Optimal Strategies for Spinning and Blocking *

L. Boguslavsky[†‡], K. Harzallah[†], A. Kreinen[‡], K. Sevcik[†] and A. Vainshtein[‡]

Technical Report CSRI-278
January 1993

† Computer Systems Research Institute
University of Toronto
CANADA

‡ LVS Corporation Moscow, RUSSIA

The Computer Systems Research Institute (CSRI) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is an Institute within the Faculty of Applied Science and Engineering, and the Faculty of Arts and Science, at the University of Toronto, and is supported in part by the Natural Sciences and Engineering Research Council of Canada.

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

Abstract

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.

Index Terms- parallel processing, parallel threads, spinning, blocking, Markov model, simulation.

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, (between spinning whether or not the lock becomes free, and finally blocking when it appears that further spinning is not worthwhile. The spin then block strategy is first proposed by Ousterhout [14]. He noted that operating system functions are invoked with a form of remote procedure call that leads to processor idling. He also observed that in multiprocessor systems the communicating processes generally execute on different processors, so context switching may limit interprocessor communication unnecessarily. As a solution he suggested a two-phase waiting scheme. The idea is to divide waiting into two phases. The first phase is called a *pause*, during which the execution state of the process remains loaded and the process idle. If the pause time expires, and the desired event has not occured yet, then the process enters the block phase and relinquishes its processor. This waiting mechanism was implemented in the Medusa operating system [12] with a user-settable pause time. One critical parameter is the 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. Mogul and Borg [11] propose a context switching cost that takes into account the effect of this loss of cache affinity. They measured the cache-performance cost of context switches and found that the net cost may vary and appears to be in the thousands of cycles for some cache parameters. Anderson et al. show that spin-waiting has a cost above that associated with that of a processor iterating in a tight loop [2]. 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.

Later work by Anderson [3] explores software and hardware alternatives to reduce the cost of spin-waiting. Anderson suggests that the tradeoff between spinning and blocking creates a second order of tradeoff between loss of cache affinity and interconnection network contention. A small spin duration implies a higher degree of locality violation, yet a longer one leads to a greater interconnection contention. Lo and Gligor [10] have compared critical path scheduling and coscheduling, they found that coscheduling achieved better performance. They credit this gain in performance mainly to the use of two-phase waiting which involves spinning for a limited time, then blocking if the awaited event has still not occured. Zahorjan et al. [15] studied the relative performance of spinning and blocking locks in the presence of uncertainty. They examined multiprogrammed and data-dependent environments. They used, as baseline model, a controllable environment, and they tracked the relative degree of degradation due to uncertainty in terms of average number of spinning processors. To model data-dependent behavior, they changed the variability of the lock holding time while keeping the mean constant. Their results show that only considerable variability will have any effect on performance. For the multiprogramming case, they considered a preemptive thread based scheduler. They noticed that at high lock contention, multiprogramming affects performance drastically, due to unscheduling threads while holding the lock. They investigated three approaches that avert idle spinning cycles. The first discipline avoids unscheduling a thread possessing the lock. The second admits critical preemption, but reactivates seized spinning threads only when the lock is liberated. The last strategy is a combination of the previous two. The benefits in performance were found to be good for all three policies. They concluded that neither source of uncertainty complicates the job of selecting the right strategy.

In a later paper, Zahorjan et al. studied the effect of scheduling policies on spin overhead considering job-based policies [16]. They also explored software approaches to reduce the sensitivity of programs to uncertainty, and found the variable self-scheduling scheme to be the most promising. Gupta et al.[9] studied the performance of a simple priority-based scheduler for a set of programs based on pure spinning. Their experiment was conducted by an instruction level simulator for a set of benchmark applications [13]. They deduced that the poor performance was mainly due to busy-waiting for preempted processes. Their compromise was to have the process spin for a quantum equal to the context switch cost before blocking. This resulted in a 37 % increase in overall processor utilization relative to pure spinning.

Recent studies on competitive spinning strategies by Karlin et al. [7] showed that the best possible competitive ratio is two for the deterministic spin strategy, and that a randomized algorithm can achieve strongly competitive ratios approaching $e/(e-1)$. In Later work , Karlin et al. [6] evaluated several competitive waiting schemes for synchronization on a lock. Their measurements showed that immediate blocking performs very poorly. It was found that some applications spend over a third of their elapsed time on context-switches. Their comparative studies of two-phase mechanisms showed that adaptive algorithms are more robust than the non-adaptive ones. Other empirical work by Lim [8] illustrates the strength of two-phase algorithms under various operating circumstances. Dimpsey and Iyer found that in a multiprogramming environment, kernel lock spinning is a major factor in degrading system performance. It accounts for 5 to 10 % of processing power or over one third of the total system overhead [5]. Their correlation analysis revealed that there is a high correlation between each processor's spin time and the times the others spend spinning, meaning that critical section accesses have a global impact in degrading the overall system. Thus, reducing kernel lock spinning is a means of improving system performance.

In most of the previous works, tradeoffs 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 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 General Model of the system 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 General Model

2.1 Model Description

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 Figure 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. We assume that each thread requires a lock with probability p. Otherwise, with probability $1-p$, it releases the processor and, after freeing it, spends a random time in the thinking state and then starts another cycle. 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 computes again in another cycle with probability $1-q$, and with probability q it completes, frees the processor, and enters the thinking phase. There is a restriction 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. In a *positive wake-up* protocol [4] the thread goes to sleep until it is signaled that the lock is free. The *signaled retry* strategy requires that the thread spins again once signaled to do so. The dark rectangle in Fig. 1 encloses phases in which a thread holds a processor.

