JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 21, 246–254 (1994)

Optimal Strategies for Spinning and Blocking*

L. BOGUSLAVSKY,[†,‡] K. HARZALLAH,[†] A. KREINEN,[‡] K. SEVCIK,[†,§] AND A. VAINSHTEIN[‡]

[†]*Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada M5S 1A4; and* [‡]*LVS Corporation, Moscow, Russia*

In parallel and distributed computing environments, threads (or processes) share access to variables and data structures. To assure consistency during updates, locks are used. When a thread attempts to acquire a lock but finds it busy, it must choose between spinning, which means repeatedly attempting to acquire the lock in the hope that it will become free, and blocking, which means suspending its execution and relinquishing its processor to some other thread. The choice between spinning and blocking involves balancing the processor time lost to spinning against the processor time required to save the context of a process when it blocks (context switch overhead). In this paper, we investigate a model that permits us to evaluate how long a process should spin before blocking. We determine conditions under which the extreme cases of immediate blocking (no spinning) and pure spinning (spin until the lock is acquired) are optimal. In other cases, we seek ways of estimating an optimal limit on spinning time before blocking. Results are obtained by a combination of analysis and simulation. © 1994 Academic Press, Inc.

1. INTRODUCTION

The performance of parallel applications on multiprocessors is highly dependent on efficient synchronization. It is not possible to know the states of all the processors, nor is it possible to have a complete knowledge about how long locks will be held. It is possible, however, to check on a lock very frequently. A useful strategy is thus to spin for a while, to see whether or not the lock becomes free, but finally block when it appears that further spinning is not worthwhile. The spin-then-block strategy was first proposed by Ousterhout [13]. This waiting mechanism was implemented in the Medusa operating system [12] with a user-settable limit on the amount of time a process spins before blocking. At one extreme, the process could spin forever before blocking, which is the strategy of *pure spinning*. At the other extreme, the process could block as soon as it finds that the desired resource is not free. This is the *immediate blocking* case.

A disadvantage of immediate blocking is that every time a process finds a lock busy, it switches context. Switching context may lead to a low cache hit rate as the instructions and data of the newly scheduled process may no longer be in the cache. On the other hand, frequent spinning by many processors at once can flood the network with memory requests, hence affecting other processors by reducing their capability for doing useful work.

In this note, we only mention some related papers. A more complete survey of prior relevant work is provided elsewhere [4]. Dimpsey and Iyer found that kernel lock spinning can account for a significant portion of total system overhead, and that system performance can be improved by reducing it [5]. Anderson *et al.* have investigated the full cost of spin-waiting [1] and sought ways of reducing its impact [2]. Mogul and Borg have suggested a way of including the effect of loss of cache affinity in estimating the cost of context switching [11]. Lo and Gligor used limited spinning followed by blocking to improve performance under coscheduling [10]. Zahorjan *et al.* investigated the effects of multiprogramming level and variability in lock holding times on the relative performance of spinning and blocking [14, 15]. Gupta *et al.* noted that poor performance for a set of benchmark applications could be attributed to excessive busy waiting (spinning) for signals from or resources held by preempted processes. They found that overall processor utilization was increased by more than 1/3 when the spin time of processes before blocking was limited to a quantum of length approximately equal to the time taken for a context switch [9]. Karlin *et al.* have studied limits on competitive ratios of spinning strategies [6, 7]. They found that immediate blocking performed very poorly due to excessive time spent in context switching. Empirical work by Lim also supports the benefits of the limited spin before blocking approach [8]. In most of the previous works, trade-offs between spinning and blocking have been examined only in the context of specific architectures and programming models. We are proposing a general model that captures the effects of both spinning and blocking, and spans a broad spectrum of system configurations. We consider a system model where both approaches, spinning and blocking, are incorporated. The goal of our work is to determine how parameters of the system affect the optimal spin time, which maximizes the

* Supported by the Natural Sciences and Engineering Research Council of Canada.

§ To whom correspondence should be addressed. E-mail: kcs@csri.toronto.edu.

system throughput. In our investigation we consider three strategies: pure spinning, immediate blocking, and limited spinning prior to blocking. We will show that each of the three is optimal under some circumstances, depending on system parameters.

Our study is conducted using analytical models (validated via simulations) and simulations alone for the more complex cases. In Section 2 we introduce the system model and the performance measures. In Section 3 we describe analytical models and present exact solutions for particular cases. Approximate solutions for more general cases under Markovian assumptions are described in Section 4. In Section 5 we present simulation results of the models under less restrictive assumptions. Finally, we summarize our observations in Section 6.

2. SYSTEM MODEL

Our model involves a set of J statistically identical threads. Each thread cycles indefinitely, each cycle being composed of two phases: a computational phase followed by a critical section phase. The critical section controls access to some shared structure such as a lock, an I/O list, a ready-list, memory allocation tables, or page allocation tables. The hardware is represented as a set of N identical processors. These threads (customers) and processors can be represented as a closed queueing system as shown in Fig. 1.

