A Decision-Process Analysis of Implicit Coscheduling

R. Poovendran*, P. Keleher†, and J. S. Baras‡
University of Maryland, College Park
College Park, MD 20742
Contact author: *keleher@cs.umd.edu*

February 3, 2000

Abstract

This paper presents a theoretical framework based on Bayesian decision theory for analyzing recently reported results on *implicit coscheduling* of parallel applications on clusters of workstations. Using probabilistic modeling, we show that the approach presented can be applied for processes with arbitrary communication mixes. We also note that our approach can be used for deciding the additional spin times in the case of spin-yield. Finally, we present arguments for the use of a different notion of fairness than assumed by prior work.

1 Introduction

High performance computing using clusters has become a viable and actively researched area for parallel computing. A central tenet of much of this work is that clusters of commodity networks and machines can provide supercomputer performance at a much lower cost. This paper focuses on the problem of achieving good parallel application performance in such environments, without sacrificing local scheduling autonomy. Specifically, we present new theoretical results on optimal decision-making for systems that use implicit coscheduling [4].

There has been a wealth of recent research in this area [3, 4, 7, 5, 6]. We classify methods of scheduling processes in clusters of workstations in the following three categories:

- *Local Process Scheduling* - Each workstation independently schedules its processes based only on local constraints. This approach is the least complex because it does not require any coordination between local schedulers. However, it can lead to poor performance for parallel applications that exhibit fine-grained communication behavior. Communicating processes are often not scheduled at the same time, leading to the handling of incoming requests being delayed until the destination process is scheduled.
- *Explicit Coscheduling* - This approach requires local schedulers to schedule all of the constituent processes of a given parallel application at exactly the same time. This may be accomplished statically by agreeing upon a global schedule in advance, or dynamically by having a "master" local scheduler direct other schedulers by communicating with them at each context switch. In general, the performance of explicit coscheduling can be expected to degrade if jobs perform heavy I/O or are interactive in nature.
- *Implicit Coscheduling* - This approach allows each of the local schedulers to make decisions independently, but relies on local schedulers to take the communication behavior of local processes into account when making decisions. Local schedulers can converge on coscheduling behavior since each sees similar or related communication behavior by local processes that are part of parallel applications. There are two major forms of implicit coscheduling in the literature. The first is *dynamic coscheduling* [7], which is based on message arrivals only. The second is *two-phase spin-blocking* [3, 4], which makes use of several types of information, such as response time, the nature of message arrivals, and the amount of scheduling progress made by each process.

This paper develops a theoretical framework for analyzing the spin-blocking implicit coscheduling. Several reports have shown that this approach performs as well as other implicit coscheduling approaches, and their approach has been the subject of the most previous analysis. Our results can be extended to the spin-yield approach [5] as well. However, we will focus on spin-blocking in this paper since the experimental setup for two-phase spin-block has been well documented in the literature.

Spin-blocking relies on the observation that a request will likely receive a fast reply to a request if the destination process is already scheduled when the request is sent. Conversely, a slow reply probably means that the destination process is not

Variables	Description
W	message wakeup cost
$2L + 4O$	round-trip time
S_{RBase}	baseline read spin $= 2o + 4L + W$
B	barrier latency
S_{BBase}	baseline barrier spin $= B$
S_{BLocal}	local barrier spin $= S_{BBase} + 2W$
V	max barrier imbalance $= 4W + 2B$
T_{Pair}	additional conditional pairwise spin

Table 1: Spin Variables

currently scheduled. This characteristic allows coscheduling to be achieved by continuing to execute processes whose remote requests complete quickly, and blocking those processes whose remote requests take a long time.

The specific mechanism used is two-phase spin-blocking. The sender of a remote request spins for some amount of time (also referred to as threshold time) while waiting for the request to complete. If a reply has not been received by the threshold time, the sender blocks and another local process runs. It is well understood that for a fixed distribution of waiting times, spinning for a time W is competitive with the optimal online algorithm with a factor of two, where W is the cost of context switching. Despite the fact that this analysis is not directly applicable to this context, extensive simulation and performance results confirm [4] that spinning for an additional time proportional to W works well.

Spin-blocking is usually analyzed in the context of bulk-synchronous applications whose processes interact through remote reads and global barriers. The central contribution of this paper is a general result showing how to arrive at optimal spin times in this case. More specifically, we pose the problem of identifying spin thresholds as a general optimization problem based on process mix information, and use a Bayesian decision approach to find the optimal spin times. Second, we present a decision method that takes into account the cost of scheduling a competing, but wrong process. This method extends prior results by allowing heterogeneous processes. Finally, we argue for a different definition of the fairness metric used to evaluate approaches to scheduling heterogeneous processes.