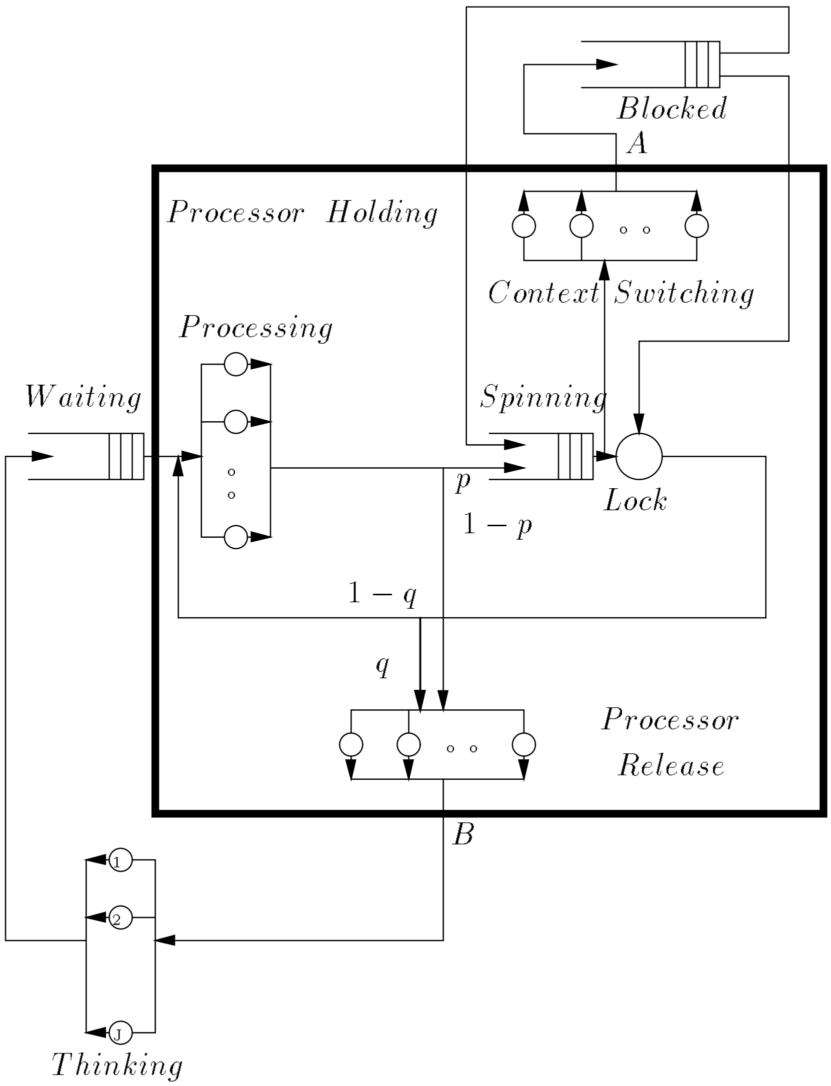

Figure 1: General Model

There are several allocation policies. We have chosen the following rules to determine who 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 on Fig.1), but the lock is still busy, then only waiting threads can acquire the free processor.

2.2 Holding Time Assumptions

In our model, the durations of various activities are governed by the following assumptions:

1. Computing (processing) times and thinking times are exponentially distributed independent random variables with means T_p and T_t respectively.

2. The limit on spinning time either (i) is a sample from an exponential distribution with mean T_s, or (ii) has a deterministic value equal to T_s.

3. Lock holding time is a random variable with either (i) an exponential distribution with mean T_l, or (ii) a hyper-exponential distribution formed from an exponential distribution of mean T_{l1} with probability r, and an exponential distribution of mean T_{l2} with probability $1 - r$, or (iii) a bimodal distribution, where the lock holding time is a constant T_{l1} with probability r, and T_{l2} with probability $1 - r$. (In the latter two cases, $T_l = rT_{l1} + (1 - r)T_{l2}$.)

4. Context switching times and processor release times are either (i) exponentially distributed random variables with means T_c and T_f, or (ii) deterministic values equal to T_c and T_f, respectively.

2.3 Thread and Processor States

Consider the model in a particular case where the positive wake up protocol is used, each transaction accesses the resource exactly once ($p = 1$, $q = 1$), and both think time and processor release time are zero ($T_f = 0$, $T_t = 0$). 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 notice signalling the availability of the lock);

- L – holding a lock.

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

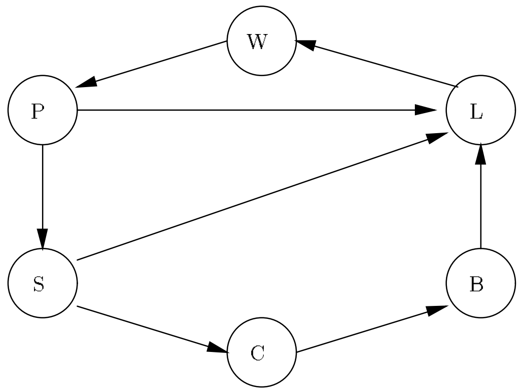

Figure 2: State transition diagram for threads.

Each processor can be in one of the following states:

- I – idle;
- P – computing a thread;
- S – busy by a spinning thread;
- C – busy by context switching;
- L – supporting a thread which 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 on Fig.3.

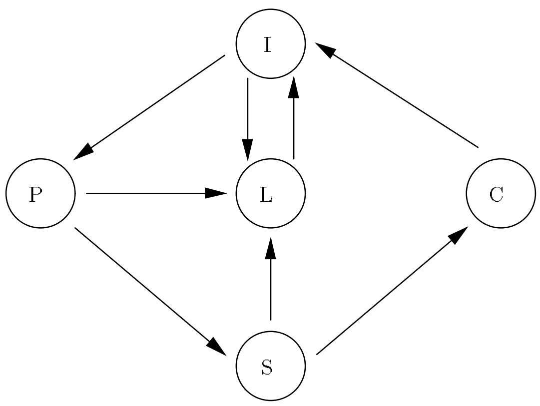

Figure 3: State transition diagram for processors.

2.4 Objective Functions

In our investigations we have used the Resource or Lock Utilization U as the primary performance measure. Resource Utilization is equal to the fraction

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

where $t_i(T)$ is the time that the ith processor spends in the lock holding state L during run-time T. We denote by

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

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:

$$X = \frac{U}{T_l}$$

Denote by S the average number of spinning processors, and by C the average number of processors in context switching mode. The quantities S, C, and U are related by the equation:

$$S + C + P + U + I = N,$$

where I is an average number of idle processors in the system and P is the average number of processors in the computing phase of a thread. 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 first consider the particular case of the General Model identified in section 2.3. The positive wake-up protocol is used, each transaction acquires the lock exactly once, and both think time and processor release time are zero ($p = 1$, $q = 1$, $T_t = 0$, $T_f = 0$). 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 is described by 12 parameters $n_1, \ldots, n_6$, where n_i is the number of threads in state $i = \{W, P, S, C, B, L\}$, $m_1, \ldots, m_5$ where m_j is the number of processors in state $j = \{I, P, S, C, L\}$, and $l = \{0, 1\}$ which indicates the state of the lock as free or busy. However, these 12 parameters are not independent. They satisfy the following equations:

$$\begin{aligned} n_1 + \ldots + n_6 &= J, \\ m_1 + \ldots + m_5 &= N, \\ n_2 &= m_2, \\ n_3 &= m_3, \\ n_4 &= m_4, \\ n_6 &= m_5 = l. \end{aligned}$$

Hence to describe the system one needs only five parameters, say l, n_2, n_3, n_4, and n_5. Thus, states can be coded by (n_2, n_3, n_4, n_5, l).

3.1 Spinning and blocking for case $N = 2$ and $J = 3$

To simplify notation in this particular case 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, C - context switching, and B - blocked. Table 1 enumerates the feasible states of this system. The graph representing the state space for the case $N = 2$, $J = 3$ is given in Fig.4.