Each thread computes on a processor for a time that is a sample from an exponential distribution with mean T_p and then attempts to capture the lock. If the lock is free then the thread acquires it immediately. If the lock is busy, then the thread spins, waiting for the lock to become free. If multiple threads are spinning when the lock is released, one of these threads will be the next one to succeed in acquiring the lock. Once the lock is acquired, it is held for an average time T_l. After releasing the lock, the thread completes, frees the processor, and returns to the waiting queue to begin another cycle. There is a limit on the time a thread can spin. If the spin duration reaches a time selected from a distribution with mean T_s units and the lock has not been acquired, then the thread stops spinning and, after context switching which takes time T_c on average, it releases the processor and enters the blocked state. The thread goes to sleep until it is signaled that the lock is free using a *positive wake-up* protocol [3]. The dark rectangle in Fig. 1 encloses phases in which a thread holds a processor.

This model is a simplified special case of a more general model [4].

There are several possible allocation policies. We have chosen the following rules to determine which thread should seize the processor:

1. If the lock becomes free and some threads are spinning, then one of them captures the lock. If there are no spinning threads, but the queue of blocked threads is not empty, and there is a free processor, then one of the blocked threads immediately captures the lock (as in the positive wake-up protocol).
2. When a processor becomes idle (points A and B in Fig. 1), but the lock is still busy, then only waiting threads can acquire the free processor.

Each thread can be in one of the following states:

- W, waiting for a processor;
- P, computing on a processor;
- S, spinning (that is, repeatedly checking the lock);
- C, context switching (that is, preparing to block);
- B, blocked (that is, waiting for a signal that the lock is available);
- L, holding the lock.

The diagram of the possible transitions of thread states is presented in Fig. 2a.

Each processor can be in one of the following states:

- I, idle;
- P, computing for a thread;
- S, busy with a spinning thread;
- C, busy context switching;
- L, supporting a thread that holds the lock. (When a thread holds the lock it still occupies a processor.)

The diagram of the possible transitions in processor states is represented in Fig. 2b. In our investigations we have used the Resource or Lock Utilization U as the primary performance measure. Resource Utilization is equal to the fraction

$$U_T = \sum_{i=1}^{N} \frac{t_i(T)}{T},$$

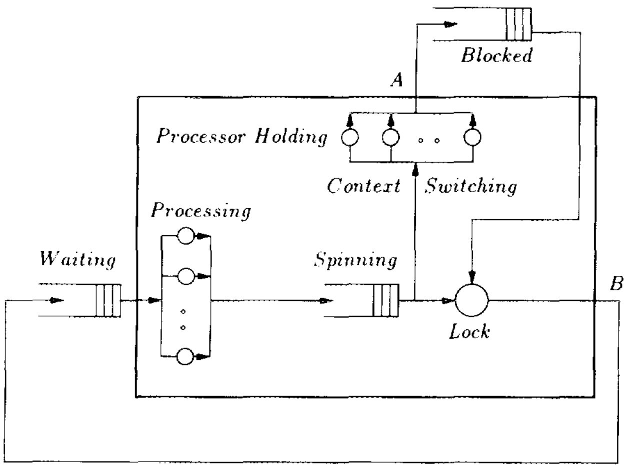

FIG. 1. System model.

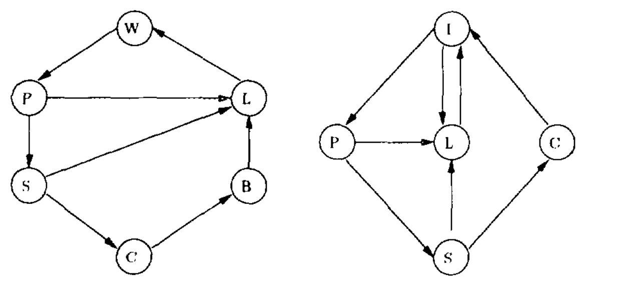

FIG. 2. State transition diagrams. (a) Threads and (b) processors.

where $t_i(T)$ is the time that the ith processor spends in the lock holding state L during a period of length T. We denote by

$$U = \lim_{T\to\infty} U_T$$

the average number of processors holding the resource in a stationary regime. The system throughput, X, is directly proportional to the resource utilization with the relationship: $X = U/T_l$. Our goal is to find the value of the parameter T_s that maximizes the value U. The optimal value of average spin time can be, for various combinations of parameters values, any of:

1. $T_s = 0$, immediate blocking;
2. $0 < T_s < \infty$, spin up to a limit then block;
3. $T_s = \infty$, pure spinning.

3. ANALYSIS

We assume that computing time, spinning time, lock holding time, and context switching time are all exponentially distributed random variables. We are interested in the steady state behavior of the system. As described, this system is modeled as a continuous time, discrete Markov process. In general, the state of the system with J threads and N processors can be described by the five parameters

$$(n_P, n_S, n_C, n_B, n_L),$$

where n_i is the number of threads in state i. From this, the number of processors in each possible processor state, as well as the state of the lock (busy or idle) can be deduced.

3.1. Spinning and Blocking for the Case $N = 2$ and $J = 3$

To simplify notation in the particular case of $N = 2$ and $J = 3$, we denote the states of the Markov chain by the triples of letters from the set $\{W, P, S, L, C, B\}$, where letters denote the state of a thread: W, waiting; P, processing; S, spinning; L, holding the lock; C, context switching; and B, blocked. The graph representing the state space for the case $N = 2$, $J = 3$ is given in Fig. 3 with $\mu_x = 1/T_x$ for $x \in \{p, s, l, c\}$.