The rest of the paper is organized as follows. Section 2 reviews previous spin-blocking analysis and shows by example that these formulations are applicable only in limited cases. Section 3 motivates the use of Bayesian analysis. Section 3 shows how to derive previous results for the single process case, and Section 4 presents the analysis for the multiple-process case, and shows that prior analysis is inaccurate even if the processes are identical. Finally, Section 5 discusses the issue of fairness and Section 6 concludes.

2 Background and Motivation

This section reviews prior spin-time formulations that were based on extensive simulations [4]. We show that these formulations are not adequate to handle non-uniform distributions of message arrival rates, or to handle multiple processes with similar or different characteristics. The spin thresholds are summarized in Table 1. We will define these terms as they occur in the text. There are two primary spinning thresholds that need to be derived: the spinning time for a process to be kept in coordination if they are already coscheduled, S_{BBase}, and the spinning time used by a remote process to wait till all the processes reach the barrier, S_{Local}.The remote request spin actually has two components: a *baseline* spin, and an additional amount of spin to be given to processes that handle incoming requests while spinning. This latter spin exists because processes that handle remote requests are contributing to overall forward process, even while otherwise spinning.

2.1 Baseline Read Spin

The basic tenet of this work is that if a process is already coscheduled with remote processes with which it communicates, it is cost effective to keep the processes coscheduled.The one exception to this rule is with regards to providing fairness to competing parallel applications. We discuss fairness in Section 5. In order to keep communicating processes coscheduled, we assume that the initiator of a remote read must spin at least the minimal amount of time required for a message to be handled and responded to. Using the LogP model [2], this *baseline read spin* is$S_{RBase} = (2L+4O+W)$, where L is the network latency, O is the processing overhead, and W is the cost of waking up a process. The wakeup cost W is included to account for the fact that the remote scheduler may elect to wake the destination process if it is not already scheduled. Hence, S_{RBase} is the minimum round trip time when the destination is not already scheduled. A process that sends a remote request and spins for this time should still be spinning when the reply returns, assuming that the destination process is immediately scheduled when the request arrives.

2.2 Incoming Messages

While a process spins waiting for a request to be handled, it may receive and process incoming requests. Overall forward progress can be made by letting processes that will be receiving incoming requests in the near future continue to spin until the requests arrive. In the following, assume that the destination of the remote request is already spinning while waiting for its own request to complete. We can derive the maximum additional spin-wait time, T_{Pair}, by addressing the following two cases:

1. Assume the destination process is scheduled. If the destination spent time T_{Pair} spin waiting when the message arrives, it takes $(2L + 4O)$ units of time for the sender to get response back from destination. Hence the total cost to the sender and destination is $(T_{Pair} + 2L + 4O)$.

2. If the destination is not scheduled, the sender unsuccessfully spins for $(2L + 4O + 2W)$ and blocks; the destination is woken up at a cost of W, sends a reply, and spins for another W; and finally, the sender is woken at a cost of W. The total cost to the system is $(2L + 4O + 5W)$.

Therefore the total cost to the system is less if $(T_{Pair} + 2L + 4O)$ is less than $(2L+4O+5W)$. Therefore, the upper bound of T_{Pair} for which additional spinning is useful is $5W$.

2.3 Barrier Spin and Load-Imbalance

The other spin time needed for bulk-synchronous applications is the spin time at barriers. This spin time consists of two components: the cost of a minimal barrier, S_{BBase}, plus the average amount of expected barrier imbalance. Neglecting the latter component would usually prevent all but the most late-arriving processes from remaining scheduled at barriers, potentially adding to the imbalance seen at subsequent barriers. Given S_{BBase}, the base spin time, S_{BLocal}, is defined to be $(S_{BBase} + W)$ in order to provide competitive behavior. If we assume that the coscheduled processes arrive at the barrier with a uniform distribution, and denote the maximal load imbalance by v, then the average load imbalance is given by $v/2$. Hence, the difference $(v/2 - S_{BLocal})$ is the average additional cost in waiting for barrier synchronization compared to just spinning for baseline time. If this additional cost is less than the time it takes to wake up a sleeping process, then it is cost effective to let the process spin for the load imbalance time. This leads to the decision to spin for the entire load-imbalance if $v/2 < (S_{BBase} + 2W)$. Otherwise, a process spins only up to time of $2(S_{BBase} + 2W)$, and then blocks.