State		# W	# P	# S	# L	# C	# B
π_0	WPP	1	2	0	0	0	0
π_1	WPL	1	1	0	1	0	0
π_2	WSL	1	0	1	1	0	0
π_3	WCL	1	0	0	1	1	0
π_4	WCP	1	1	0	0	1	0
π_5	BPL	0	1	0	1	0	1
π_6	BSL	0	0	1	1	0	1
π_7	BCL	0	0	0	1	1	1
π_8	BBL	0	0	0	1	0	2

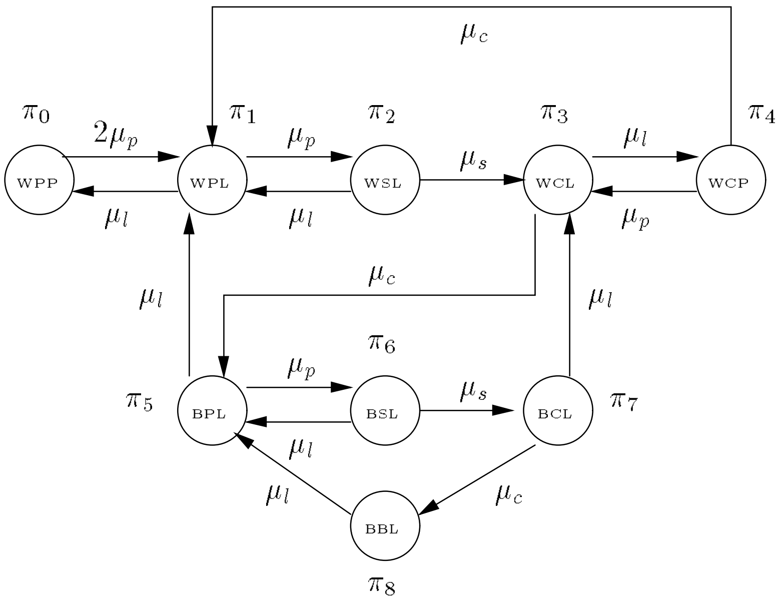

Figure 4: Markov chain for the case $J = 3, N = 2$

Table 1: Distinct states when $N = 2$ and $J = 3$

The stationary probabilities of the states $\pi_i, i = 0, \ldots, 8$ satisfy the equations

$$\begin{aligned}
2\mu_p \pi_0 &= \mu_l \pi_1 \\
(\mu_l + \mu_s)\pi_2 &= \mu_p \pi_1 \\
(\mu_c + \mu_l)\pi_3 &= \mu_l \pi_7 + \mu_s \pi_2 + \mu_p \pi_4 \\
(\mu_p + \mu_c)\pi_4 &= \mu_l \pi_3 \\
(\mu_l + \mu_p)\pi_5 &= \mu_c \pi_3 + \mu_l \pi_6 + \mu_l \pi_8 \\
(\mu_s + \mu_l)\pi_6 &= \mu_p \pi_5 \\
(\mu_l + \mu_c)\pi_7 &= \mu_s \pi_6 \\
\mu_l \pi_8 &= \mu_c \pi_7 \\
\sum_{i=0}^{8} \pi_i &= 1
\end{aligned}$$

$$\text{where} \qquad \mu_p = \frac{1}{T_p}, \quad \mu_s = \frac{1}{T_s}, \quad \mu_l = \frac{1}{T_l}, \quad \mu_c = \frac{1}{T_c}.$$

An average number of processors serving threads in their processing phase is given by

$$P = 2\pi_0 + \pi_1 + \pi_4 + \pi_5$$

The resource utilization is:

$$U = 1 - \pi_0 - \pi_4$$

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^* $(0 < \mu_s^* < \infty)$.*

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. The proof of Proposition 1 is given in the appendix.

3.2 Extreme Cases of Pure Spinning and Immediate Blocking

Assume that $J \geq N$ and the spinning time $T_s = \infty$. This corresponds to the case of pure spinning (see Figure 5). It is easy to see that no processor ever becomes idle. The system's states can therefore be represented by the number of active processors. This model is described by the standard birth-and-death process shown in Figure 6. The stationary probabilities satisfy the following set of equations:

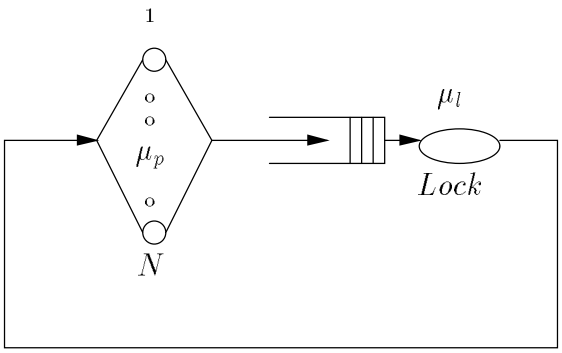

Figure 5: Pure spinning model

$$\begin{aligned} N\mu_p\pi_N &= \mu_l\pi_{N-1} \\ (N-1)\mu_p\pi_{N-1} &= \mu_l\pi_{N-2} \\ &\cdots \end{aligned}$$

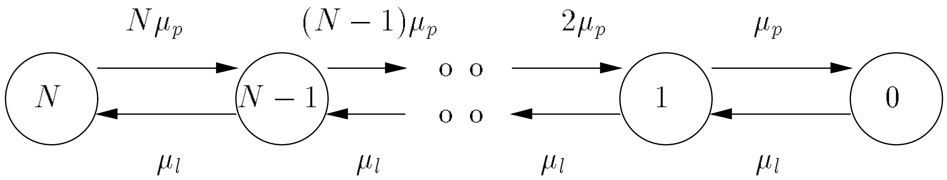

Figure 6: Markov process for pure spinning

$$\begin{aligned} i\mu_p\pi_i &= \mu_l\pi_{i-1} \\ \dots & \\ 2\mu_p\pi_2 &= \mu_l\pi_1 \\ \mu_p\pi_1 &= \mu_l\pi_0 \\ \sum_{i=0}^{N}\pi_i &= 1 \end{aligned}$$

From this set of equations, one can find the stationary probabilities

$$\pi_i \quad = (\rho^i/i!)/\textstyle\sum_{j=0}^{N}(\rho^j/j!) \quad , \quad i = 0, \dots, N \text{ and } \rho = \mu_l/\mu_p$$

Then the average number of spinning threads under pure spinning, $\tilde{S}$, is given by formula

$$\tilde{S} = \sum_{k=0}^{N-1}(N-k-1)\pi_k = N - (1+\rho)(1-\pi_N)$$

The average number of processors serving threads in the computing phase is:

$$P = \sum_{k=1}^{N} k\pi_k = \rho(1-\pi_N)$$

Finally, the resource utilization is given by the formula

$$U = N - \tilde{S} - P = 1 - \pi_N$$

In the case of immediate blocking we have $T_s = 0$ ($\mu_s = \infty$). We found that the complexity of the solution in this case was comparable to that of the general case. For instance, in the particular case where $J = 3$, and $N = 2$, the nine balance equations for a given μ_s reduces to seven equations with $\mu_s = \infty$ (since states π_2 and π_6 disappear).

Proposition 2 *If* $\quad a_1/a_2 > \; b_1/b_2 > \; c_1/c_2 \qquad$ *(see definitions below)*

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

which means that immediate blocking is the best choice.

$$If \quad c_1/c_2 > \; b_1/b_2 > \; a_1/a_2$$

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:

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

The proof of this proposition follows directly from the characteristics of the formula for U in the proof of Proposition 1. (See equation 11 in the Appendix.)