The stationary probabilities of the nine states can be calculated straightforwardly from the balance equations derived from the transition diagram. Performance measures can then be expressed in terms of these probabilities. Letting π_S denote the stationary probability of state S, the average number of processors serving threads in their processing phase is given by

$$P = 2\pi_{WPP} + \pi_{WPL} + \pi_{WCP} + \pi_{BPL}.$$

The resource utilization is

$$U = 1 - \pi_{WPP} - \pi_{WCP}.$$

PROPOSITION 1. *For the case where $N = 2$, and $J = 3$, there exist values of parameters μ_p, μ_l, μ_c such that maximum resource utilization can be attained with μ_s^* where $0 < \mu_s^* < \infty$.*

Proposition 1 states that, at least for some combinations of processing time, lock holding time, and context switching time, the optimal strategy is neither pure spinning nor immediate blocking, but rather spinning for a time that is a sample from an exponential distribution with parameter μ_s^*, before blocking. This result is somewhat surprising in this case where all parameters have exponential distributions.

PROPOSITION 2. *If $a_1/a_2 > b_1/b_2 > c_1/c_2$ (see definitions below)*

$$U(\infty) = \max_{\mu_s} U(\mu_s) = a_1/a_2,$$

which means that immediate blocking is the best choice.

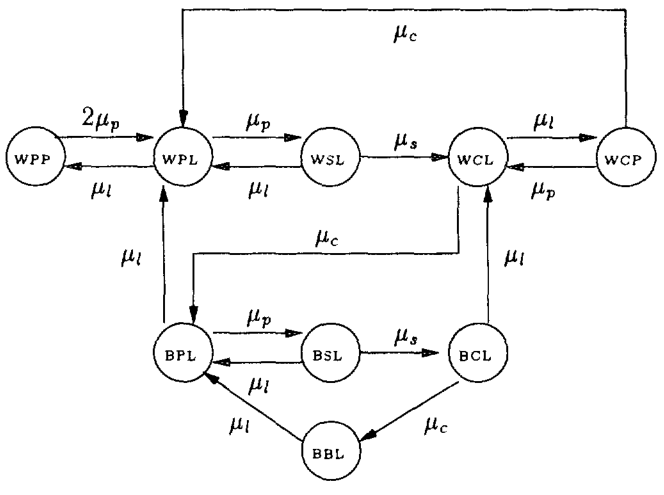

FIG. 3. Markov state transition diagram for the case $J = 3$, $N = 2$.

If $c_1/c_2 > b_1/b_2 > a_1/a_2$

then pure spinning is the optimal strategy, leading to

$$U(0) = \max_{\mu_s} U(\mu_s) = c_1/c_2,$$

where a_i, b_i, *and* c_i *are given by*

$$\begin{aligned}
a_1 &= \mu_l\mu_c(2\mu_p\mu_l + \mu_p\mu_c + (\mu_c + \mu_l)^2) + 2\mu_p^2(\mu_p\mu_l \\
&\quad + \mu_l^2 + \mu_l\mu_c) \\
a_2 &= \mu_l^2\mu_c(2\mu_p\mu_l + \mu_p\mu_c + (\mu_c + \mu_l)^2) + \mu_p\mu_l^2(\mu_p \\
&\quad + \mu_c)(\mu_l + \mu_p + \mu_c) + \mu_p\mu_l\mu_c(\mu_p\mu_c + \mu_c^2 \\
&\quad + \mu_p\mu_l + \mu_l\mu_c) + \mu_p^2\mu_c(\mu_p + \mu_c)(\mu_c + \mu_l) \\
b_1 &= \mu_l^2\mu_c(3\mu_p\mu_l + 2\mu_p\mu_c + 2(\mu_c + \mu_l)^2) + 2\mu_p^2\mu_l^2(\mu_l \\
&\quad + \mu_c) \\
b_2 &= \mu_l^3\mu_c(3\mu_p\mu_l + 4\mu_l\mu_c + 2\mu_p\mu_c + 2\mu_l^2 + 2\mu_c^2) \\
&\quad + \mu_p\mu_l^2\mu_c(2\mu_p\mu_l + \mu_p\mu_c + (\mu_c + \mu_l)^2) \\
&\quad + \mu_p\mu_l^3(\mu_p\mu_l + \mu_l\mu_c + \mu_p\mu_c + \mu_c^2) + \mu_p\mu_l\mu_c(\mu_p\mu_c \\
&\quad + \mu_p\mu_l + \mu_c^2 + \mu_l\mu_c)(\mu_l + \mu_p) \\
c_1 &= 1 \\
c_2 &= \mu_l + \mu_p.
\end{aligned}$$

The proofs of Proposition 1 and 2 follow directly from the characteristics of the formula for U, and are given elsewhere [4].