3 A Bayesian Approach to Local Cost-Benefit Analysis

Non-uniform arrival probabilities can result in multiple processes with the same average threshold values, but with different upper bounds for imbalance. This section shows how to choose optimal thresholds that minimize the processor time wasted on spinning and context switching among all competing jobs. We take a probabilistic view of the competing jobs in deriving the necessary *optimal* (optimality in the sense of selecting optimal expected total cost) threshold for each competing job. Aside from the mathematical framework itself, our approach also shows that even the optimal thresholds (which is average value by our formulation to be presented below) vary for each process.

Nagar [5] also argues that the results of [4] do not generalize because the study in [4] assumed Split-C/Active Messages, which has tightly coupled, communicating processes. Most reply messages arrive with very little, if any, delay. However, Nagar et al. do not give a probabilistic model to remedy the problem. Instead they discuss additional variations of deterministic models. We note that our approach may be used to interpret some of the spin-blocking results reported by Nagar [5] as well.

Omitting the algebra for the moment, we can summarize the rest of the section as follows. If the competing process mix is such that there are processes that have higher probabilities of performing communication (higher communication rates), they must, *on average*, have larger (upper bounds on additional waiting times) thresholds. Jobs with lower probabilities of communication should have shorter spin-wait thresholds.

This result can be interpreted as follows. Processes with higher probabilities of communication will not need to wait longer to receive the message, on average, even if allowed longer spin waits. Processes that communicate frequently should be given a *higher chance* of being able of receiving a message, and hence should be allowed to *waste*, if need be, the amount of time they are alloted in spin-waiting in order to make progress. The converse is also true: overall efficiency will benefit if processes that communicate less frequently spin-wait for shorter times, and block more frequently.

We also note that the use of the same thresholds as in [4] for all jobs is equivalent to assuming that there is no penalty in spinning a *wrong job*. We will elaborate on this in detail later.

3.1 Single Process

Deciding whether or not to let a process spin-wait is a binary decision process. The possible outcomes are that either the barrier closes within the allocated threshold time, or not. The decision design space for load-imbalance spin-wait is therefore 2 x 2.The input to the local scheduler can therefore be reduced to a binary event, with event H_1 denoting the inputs corresponding to the process of interest, and event H_0 denoting the inputs corresponding to *all other processes*. Each of the inputs from these two events has a range of possible values, and is random in nature. We denote the observation space corresponding to event H_1 by Z_1, and the space corresponding to event H_0 by Z_0. We denote the entire event space by $Z = Z_0 \cup Z_1$. We denote the incoming input random

variable by Y, and the probability densities of Y under event spaces Z_0 and Z_1 by f_0 and f_1. To be more precise, we say that $f_0 = f(Y = y|H_0)$, and $f_1 = f(Y = y|H_1)$. Each time that a local scheduler makes a binary decision based on the expectation that an event, H_1, will occur, there are four possible outcomes:

1. H_0 was decided and H_0 occurred.
2. H_0 was decided and H_1 occurred.
3. H_1 was decided and H_0 occurred.
4. H_1 was decided and H_1 occurred.

The local scheduler makes correct decisions in cases (1) and (4), and incorrect decisions in cases (2) and (3). Let H_1 mean that the barrier closes, and H_0 mean that it does not. We can then summarize the four decisions and outcomes as follows:

1. The process blocks but the barrier does not close.
2. The process blocks and the barrier does close.
3. The process spins but the barrier does not close.
4. The process spins and the barrier closes.

We note that the case (2) is related to a *miss* and the case (3) is related to a *false alarm* in the context of decision theory [8, 1]. In using Bayesian criteria, we assume that the probabilities of each outcome, $P_0 = P(H_0)$ and $P_1 = P(H_1)$, are known a priori. We note that:

$$P_0 + P_1 = 1 \tag{1}$$

Each of these decisions has an associated cost. For example, if the local process was allowed to spin, but the barrier did not close, then the time spent spinning could possibly have been spent more profitably on another process. Similarly, if the local process blocked and the barrier closed immediately thereafter, unnecessary context switches occur.