3.3 Numerical Examples

Fig.7 indicates the combination 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 $U_{spin} > U_{block}$ 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 $T_p/T_l \longrightarrow \infty$ then $U_{spin} > U_{block}$ if $T_c/T_l > 1$, and $U_{spin} < U_{block}$ 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.

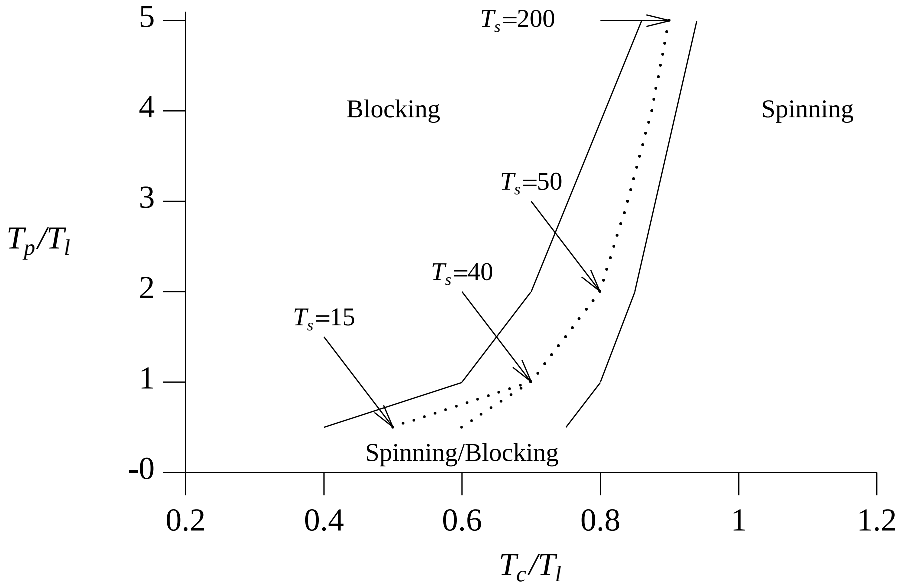

Figure 7: Optimal strategies classification for $J = 3$ and $N = 2$, $(T_l = 60\ )$.

The values of T_s in Fig.7 identify the approximate optimal spinning times for the points on 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 $T_c \longrightarrow 0$, then immediate blocking is the best choice for all other parameters. Consider the behavior of the measure $M_B = \lim_{T_c \longrightarrow 0} U_{block}/U_{spin}$. For $J = 3$ and $N = 2$, we have

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

Thus, the more dominant the processing time is relative to resource holding time, the less the choice of spinning/blocking strategy matters. If $T_c \longrightarrow \infty$ then pure spinning is the best choice and $U_{block} \longrightarrow 0$. Fig.8 presents the areas of parameters where one policy is optimal in the case $J = 5, N = 4$. 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. Fig.9 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 $U_{block} > U_{spin}$.

Fig.10 presents an example of the curves for resource utilization corresponding to the cases analyzed above. For the parameters considered, immediate blocking is optimal for small values

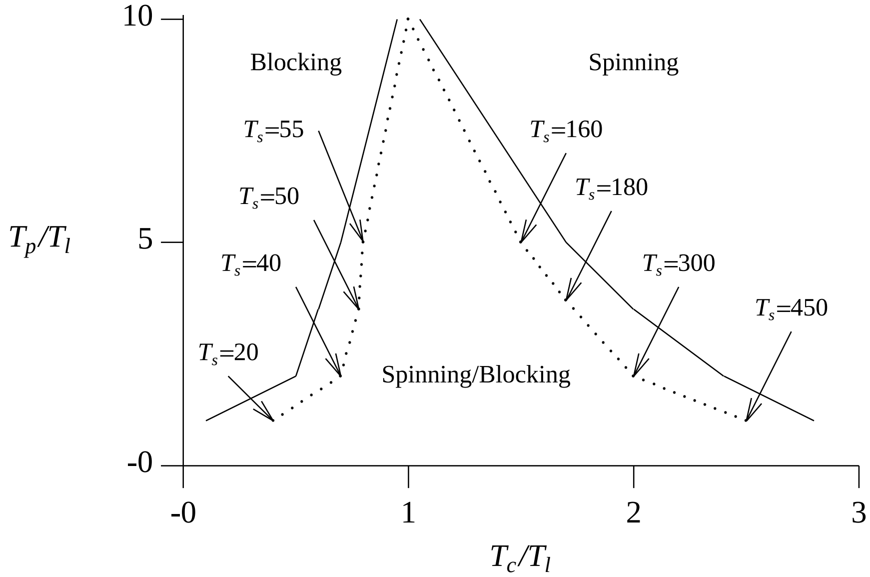

Figure 8: Optimal strategies classification for $J = 5$ and $N = 4$, ($T_l = 60$).

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 Fig.11) 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 $\frac{J}{N}$ 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 is the average number of processors serving threads in their processing phase
- S is the average number of spinning processors
- C is the average number of processors in the context switching state
- U is the utilization of the lock (and also the average number of processors serving threads that hold the lock)

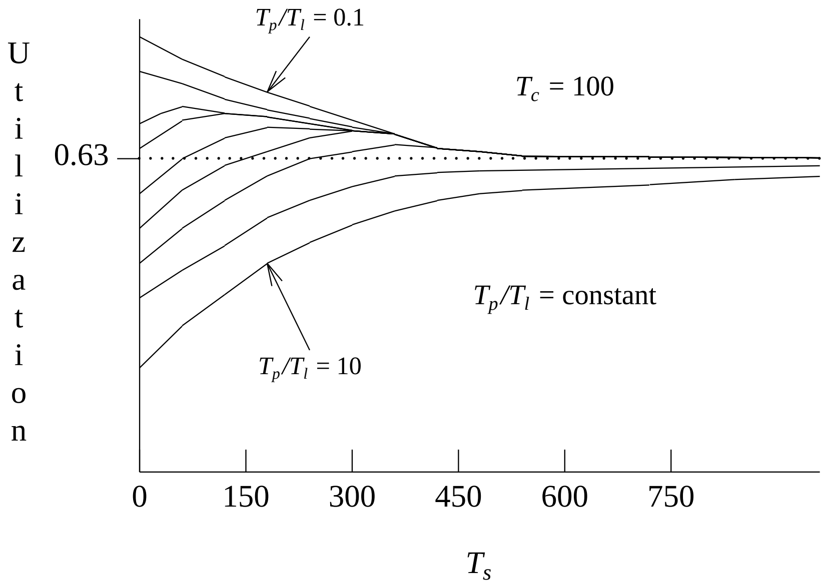

Figure 9: Utilization versus T_s.