3.2. Numerical Examples

Figure 4 indicates the combinations of the parameters where one of the immediate blocking, pure spinning or spinning and blocking is optimal, in the case $J = 3$, $N = 2$. Dotted curves indicate the set of points $(T_c/T_l, T_p/T_l)$ for which the spin/block strategy is guaranteed to be optimal. Solid curves indicate the points for which immediate blocking (left curve), or pure spinning (right curve) is guaranteed to be optimal. At selected points along the dotted curve (indicated by the arrows), near optimal values of the spin time T_s are shown. Note that if $T_p/T_l < \infty$, pure spinning can be the best choice even if $T_l > T_c$. Let U_{spin} be the resource utilization in pure spinning case, and U_{block} be the resource utilization in the immediate blocking case. As an example, if $T_p/T_l = 1$ and $T_c/T_l = 0.95$ then $U_{\text{spin}} > U_{\text{block}}$ and the difference between U_{spin} and U_{block} is about 4%. So this is the cost of making the wrong choice in the simple case ($J = 3$, $N = 2$). If $T_p/T_l \to \infty$ then $U_{\text{spin}} > U_{\text{block}}$ if $T_c/T_l > 1$, and $U_{\text{spin}} < U_{\text{block}}$ if $T_c/T_l < 1$. This rule can be applied for any T_p/T_l ratio greater than 5, since the consequent loss in utilization is negligible.

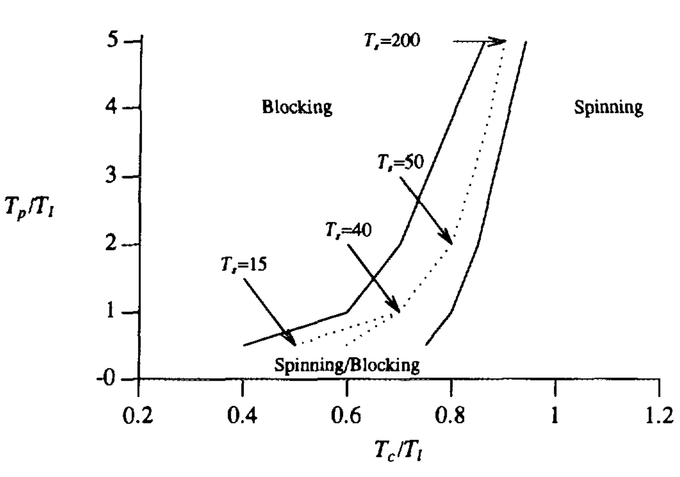

FIG. 4. Optimal strategies classification for $J = 3$ and $N = 2$ ($T_l = 60$).

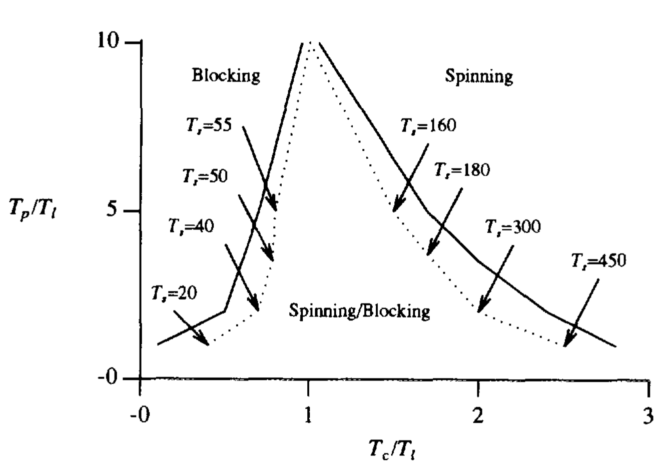

FIG. 5. Optimal strategies classification for $J = 5$ and $N = 4$ ($T_l = 60$).

The values of T_s in Fig. 4 identify the approximate optimal spinning times for selected points on the curves. Note that the benefits that can be derived with the spinning then blocking (spinning/blocking) policy in this case is small. As an example for $T_c/T_l = 0.5$ and $T_p/T_l = 0.5$, the utilization can be improved only by 0.6% relative to pure spinning or immediate blocking.

If $T_c \to 0$, then immediate blocking is the best choice for all other combinations of parameter values. Consider the behavior of the measure $M_B = \lim_{T_c \to 0} U_{\text{block}}/U_{\text{spin}}$. For $J = 3$ and $N = 2$, we have

$$\begin{aligned}
M_B &= 1.070 \quad \text{if } T_p/T_l = 1 \\
M_B &= 1.060 \quad \text{if } T_p/T_l = 2 \\
M_B &= 1.021 \quad \text{if } T_p/T_l = 5 \\
M_B &= 1.008 \quad \text{if } T_p/T_l = 10.
\end{aligned}$$

Thus, the more dominant the processing time is relative to resource holding time, the less the choice of spinning/blocking strategy matters. If $T_c \to \infty$ then pure spinning is the best choice and $U_{\text{block}} \to 0$. Figure 5 indicates which policy is optimal for various parameter value combinations in the case $J = 5$, $N = 4$ (as determined by simula-

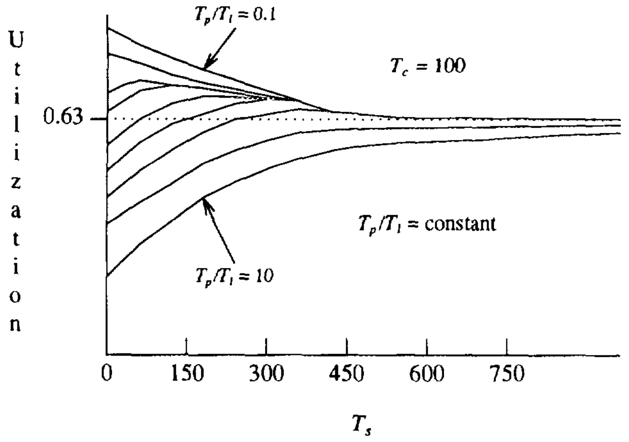

FIG. 6. Utilization versus T_s.

tion). In this case we have found that spinning/blocking is better than pure spinning even if $T_c > 2.5T_l$ for sufficiently small T_p/T_l. Figure 6 shows the effect of various values of T_s on the utilization of the resource. Note that the spin/block strategy may be optimal even in situations where $U_{\text{block}} > U_{\text{spin}}$.

