, J. Zahorjan, and E.D. Lazowska, "Speedup versus Efficiency in Parallel Systems," *IEEE Trans. Computers*, vol. 38, no. 3, pp. 408-423, Mar. 1989.

[3] K. Dussa, B. Carlson, L.W. Dowdy, and K.-H. Park, "Dynamic Partitioning in a Transputer Environment," *ACM SIGMETRICS*, pp. 203-213, 1990.

[4] D.G. Feitelson and L. Rudolph, "Distributed Hierarchical Control for Parallel Processing," *Computer*, vol. 23, no. 5, pp. 65-77, May 1990.

[5] D. Ghosal, G. Serazzi, and S.K. Tripathi, "Processor Working Set and Its Use in Scheduling Multiprocessor Systems," *IEEE Trans. Software Eng.*, vol. 17, no. 5, pp. 443-453, May 1991.

[6] A. Gupta, A. Tucker, and S. Urushibara, "The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications," *ACM SIGMETRICS*, pp. 120-132, 1991.

[7] Intel Corporation, *Paragon OSF/1 User's Guide*, 1993.

[8] L. Kleinrock, *Queueing Systems*, vol. 1. Wiley Interscience, 1975.

[9] S.T. Leutenegger and M.K. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Policies," *ACM SIGMETRICS*, pp. 226-236, 1990.

[10] S. Majumdar, "The Performance of Local and Global Scheduling Strategies in Multiprogrammed Parallel Systems," *Proc. 11th Ann. Conf. Computers and Comm.*, pp. 1.3.4.1-1.3.4.8, 1992.

[11] S. Majumdar, D.L. Eager, and R.B. Bunt, "Scheduling in Multiprogrammed Parallel Systems," *ACM SIGMETRICS*, pp. 104-113, 1988.

[12] S. Majumdar, D.L. Eager, and R.B. Bunt, "Characterization of Programs for Scheduling in Multiprogrammed Parallel Systems," *Performance Evaluation*, vol. 13, no. 2, pp. 109-130, 1991.

[13] C. McCann, R. Vaswani, and J. Zahorjan, "A Dynamic Processor Allocation Policy for Multiprogrammed Shared Memory Multiprocessors," *ACM Trans. Computer Systems*, vol. 11, no. 2, pp. 146-178, 1993.

[14] C. McCann and J. Zahorjan, "Processor Allocation Policies for Message-Passing Parallel Computers," *ACM SIGMETRICS*, pp. 19-32, 1994.

[15] C. McCann and J. Zahorjan, "Scheduling Memory Constrained Jobs on Distributed Memory Parallel Computers," *ACM SIGMETRICS*, pp. 208-219, 1995.

[16] V.K. Naik, S.K. Setia, and M.S. Squillante, "Performance Analysis of Job Scheduling Policies in Parallel Supercomputing Environments," *Proc. Supercomputing '93*, pp. 824-833, 1993.

[17] J. Ousterhout, "Scheduling Techniques for Concurrent Systems," *Proc. Third Int'l Conf. Distributed Computing Systems*, pp. 22-30, 1982.

[18] K.-H. Park and L.W. Dowdy, "Dynamic Partitioning of Multiprocessor Systems," *Int'l J. Parallel Programming,* vol. 18, no. 2, pp. 91-120, 1989.

[19] E. Rosti, E. Smirni, L.W. Dowdy, G. Serazzi, and B. Carlson, "Robust Partitioning Policies for Multiprocessor Systems," *Performance Evaluation*, vol. 19, nos. 2-3, pp. 141-165, 1994.

[20] E. Smirni, "Processor Allocation and Thread Placement Policies in Parallel Multiprocessor Systems," PhD dissertation, Vanderbilt Univ., May 1995.

[21] S.K. Setia, M.S. Squillante, and S.K. Tripathi, "Processor Scheduling in Multiprogrammed, Distributed Memory Parallel Computers," *ACM SIGMETRICS*, pp. 158-170, 1993.