Now assume that the system provides service to an external flow of transactions with intensity $\tilde{\lambda}$, 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*

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

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

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

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

$$\tilde{\lambda} = P\mu_p = S\mu_s + U\mu_l \tag{4}$$

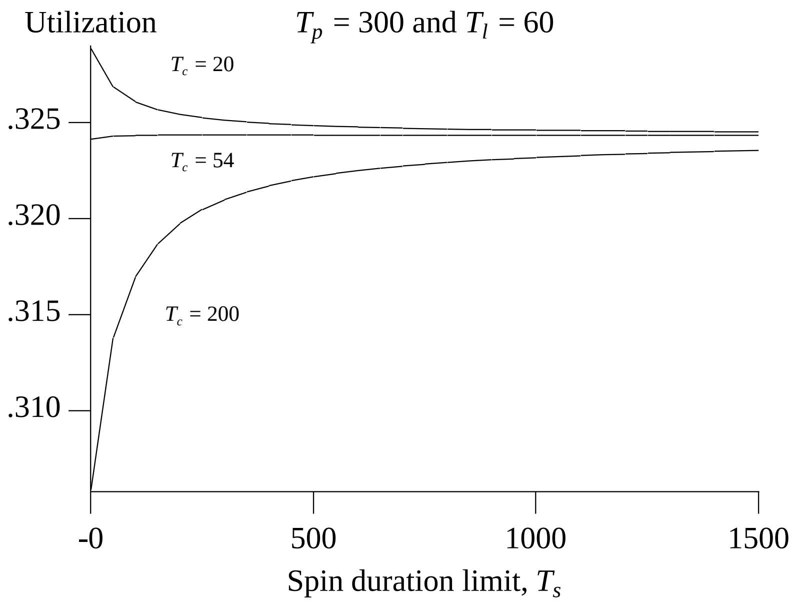

Figure 10: Resource utilization versus spintime duration limit for different T_c costs.

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

$$U = 1 - \pi_0 \tag{5}$$

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

$$C = \frac{\mu_s S}{\mu_c}, \tag{6}$$

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

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

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

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

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

$$N = P + (1 - \pi_0) + \mu_p P R - \mu_l(1 - \pi_0)R \tag{8}$$

Solving for P yields:

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

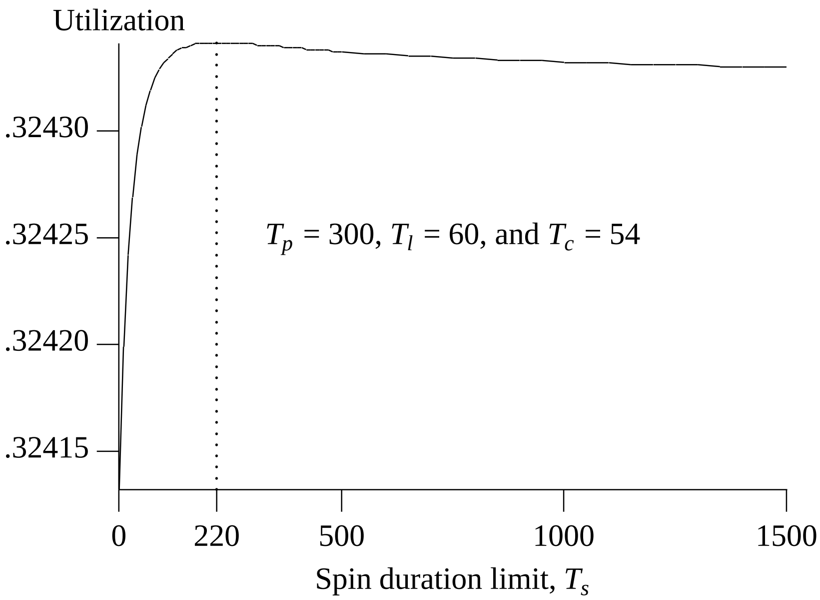

Figure 11: Resource utilization versus spintime duration

$$\text{with} \quad \pi_0 = \frac{1}{\sum_{k=0}^{\infty} \prod_{j=0}^{k-1} \frac{P\mu_p}{\mu_l + j\mu_s}} \tag{10}$$

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

Now we would like to compare our approximation with the exact solution (from section 3.2) for the pure spinning case ($T_s = \infty$). If $T_s = \infty$, then $\mu_s = 0$, and $C = 0$, so that

$$\pi_0 = \frac{1}{\sum_{k=0}^{\infty} (\frac{\tilde{\lambda}}{\mu_l})^k} = 1 - \frac{\tilde{\lambda}}{\mu_l},$$

since $\tilde{\lambda} < \mu_l$ in order for the system to be stable.
This implies that

$$U = \frac{\tilde{\lambda}}{\mu_l}$$

From equation (7)

$$\lim_{\mu_s \longrightarrow 0} S = \lim_{\mu_s \longrightarrow 0} \frac{\tilde{\lambda} - \mu_l(1 - \pi_0)}{\mu_s} = \lim_{\mu_s \longrightarrow 0} \mu_l \frac{\pi_0 - (1 - \frac{\tilde{\lambda}}{\mu_l})}{\mu_s}$$

Noting that the fraction is π_0 less the limit of π_0 as $\mu_s \longrightarrow 0$ all divided by μ_s, we have

$$\lim_{\mu_s \longrightarrow 0} S = \mu_l \frac{\partial \pi_0}{\partial \mu_s}|_{\mu_s=0} = \frac{(\frac{\tilde{\lambda}}{\mu_l})^2}{1-\frac{\tilde{\lambda}}{\mu_l}}$$

Substituting C, S, and U in equation (1) we obtain

$$P + \frac{(\frac{\tilde{\lambda}}{\mu_l})^2}{1-\frac{\tilde{\lambda}}{\mu_l}} + \frac{\tilde{\lambda}}{\mu_l} = N$$

Solving for P and using $\tilde{\lambda} = \mu_p P$ yields P^*

$$P^* = \frac{1}{2}\left(1 + N + \rho - \sqrt{(1+N+\rho)^2 - 4\rho N}\right) \quad \text{where} \quad \rho = \frac{\mu_l}{\mu_p} = \frac{T_p}{T_l}$$

From the balance equation $P^* \mu_p = U \mu_l$ we have

$$U = \frac{P^*}{\rho}$$

As shown in Fig.12, the approximate analysis faithfully predicts the model performance. The reliability of the approximation increases with larger models as illustrated by Fig.13. 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.13)