Figure 7a presents an example of the curves for resource utilization corresponding to the cases analyzed above. For the parameter values considered, immediate blocking is optimal for small values of T_c (e.g., $T_c = 20$), while pure spinning is optimal for large values of T_c (e.g., $T_c = 200$). For intermediate values of T_c (e.g., $T_c \simeq 50$) the utilization is insensitive to the spin quantum, but (as is shown in Figure 7b) a maximum value is attained for a spin quantum near $T_s = 220$.

4. APPROXIMATE SOLUTION

In this section, we propose an approximate solution to the model. The idea of our approximate solution is based on the decomposition principle for closed queueing systems with a lock-type node. Consider a closed queueing system with N processors and one lock. If J/N is sufficiently large that the number of idle processors is effectively zero then we can assume that

$$P + S + C + U = N, \tag{1}$$

where P, S, C, U are, respectively, the average number of processors serving threads in their processing phase, spinning, context switching, and serving threads that hold the lock. Now assume that the system provides service to an external flow of transactions with intensity $\bar{\lambda}$, so we have an open queueing system. This means that we assume that the number of processors N in the system is sufficiently large that the rate of threads reaching the point of requesting the shared resource is insensitive to the number that are already spinning or blocked.

PROPOSITION 3. *Let π_k be the stationary probability for the state S_k in which one processor holds the lock, and $k - 1$ are in the spinning state. π_0 is the stationary probability that the lock is free. Then*

$$\pi_k = \pi_0 \prod_{i=0}^{k-1} a_i, \quad k \geq 1, \tag{2}$$

where

$$a_i = \frac{\bar{\lambda}}{\mu_l + i\mu_s} \quad and \quad \pi_0 = \frac{1}{\sum_{k=0}^{\infty} \prod_{i=0}^{k-1} a_i}.$$

$$\left(with \prod_{l=0}^{-1} a_l = 1\right). \tag{3}$$

The proof of the proposition follows when we note that the process is a birth-and-death Markov process. From

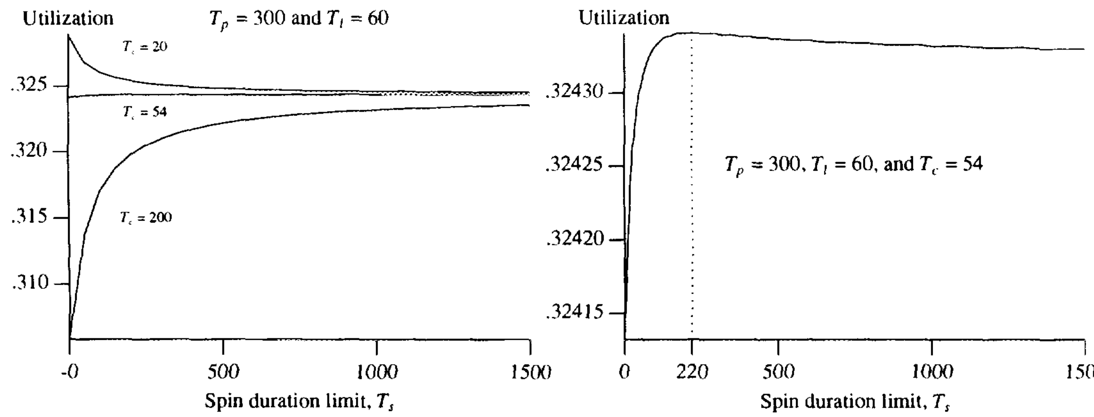

FIG. 7. Resource utilization versus spintime duration limit for different T_c costs. (a) Examples of three optimal cases. (b) Optimal case for spin/block strategy.

state S_k, the process enters the state S_{k-1} with intensity $\mu_l + (k-1)\mu_s$, and the state S_{k+1} with intensity $\tilde{\lambda}$. Now we can start with Eq. (1) and derive an expression for P in terms of N. First, the throughput of the system must satisfy

$$\tilde{\lambda} = P\mu_p = S\mu_s + U\mu_l. \quad (4)$$

Since π_0 is the probability the resource is free, the resource utilization is given by

$$U = 1 - \pi_0. \quad (5)$$

The average number of processors in the context switching state is given by the equation

$$C = \frac{\mu_s S}{\mu_c}. \quad (6)$$

From Eqs. (4) and (5), the average number of spinning processors is given by

$$S = \frac{\tilde{\lambda} - \mu_l(1-\pi_0)}{\mu_s}. \quad (7)$$

Now we return to the closed system satisfying Eq. (1). Using expressions (5)–(7) for C, S, and U in (1), we obtain

$$N = P + (1-\pi_0) + \left(1 + \frac{\mu_s}{\mu_c}\right)\left(\frac{\tilde{\lambda} - \mu_l(1-\pi_0)}{\mu_s}\right).$$