We define C_{ij} as the cost associated with making decision in favor of event i given that the true hypothesis is H_j. In particular, the costs for all possible outcomes can be denoted as (a) C_{00} for case 1 , (b) C_{01} for case 2, (c) C_{10} for case 3, and (d) C_{11} for case 4. Since there is no penalty in making correct decision, $C_{00} = C_{11} = 0$. Letting $P(i, j)$ denote the probability that decision is i and the outcome is j, the average cost/risk associated with the decision process can be written as:

$$R = E[C] = C_{01}P(0, 1) + C_{10}P(1, 0) \tag{2}$$

From this equation, after some algebra, we can derive a condition

$$P_1C_{01}f(Y = y|H_1) < P_0C_{10}f(Y = y|H_0) \tag{3}$$

In this case, all the values of Y for which $P_1C_{01}f(Y = y|H_1) > P_0C_{10}f(Y = y|H_0)$ are assigned to event space Z_1. If we let $\Lambda_1 = \frac{f_1}{f_0}$, and $\eta_1 = \frac{p_1}{p_0}$, the decision process reduces to:

$$\text{Decide} \begin{cases} \text{spin-wait} & \text{if } \Lambda_1\eta_1C_{01} > C_{10} \\ \text{block immediately} & else \end{cases}$$

It may be noted that our derivations depend on the ratios of C_{01} or C_{10} only. We now interpret the results reported of Dusseau [4] in terms of the binary decision formula given above. The cost of *False Alarm type* spin-waiting was chosen to be $C_{10} = v/2$. The cost of *Miss type* spinning partially, blocking, and being woken up later was chosen to be $C_{01} = B + 2W$. Using these values, the binary decision process would decide to spin wait if $\Lambda\eta(B + 2W) > v/2$, and to block otherwise. In the case of Dusseau's results, the threshold implicitly assumed that all the processes have identical distributions, and hence set $\eta_1\Lambda_1 = 1$. Then, a binary decision process would decide to spin-wait if $(C_{01} = B + 2W) > (C_{10} = v/2)$, and to block otherwise. This reduces to the condition stated in Section 2.3, i.e. it is cost effective to spin-wait up until $2(B + 2W)$. In general, however, the product $\Lambda_1\eta_1 \neq 1$. From the threshold conditions, we note that for different values of the product term $\Lambda_1\eta_1$, the permissible optimal threshold value on load-imbalance v will vary. For example, if $\Lambda_1\eta_1 = 5$, the optimal spin-wait threshold value will be $v < 10(B + 2W)$.

3.2 Incoming Messages

The analysis for pair-wise cost-benefit is identical except for the values of the cost functions. Section 2.2 showed that overall performance will be improved if a spinning process that handles incoming messages is granted extra spin time. As before, a round trip time is assumed to cost $2L + 4O$. The decision-process costs, derived from various time requirements in [4], are enumerated as follows: (1) $C_{00} = 0$ (i.e. if blocking was correct no penalty is payed), (2) $C_{01} = (2L + 4O + 5W)$.i.e. the penalty for blocking is the flight latency, processing time, "short spin" time and three more block related penalties, (3) $C_{10} = (2L+4O+T)$. i.e. the penalty for latency + processing + additional wait, (4)$C_{11} = 0$. i.e. pay no penalty because the process spins until the barrier closes. With these quantities, the binary decision process would decide to spin wait if $(C_{10} = 2L + 4O + T) < \Lambda_1\eta_1(C_{01} = 2L + 4O + 5W)$, or block otherwise. Again, the canonical choice of threshold discussed in Section 2 can be obtained by setting the product term $\Lambda_1\eta_1 = 1$, leading to a decision to spin wait if $T < 5W$, and to block otherwise. As in the case of local cost-benefit analysis, we note that the threshold values can vary depending on the process densities and probabilities that decide the value of $\Lambda\eta$. For example, if

$\Lambda\eta = 2$, the optimal threshold spin-wait time can be as large as $T < 10W$.

4 Multiple Processes

We now formulate the spin wait time decision problem for the case of multiple processes. This is closely coupled with the issues of fairness and scalability. When there are n processes with process i having probability distribution f_i and the prior probability of conditional message arrival p_i, the thresholds are determined using a set of simultaneous equations

$$\Lambda_i \eta_i C_{0i} \geq C_{i0} + \sum_{j=1; j \neq i}^{j=n} \Lambda_j \eta_j (C_{ij} - C_{0j}) \text{i= } 1, \cdots n \qquad (4)$$

where C_{ij} denotes the cost of predicting that the next message arrival will be for i, when it is actually for j. Also, $\Lambda_i = \frac{f_i}{f_0}$, and $\eta_i = \frac{p_i}{p_0}$. We first show that for the multiple processes case, Dusseau's use of the threshold for multiple processes seem to miss a constant term $5W$ in computing the costs C_{ij}.

?

Values of the penalty terms are $C_{0i} = C_{0j} = (2L+4O+5W)$; f or i, j =$1, \cdots, n$,$C_{i0} = (2L + 4O + T_i)$, and the maximum value of $C_{ij} = (T_i + 2L + 4O + 5W)$.