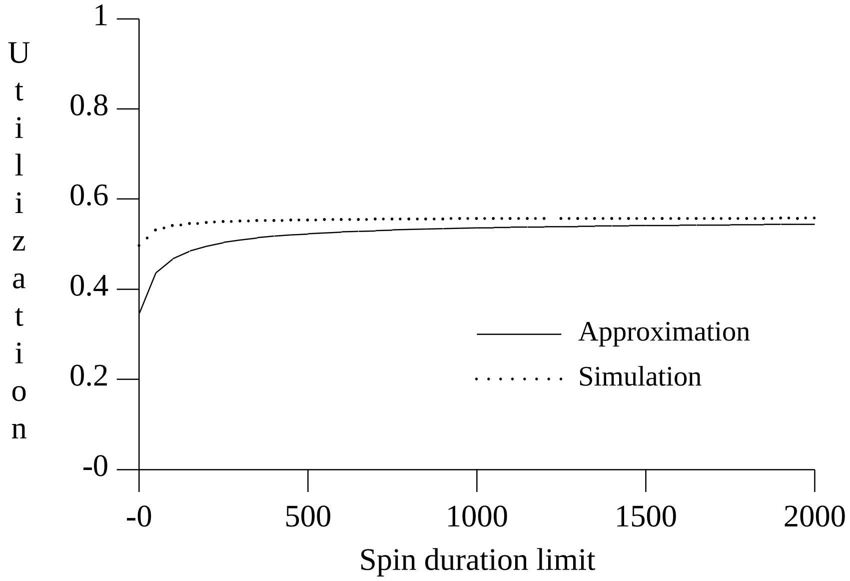

Figure 12: $N = 16$, $J = 50$, $T_p = 800$, $T_l = 30$, and $T_c = 300$

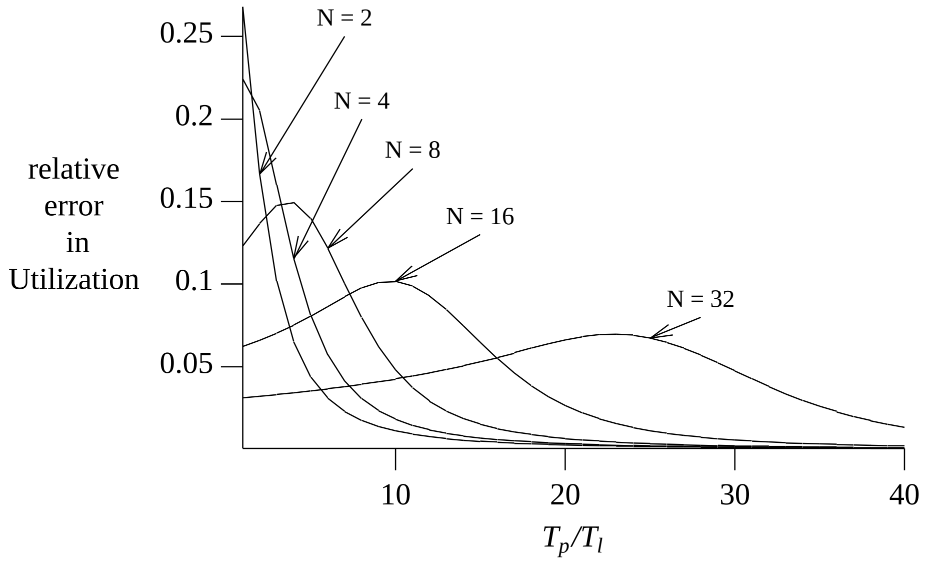

Figure 13: Relative error versus T_p/T_l for different values of N when $T_s = \infty$.

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 from section 3.1, and section 3.2 (for the pure spinning case). In multiprogramming, 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 hyper-exponential 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 hyper-exponential 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 $\frac{T_l - rT_{l1}}{1-r}$. 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.14, we observe that as the variance grows, (that is, r closer to 1), the choice of spin duration limit become 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 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.14 reach a maximum at the same value of T_s. The curves of Fig.15 show the utilization of the lock for different spin time durations. The bimodal distribution is more sensitive to spin duration than the hyper-exponential. Fig.16 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, and since the purpose of our investigation is to study the relative (rather the absolute) performance of the spin then block strategy, the exponential assumption seems to not be very restrictive. Finally, Fig.17 and Fig.18 demonstrate that the maximization of utilization involves a clear trade-off between the cost of spinning and the cost of context switching. The curve of Fig.17 shows a pronounced slope in 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.

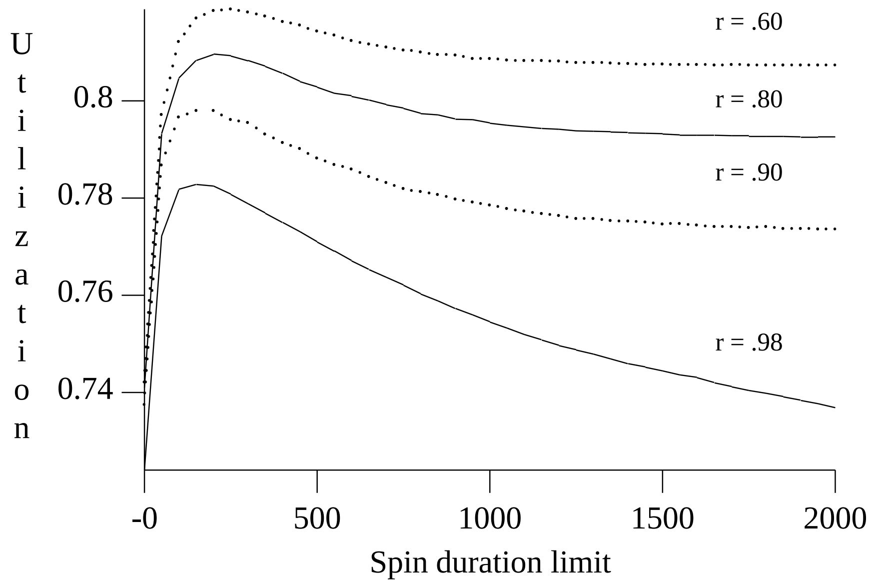

Figure 14: Effect of lock holding time variance on Utilization for different r .

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

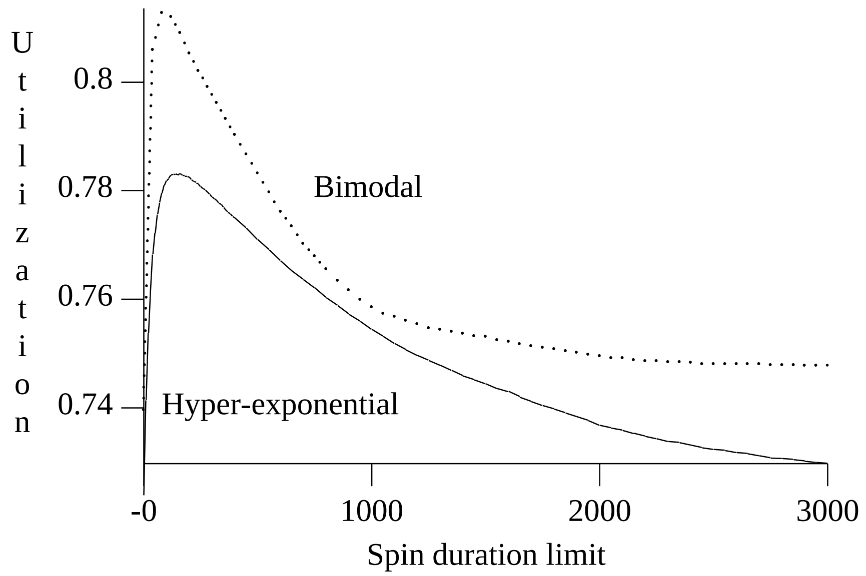

Figure 15: Bimodal versus Hyper-exponential lock holding time.