Since $\tilde{\lambda} = \mu_p P$, and letting $R = (1/\mu_s) + (1/\mu_c)$,

$$N = P + (1-\pi_0) + \mu_p PR - \mu_l(1-\pi_0)R. \quad (8)$$

Solving for P yields

$$P = \frac{N - (1-\pi_0)(1-\mu_l R)}{1 + \mu_p R} \quad (9)$$

with

$$\pi_0 = 1 \Big/ \left(\sum_{k=0}^{\infty} \prod_{j=0}^{k-1} \frac{P\mu_p}{\mu_l + j\mu_s}\right). \quad (10)$$

Equations (9) and (10) can be solved iteratively, starting with $\pi_0 = 0$, then updating values of P and π_0 alternately until convergence occurs. Once π_0 is evaluated, the resource utilization is just $1 - \pi_0$.

The validity of this approximation has been investigated throughly in the case of pure spinning ($T_s = \infty$) [4]. The reliability of the approximation increases with larger

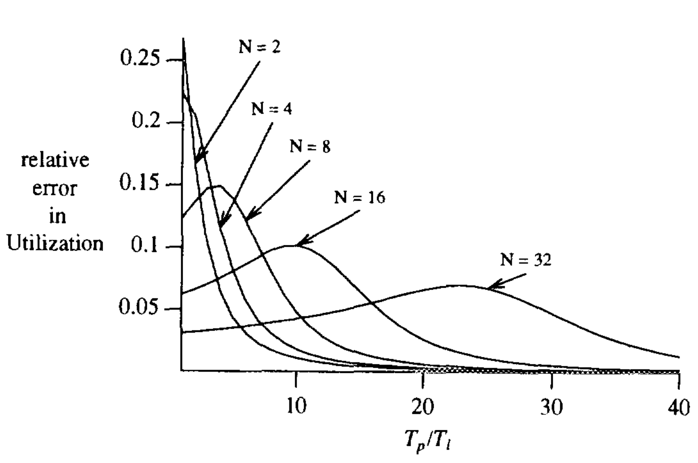

FIG. 8. Relative error versus T_p/T_l for different values of N when $T_s = \infty$.

models as illustrated by Fig. 8. The relative error between the exact value of the utilization and the approximated one when $T_s = \infty$ is displayed for various values of N. In our case where $\rho = 80/3$ the error is less than 1% for all values of N. (See Fig. 8.)

5. SIMULATION RESULTS

A simulator was implemented to study the performance of the spin-then-block strategy under a wide range of parameters and distributions. It was validated with the exact results obtained in Section 3. In multiprogramming systems, high variability in lock holding times may result from a process being preempted while inside a critical section, or due to cache misses, data-dependent execution, page faults, or remote memory access in NUMA (Non-Uniform Memory Access) architectures. In those cases the lock holding time can be modeled by a hyperexponential or a bimodal (we mean two-spike) lock holding time distribution.

In a first experiment, we study the effect of variance on lock utilization for a hyperexponential lock holding time distribution. We fix T_{l1} (e.g., $T_{l1} = 30$), and vary the probability r while keeping the mean T_l the same (e.g., $T_l = 49.4$), so that T_{l2} is equal to $(T_l - rT_{l1})/(1-r)$. All the curves shown later were obtained using the parameters $N = 16, J = 50, T_p = 800, T_{l1} = 30$ in 98% of the requests, $T_{l2} = 1000$ in the remaining 2%, and $T_c = 300$. The limit on spinning time has a deterministic value denoted by T_s. As illustrated by Fig. 9, we observe that as the variance grows (that is, r closer to 1), the choice of spin duration limit becomes more critical. For low variance, it is sufficient to make sure that the spin time is big enough. For high variance, a greater peak in utilization is observed so the spin duration limit must be chosen carefully to attain near maximum throughput. This may also explain why

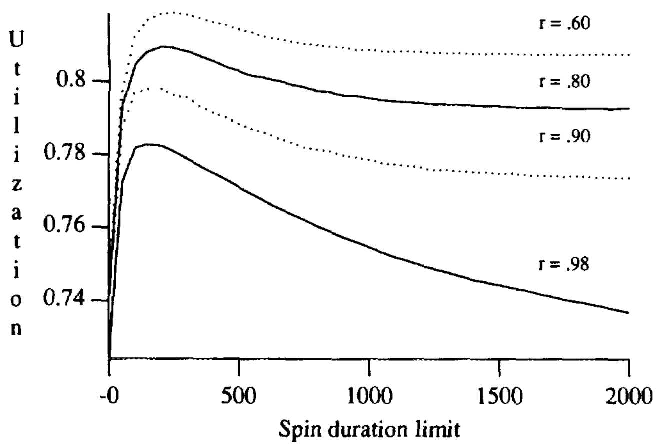

FIG. 9. Effect of lock holding time variance on utilization for different r.