The last term is due to an incorrect decision to spin-wait process i instead of the correct process j. The penalty is the cost of blocking the current process and waking up the destination process. Substituting these values, the spin-wait threshold for process i reduces to:

$$T_i < \frac{(\Lambda_i \eta_i - 1)(2L + 4O) + \Lambda_i \eta_i 5W}{1 + \sum_{j=1; j \neq i}^{j=n} \Lambda_j \eta_j} \qquad (5)$$

If we set $\Lambda_j \eta_j = 1$, where $j = 1, \cdots, n$, this leads to a threshold for process i of $T_i < 5W/n$ when there are n identical competing jobs. We note that the derivations in [4] doesn't account for the time 5W in the cost assignment of C_{ij}. For heterogeneous processes, however, we can show that the required thresholds are different. This is along the lines of the counter examples we gave earlier. For illustration, Table 2 summarizes a set of values of $\Lambda_i \eta_i$'s and the corresponding values of the thresholds, assuming three competing processes.

From the table, we note that it may be highly cost effective to let processes with fine-grained communication behavior to spin-wait for long period of times. Since they have higher rates of communication, they usually do not need to wait for longer time. However, the larger spin waits indicate the bounds within which *all* their communication can be captured with optimal overall spin-wait costs, as derived using the Bayes decision method.

5 Interpretation for Fairness

Scheduling policies can not be discussed without at least alluding to the issue of fairness. Although we do not explore the issue rigorously here, we do have comments. Specifically, we take issue with the common assumption that a scheduling policy is "fair" if all processes finish at approximately the same time, given equal finishing times in isolation. The reason is that a given process's impact on the execution of other processes is not governed by the amount of time it spends doing useful work, but by the amount of time that it occupies the CPU (and hence prevents other processes from doing useful work). In this light, a "fair" scheduling policy is one that, and any time, has executed a total number of cycles for each process in proportion to desired ratios. For example, if three processes have equal priorities and start at the same time, then the running count of CPU cycles spent on each process should be approximately equal at any given time during the life of the processes. This is regardless of whether the cycles counted for a process were actually spent doing useful work, or was spent spin-waiting.

6 Conclusions

This paper has presented a mathematical framework for analyzing, interpreting, and extending the extensive simulation results for implicit coscheduling reported in the literature. We showed that bounds on commonly cited spin thresholds are the solution to a special case of a general probabilistic solution. In particular, previous results assumed uniform message arrival distributions both within and between processes. Our approach, by contrast, results in a general solution that can accommodate arbitrary distributions.

We intend to extend the formulation to accommodate spin-yielding, as well as spin-blocking, in future work.

References

[1] M. Barkat. *Signal Detection and Estimation*. Artech House, Boston, 1991.

[2] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subromanian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *The Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 262–273, May 1993.

[3] A. C. Dusseau, R. H. Arpaci, and D. E. Culler. Effective distributed scheduling of parallel workloads. In *The 1996*

$\Lambda_1\eta_1$	$\Lambda_2\eta_2$	$\Lambda_3\eta_3$	**Per-Process Thresholds**		
			$\mathbf{T_1}$	$\mathbf{T_2}$	$\mathbf{T_3}$
1	1	1	$5W/3$	$5W/3$	$5W/3$
1	1/2	1/2	$5W/2$	$W - L/5 - 2O/5$	$W - L/5 - 2O/5$
2	1	1	$10W/3 + (2L + 4O)/3$	5W/4	5W/4
10	1	1	50W/3 + 3(2L + 4O)	$5W/12$	$5W/12$
100	1	1	$\approx$ 500W/3 + 33(2L + 4)	$\approx 5W/102$	$\approx 5W/102$

Table 2: Spin Thresholds

ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, 1996.

[4] A. C. Dusseau, D. E. Culler, and A. M. Mainwaring. Scheduling with implicit information in distributed systems. In *The 1998 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, 1998.

[5] S. Nagar, A. Banerjee, A. Aivasubramaniam, and C. Das. A closer look at scheduling strategies for a network of workstations. Technical Report TR CSE-98-009, Department of Computer Science and Engineering, The Pennsylvania State University, October 1998.

[6] J.K. Ousterhout. Scheduling techniques for concurrent systems. In *The 3rd International Conference in Distributed Systems*, pages 22–30, May 1982.

[7] Patrick G. Sobalvarro, Scott Pakin, William E. Weihl, and Andrew A. Chien. Dynamic coscheduling on workstation clusters. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[8] H. L. Van Trees. *Detection, Estimation, and Modulation Theory-Part I*. John Wiley and Sons, New York, New York, 1968.