[22] K.C. Sevcik, "Characterization of Parallelism in Applications and Their Use in Scheduling," *ACM SIGMETRICS*, pp. 171-180, 1989.

[23] K.C. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Multiprocessors," *Performance Evaluation*, vol. 19, nos. 2-3, pp. 107-140, 1994.

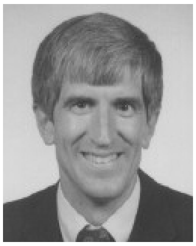
[24] E. Smirni, E. Rosti, G. Serazzi, L.W. Dowdy, and K.C. Sevcik, "Performance Gains from Leaving Idle Processors in Multiprocessor Systems," *Proc. Int'l Conf. Parallel Processing*, pp. III.203-III.210, 1995.

[25] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors," *Proc. 12th ACM Symp. Operating Systems Principles*, pp. 159-166, 1989.

[26] J. Zahorjan and C. McCann, "Processor Scheduling in Shared Memory Multiprocessors," *ACM SIGMETRICS*, pp. 214-225, 1990.

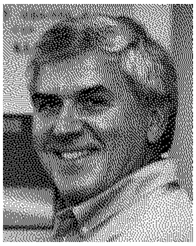
Emilia Rosti received a "Laurea" degree and a PhD degree, both in computer science, from the University of Milan, Italy, in 1987 and 1993, respectively. She is an assistant professor in the Department of Computer Science at the University of Milan, Italy. Her research interests include distributed and parallel systems performance evaluation, workload characterization, processor scheduling policies for multiprocessor systems, models for computer performance prediction, and performance aspects of computer and network security.

Evgenia Smirni received the Diploma degree in computer engineering and informatics from the University of Patras, Greece, in 1987, the MS in computer science from Vanderbilt University, Nashville, Tennessee, in 1993, and the PhD in computer science from Vanderbilt University in 1995. From August 1995 to June 1997, she held a postdoctoral research associate position at the University of Illinois at Urbana-Champaign. She is currently an assistant professor in the Department of Computer Science at the College of William and Mary, Williamsburg, Virginia. Her research interests include parallel input/output, parallel workload characterization, models for computer performance prediction, processor scheduling policies, and distributed and parallel systems.



Lawrence W. Dowdy received a BS in mathematics from Florida State University in 1974 and a PhD in computer science from Duke University in 1977. He is a professor and chair of the Computer Science Department at Vanderbilt University, Nashville, Tennessee. He spent three years at the University of Maryland before joining the faculty at Vanderbilt. During 1987-1988, he spent a sabbatical year in West Germany at the University of Erlangen-Nürnberg. Professor Dowdy's current research interests include models for computer performance prediction, parallel workload characterization, multiprocessor modeling, and calibration.

Giuseppe Serazzi received the "Laurea" degree in mathematics from the University of Pavia, Italy, in 1969. From 1978 to 1987, he was an associate professor in the Department of Mathematics at the University of Pavia. From 1988 to 1991, he was a professor at the University of Milano, Italy. In 1992, he joined the Electrical Engineering and Computer Science Department at the Politecnico di Milano, Italy, where he is currently a professor of computer science. His research interests include workload characterization, modeling, and other topics related to computer systems and network performance evaluation.



Kenneth C. Sevcik holds degrees from Stanford University (BS, mathematics, 1966) and the University of Chicago (PhD, information science, 1971). He is a professor of computer science with a cross-appointment in electrical and computer engineering at the University of Toronto, Canada. He is past chairman of the Department of Computer Science and past director of the Computer Systems Research Institute. His primary area of research interest is in developing techniques and tools for performance evaluation and applying them in such contexts as distributed systems, database systems, local area networks, and parallel computer architectures.

Dr. Sevcik served for six years as a member of the Canadian Natural Sciences and Engineering Research Council, the primary founding body for research in science and engineering in Canada. He is coauthor of the book *Quantitative System Performance: Computer Systems Analysis Using Queueing Network Models* and co-developer of *MAP*, a software package for the analysis of queuing network models of computer systems and computer networks.