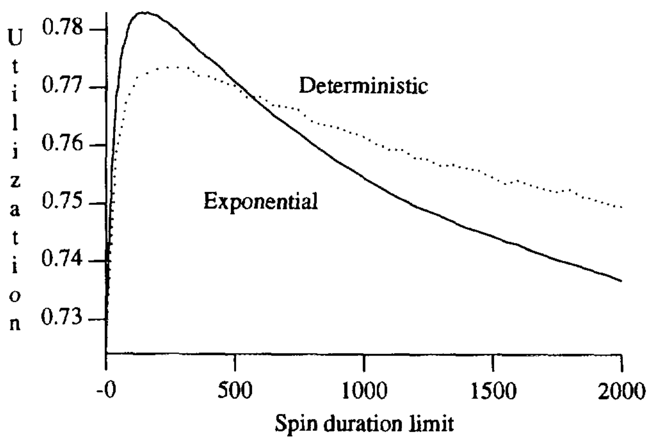

FIG. 11. Deterministic versus exponential spin time limit.

little performance gain is obtainable with the optimal spin/block strategy when lock holding times are exponentially distributed. The variance, which is equal to the square of the mean in that case, is not high enough to make spinning then blocking substantially better than either extreme. All the plots in Fig. 9 reach a maximum at the same value of T_s. The curves of Fig. 10 show the effect of the holding time distribution form on utilization of the lock for different spin time durations. The bimodal distribution is more sensitive to spin duration. Figure 11 illustrates the performance difference between a deterministic spin time limit and an exponential one. It is interesting to note that both curves manifest a peak for the same mean spin time duration. Since the purpose of our investigation is to study the relative (rather the absolute) performance of the spin-then-block strategy, the exponential assumption is not very restrictive. The maximization of utilization involves a clear trade-off between the

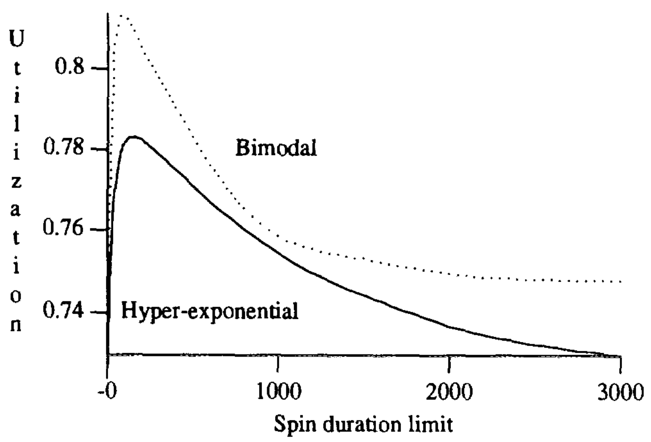

FIG. 10. Bimodal versus hyperexponential lock holding time.

cost of spinning and the cost of context switching. Spinning cost increases and context switching cost decreases as the spin time limit is increased. The curves of Fig. 11 show a pronounced slope on the left side of the optimal spin time threshold. This suggests that it is safer to overestimate the threshold than to underestimate it. If the spin duration limit is too large, then much processor time may be wasted in spinning; if it is too small, then the processor time spent in context switching may be high.

6. CONCLUSIONS

Since synchronization accounts for a large portion of parallel system overhead, an efficient implementation of synchronization is vital to parallel system performance. There exists an important trade-off between processor time lost to spinning and processor time required to save the context of a process when it blocks. The purpose of this study has been to determine an appropriate balance between the extremes of this trade-off. We formulated a queueing theoretic model of resource sharing to study the problem of contention due to synchronization. Our analytical model includes both spinning and blocking effects, and is applicable in a broad variety of situations. Our objective is to provide a better understanding of waiting strategies. In particular, we illustrated the potential for improvements in system performance by selecting an appropriate spinning strategy. Both analytical and simulation results showed that there are domains where the extreme policies are optimal. Despite the Markovian assumptions, our model is still too complicated to obtain exact solutions for arbitrary parameter values. Consequently, an approximate solution based on the decomposition principle has been developed. Simulation has shown that our approximate solution under Markovian assumptions reflects the dynamics of the performance quite faithfully. Qualitatively, the relative performance of pure spinning, immediate blocking, and spinning with blocking are much the same whether lock holding times are exponentially distributed or whether they have higher variance. However, the degree of gain in picking the best strategy over another can become quite large as the variance of the lock holding times grows, whereas the gain is very limited with exponential lock holding times.

There are several sources for uncertainties that can make shared resource holding time quite unpredictable. In future scalable parallel systems, unpredictability will likely be more significant, since the degree of uncertainty tends to scale with the size of the system. A strategy that ignores the high variation in resource sharing time may waste processing power from excessive context switching or spinning. Our simulation results, in which some of the restrictive assumptions made in the analysis were relaxed, allowed us to quantify the utility of the spin-then-block policy and to assess the significance of the variance in resource holding time. It was found that the spin-then-block strategy copes very well with highly variable lock holding times. The results in this paper may be used to determine heuristics useful in developing effective environments for executing parallel programs. For instance, when deriving a spin time threshold value, the penalty of overestimating seems to be much lower than that of underestimating by the same amount.