tradeoff 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 tradeoff. 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 allows the evaluation of a relatively large range of multiprocessor architectures. 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 indicates 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,

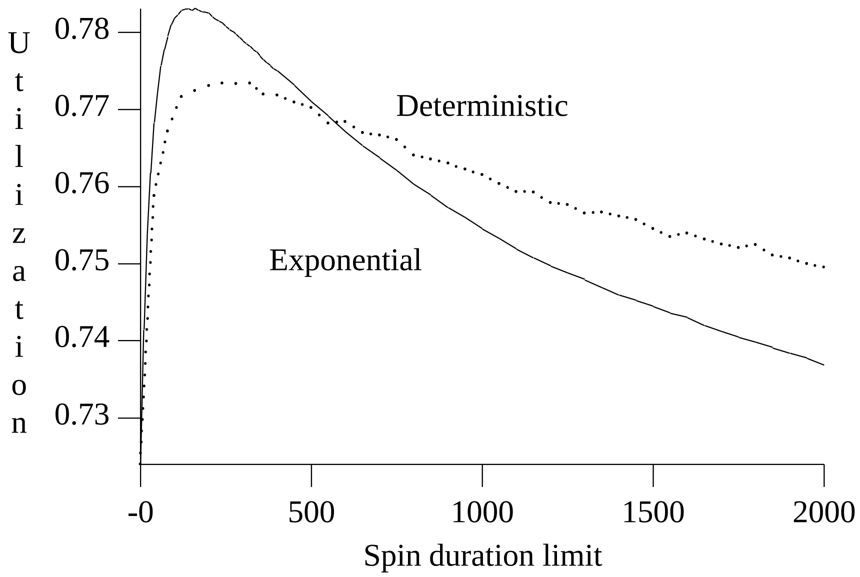

Figure 16: Deterministic versus Exponential spin time limit.

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, obtained by relaxing some of the restrictive assumptions made in the analysis, 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.

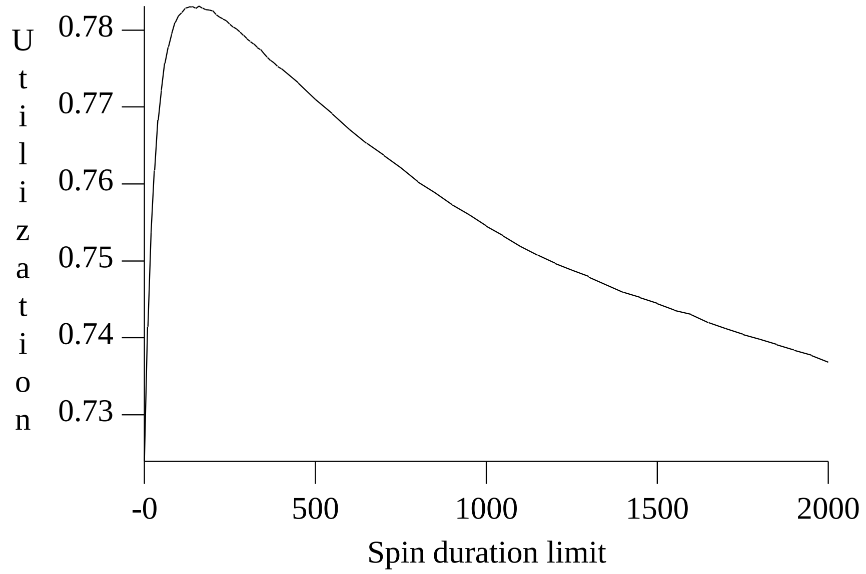

Figure 17: Utilization versus spin time limit

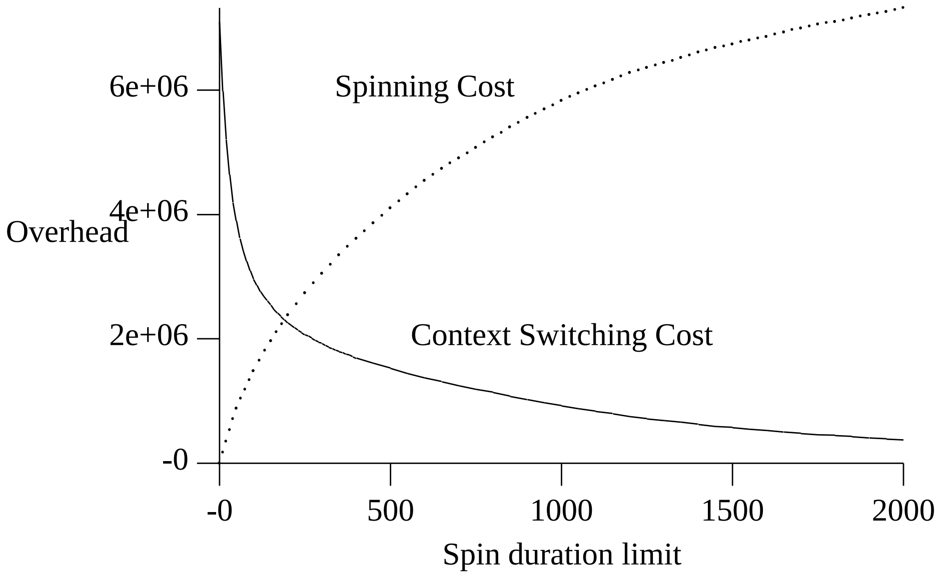

Figure 18: Spinning Cost versus Blocking Cost

References

[1] T.E. Anderson, B.N. Bershad, E.D. Lazowska, and H.M. Levy. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. University of Washington Technical Report 90-04-02, October 1990.

[2] T.E. Anderson, E.D. Lazowska, and H.M. Levy. The Performance Implication of Thread Management Alternatives for Shared-Memory Multiprocessors. In *IEEE Transactions on Computers, Vol. C-38, No. 12 (December 1989), pp. 1631-1644.*

[3] Thomas E. Anderson. The Performance Implication of Spin-Waiting Alternatives for Shared-Memory Multiprocessors. In *Proceedings of the 1989 International Conference on Parallel Processing, pages II:170-174, 1989.*

[4] Anne Dinning. A Survey of Synchronization Methods for Parallel Computers. *Computer, vol 22 (7), pp 66-77, July 1989.*

[5] R. T. Dimpsey, and R. K. Iyer. Modeling and Measuring Multiprogramming and System Overheads on a Shared-Memory Multiprocessor: Case Study. *Journal of Parallel and Distributed Computing 12, pp 402-414, August 1991.*

[6] Anna R. Karlin, Kai Li, Mark S. Manasse, Susan Owicki. Empirical Studies of Competitive Spinning for A Shared-Memory Multiprocessor. *Proc. 13th ACM Symp. on Op. Sys. Prin. pp 41-55, October 1991.*

[7] A.R. Karlin, M.S. Manasse, L. Mcgeoch, and S.Owicki. Competitive Randomized Algorithms for Non-Uniform Problems. In *1st Annual ACM Symposium on Discrete Algorithms, pages 301-309, 1989.*

[8] Beng-Hong Lim. Waiting Algorithms for Synchronization in Large-Scale Multiprocessors. Master's thesis, EECS Department, Massachusetts Institute of Technology, Cambridge, MA, February 1990.