REFERENCES

1. Anderson, T. E., Lazowska, E. D., and Levy, H. M. The performance implications of thread management alternatives for shared-memory multiprocessors. *IEEE Trans. Comput.* **C-38**, 12 (Dec. 1989), 1631–1644.
2. Anderson, T. E. The performance of spin lock alternatives for shared-memory multiprocessors. *IEEE Trans. Parallel Distribut. Systems* **1**, 1 (Jan. 1990), 6–16.
3. Dinning, A. A survey of synchronization methods for parallel computers. *IEEE Comput.* **22**, 7 (July 1989), 66–77.
4. Boguslavsky, L., Harzallah, K., Kreinen, A., Sevcik, K., and Vainshtein, A. Optimal strategies for spinning and blocking. CSRI Tech. Rep. 277, Univ. of Toronto, Dec. 1992.
5. Dimpsey, R. T., and Iyer, R. K. Modeling and measuring multiprogramming and system overheads on a shared-memory multiprocessor: Case study. *J. Parallel Distribut. Comput.* **12**, 4 (Aug. 1991), 402–414.
6. Karlin, A. R., Li, K., Manasse, M. S., and Owicki, S. Empirical studies of competitive spinning for a shared-memory multiprocessor. *Proc. 13th ACM Symp. on Op. Sys. Prin.* Oct. 1991, pp. 41–55.
7. Karlin, A. R., Manasse, M. S., Mcgeoch, L., and Owicki, S. Competitive randomized algorithms for non-uniform problems. *First Annual ACM Symposium on Discrete Algorithms*. 1989, pp. 301–309.
8. Lim, B.-H. Waiting algorithms for synchronization in large-scale multiprocessors. Master's thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, MA, Feb. 1990.
9. Gupta, A., Tucker, A., and Urushibara, S. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. *ACM SIGMETRICS Conference on Measurement and Modeling Computer Systems*. May 1991, pp. 120–132.
10. Lo, S., and Gligor, V. A Comparative analysis of multiprocessor scheduling algorithms. *7th International Conference on Distributed Computing Systems*. Sept. 1987, pp. 356–363.
11. Mogul, J. C., and Borg, A. The effect of context switches on cache performance. *Proc. 4th International Conference on Architectural Support for Programming Languages and Operating Systems*. Apr. 1991, pp. 75–84.
12. Ousterhout, J. K., Scelza, D. A., and Sindhu, P. S. Medusa: An experiment in distributed operating system structure. *Comm. ACM* **23**, 2 (Feb. 1980), 92–105.
13. Ousterhout, J. K. Scheduling techniques for concurrent systems. *Proc. 3rd International Conference on Distributed Computing Systems*. Oct. 1982, pp. 22–30.
14. Zahorjan, J., Lazowska, E. D., and Eager, D. L. Spinning versus blocking in parallel systems with uncertainty. In T. Hasegawa, H. Takagi, and Y. Takahashi (Eds.). *Proc. Performance of Distributed and Parallel Systems*. North-Holland, Amsterdam, Dec. 1989, pp. 455–472.
15. Zahorjan, J., Lazowska, E. D., and Eager, D. L. The effect of scheduling discipline on spin overhead in shared memory parallel processors. *IEEE Trans. Parallel Distribut. Systems* **2**, 2 (Apr. 1991), 180–198.

LEONID BOGUSLAVSKY is a visiting professor in the Department of Computer Science at University of Toronto. At the same time he holds a position as the Head of the Laboratory of the Institute of Control Sciences, Russian Academy of Science in Moscow. His current research interests are in performance evaluation and modeling of computer systems, with a focus on multiprocessor architecture, computer networks, and parallel processing. Boguslavsky received an M.S. and Ph.D in computer science from the Moscow Institute of Transport Engineering. He is the author of three books, one on traffic control in computer networks, one on design of computer networks, and one on performance evaluation of multiprocessors. He is a member of the American Mathematical Society.

KARIM HARZALLAH is a Ph.D candidate in computer science at the University of Toronto. He received the B.Sc. in computer engineering (honors) from The Pennsylvania State University in 1988, and the M.S. degree in electrical engineering from Purdue University in 1990. His research interests are in the performance evaluation and architecture of parallel systems. He is a member of Eta Kappa Nu, ACM, and IEEE.

ALEXANDER KREININ is a senior research scientist for LVS Corp., Moscow, and is head of the Laboratory of Computers and Controllers Languages at the Institute of Computers and Control, Moscow. He is also an associate professor at the Moscow Institute of Transport Engineering. He received his doctoral degree in 1982 from Vilnius State University, Lithuania. His research interests include the performance evaluation of computer systems, languages for microcomputer control, and applied mathematics.

KENNETH C. SEVCIK is a professor of computer science with a cross-appointment in electrical engineering at the University of Toronto. He is past Chairman of the Department of Computer Science and

also of the Computer Systems Research Institute. He holds degrees from Stanford (B.S., mathematics, 1966) and the University of Chicago (Ph.D., information science, 1971). 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. He is currently a member of Canada's Natural Sciences and Engineering Research Council, the primary funding body for research in Science and Engineering in Canada.

Received April 16, 1992; revised November 6, 1992; accepted January 5, 1993

ALEXANDER VAINSHTEIN is a senior research scientist for LVS Corp., Moscow, and an associate professor in the Department of Applied Mathematics at the Moscow Institute of Transport Engineering. He received the M.Sc. degree in applied mathematics and computer science from the Moscow Institute of Transport Engineering in 1979 and Ph.D in mathematics from the Institute of Control Sciences, USSR Academy of Sciences, in 1987. His research interests include combinatorial algorithms and queueing theory. He is a member of the Moscow Mathematical Society.