[9] Anoop Gupta, Andrew Tucker, and Shigeru Urushibara. The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications. In *ACM SIGMETRICS Conference on Measurement and Modeling Computer Systems, pp 120-132, May 1991.*

[10] S. Lo and V. Gligor. A Comparative Analysis of Multiprocessor Scheduling Algorithms. In *7th International Conference on Distributed Computing Systems, pages 356-363, Sept. 1987.*

[11] Jeffrey C. Mogul and Anita Borg, The Effect of Context Switches on Cache Performance. In *Proc. 4th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 75-84.(April 1991)*

[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] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared Memory. Technical Report CSL-TR-91-469, Computer Systems Laboratory, Stanford University, April 1991.

[14] John K. Ousterhout. Scheduling Techniques for Concurrent Systems. *Proc. 3rd international Conference on Distributed Computing Systems, pp. 22-30, October 1982.*

[15] John Zahorjan, Edward D. Lazowska, and Derek L. Eager. Spinning Versus Blocking in Parallel Systems with Uncertainty. *Proc. Intern. Seminar Performance of Distributed and Parallel Systems, North Holland, December 1988.*

[16] John Zahorjan, Edward D. Lazowska, and Derek L. Eager. The Effect of Scheduling Discipline on Spin Overhead in Shared Memory Parallel Processors. *IEEE Transactions on Parallel and Distributed Systems, vol 2, pp 180-198, April 1991.*

Appendix

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^* $(0 < \mu_s^* < \infty)$.

Proof: According to Kramer's rule U, is a fraction whose denominator is the determinant of the matrix of system (5) while the numerator equals one minus the sum of two determinants obtained from the above one by replacing specific columns by the vector (0,0,0,0,0,0,0,0,1). Moreover all these determinants regarded as polynomials in μ_s have a degree of at most two. Hence

$$U = \frac{a_1\mu_s^2 + b_1\mu_s + c_1}{a_2\mu_s^2 + b_2\mu_s + c_2}, \tag{11}$$

where a_i, b_i, c_i are certain expressions in μ_p, μ_l, μ_c. Next ,

$$\frac{\partial U}{\partial \mu_s} = \frac{\mu_s^2(a_1b_2 - a_2b_1) + 2\mu_s(a_1c_2 - a_2c_1) + (b_1c_2 - b_2c_1)}{(a_2\mu_s^2 + b_2\mu_s + c_2)^2}$$

hence the necessary and sufficient condition for the existence of an equilibrium is the existence of positive roots for the quadratic equation.

$$\triangle_1 Z^2 + 2\,\triangle_2\, Z + \triangle_3 = 0 \tag{12}$$

with $\triangle_1 = a_1b_2 - a_2b_1, \triangle_2 = a_1c_2 - a_2c_1, \triangle_3 = b_1c_2 - b_2c_1$.
It is evident that equation (14) has exactly one positive root if $\triangle_1 < 0$ and $\triangle_3 > 0$. Direct calculations done by MAPLE give

$$\begin{aligned}
a_1 &= A_1 + A_3 + A_5 + A_7 + A_8 \\
b_1 &= B_1 + B_2 + B_3 + B_5 + B_6 \\
c_1 &= C_1 + C_2 \\
a_2 &= a_1 + A_0 + A_4 \\
b_2 &= b_1 + B_0 + B_4 \\
c_2 &= c_1 + C_0 \\
A_2 &= 0, B_7 = B_8 = 0, C_3 = C_4 = C_5 = C_6 = C_7 = C_8 = 0 \\
A_0 &= \mu_l^3\mu_c(2\mu_p\mu_l + \mu_p\mu_c + \mu_c^2 + 2\mu_c\mu_l + \mu_l^2) \\
A_1 &= 2\mu_p\mu_l^2\mu_c(2\mu_p\mu_l + \mu_p\mu_c + \mu_c^2 + \mu_l^2 + 2\mu_l\mu_c) \\
A_3 &= 2\mu_p^2\mu_l^2(\mu_p^2 + 2\mu_p\mu_c + \mu_p\mu_l + \mu_l\mu_c + \mu_c^2) \\
A_4 &= 2\mu_p^2\mu_l^2(\mu_p\mu_l + \mu_l^2 + \mu_l\mu_c) \\
A_5 &= 2\mu_p^2\mu_l\mu_c(\mu_p\mu_c + \mu_c^2 + \mu_p\mu_l + \mu_l\mu_c)
\end{aligned}$$

$$
\begin{aligned}
A_7 &= 2\mu_p^3\mu_l\mu_c(\mu_p+\mu_c)\\
A_8 &= 2\mu_p^3\mu_c^2(\mu_p+\mu_c)\\
B_0 &= \mu_l^4\mu_c(3\mu_p\mu_l+2\mu_p\mu_c+2\mu_c^2+4\mu_l\mu_c+2\mu_l^2)\\
B_1 &= 2\mu_p\mu_l^3\mu_c(3\mu_p\mu_l+4\mu_l\mu_c+2\mu_p\mu_c+2\mu_l^2+2\mu_c^2)\\
B_2 &= 2\mu_p^2\mu_l^2\mu_c(2\mu_p\mu_l+\mu_p\mu_c+\mu_c^2+2\mu_l\mu_c+\mu_l^2)\\
B_3 &= 2\mu_p^2\mu_l^3(\mu_p\mu_l+\mu_l\mu_c+\mu_p\mu_c+\mu_c^2)\\
B_4 &= 2\mu_p^2\mu_l^4(\mu_l+\mu_c)\\
B_5 &= 2\mu_p^2\mu_l^2\mu_c(\mu_p\mu_c+\mu_c^2+\mu_p\mu_l+\mu_l\mu_c)\\
B_6 &= 2\mu_p^3\mu_l\mu_c(\mu_p\mu_c+\mu_p\mu_l+\mu_c^2+\mu_l\mu_c)\\
C_0 &= \mu_l^5\mu_c(2\mu_l\mu_c+\mu_l^2+\mu_p\mu_l+\mu_p\mu_c+\mu_c^2)\\
C_1 &= 2\mu_p\mu_l^4\mu_c(2\mu_l\mu_c+\mu_l^2+\mu_p\mu_l+\mu_p\mu_c+\mu_c^2)\\
C_2 &= 2\mu_p^2\mu_l^3\mu_c(2\mu_l\mu_c+\mu_l^2+\mu_p\mu_l+\mu_p\mu_c+\mu_c^2)
\end{aligned}
$$

Then it is easy to check that for $\mu_l = 2\mu_p$ and $\mu_l = 0.8\mu_c$ we have $\triangle_1 < 0$ and $\triangle_3 > 0$, which completes the proof .