Modeling and Analysis of Dynamic Coscheduling in Parallel and Distributed Environments*

Mark S. Squillante
IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, NY 10598
mss@watson.ibm.com

Yanyong Zhang, Anand Sivasubramaniam
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
{yyzhang,anand}@cse.psu.edu

Natarajan Gautam
Department of Ind & Mfg Engineering
Pennsylvania State University
University Park, PA 16802
ngautam@psu.edu

Hubertus Franke, Jose Moreira
IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, NY 10598
{jmoreira,frankeh}@us.ibm.com

ABSTRACT

Scheduling in large-scale parallel systems has been and continues to be an important and challenging research problem. Several key factors, including the increasing use of off-the-shelf clusters of workstations to build such parallel systems, have resulted in the emergence of a new class of scheduling strategies, broadly referred to as dynamic coscheduling. Unfortunately, the size of both the design and performance spaces of these emerging scheduling strategies is quite large, due in part to the numerous dynamic interactions among the different components of the parallel computing environment as well as the wide range of applications and systems that can comprise the parallel environment. This in turn makes it difficult to fully explore the benefits and limitations of the various proposed dynamic coscheduling approaches for large-scale systems solely with the use of simulation and/or experimentation.

To gain a better understanding of the fundamental properties of different dynamic coscheduling methods, we formulate a general mathematical model of this class of scheduling strategies within a unified framework that allows us to investigate a wide range of parallel environments. We derive a matrix-analytic analysis based on a stochastic decomposition and a fixed-point iteration. A large number of numerical experiments are performed in part to examine the accuracy of our approach. These numerical results are in excellent agreement with detailed simulation results. Our mathematical model and analysis is then used to explore several fundamental design and performance tradeoffs associated with the class of dynamic coscheduling policies across a broad spectrum of parallel computing environments.

*This research has been supported in part by several NSF grants including Career Award 9701475, 0103583, 0097998, 9988164, 9900701, 9818327 and 0130143.

1. INTRODUCTION

Parallel system scheduling is challenging because of the numerous factors involved in implementing a scheduler. Some of these influencing factors are the workload, native operating system, node hardware, network interface, network, and communication software. The recent shift towards the adoption of off-the-shelf clusters for cost-effective parallel computing makes the design of an efficient scheduler even more crucial and challenging. Clusters are gaining acceptance not just in scientific applications that need supercomputing power, but also in domains such as databases, Internet services and multimedia, which place diverse demands on the underlying system. Further, these applications have diverse characteristics in terms of the computation, communication and I/O operations which raise complications when multiprogramming the system. Traditional solutions that have been used in conventional parallel systems are not adequately tuned to handle the diverse workloads and performance criteria.

Scheduling is usually done in two steps. The first step, spatial scheduling, consists of assigning tasks[1] of an application to processors. The second step, temporal scheduling, consists of time multiplexing the various tasks on the corresponding processor. There is a considerable body of literature regarding spatial scheduling (e.g., refer to [6]) and we do not consider this problem herein, nor do we examine the issue of migrating tasks during execution for better load balance. The focus of this paper is on temporal scheduling, which is an especially important issue on clusters.

With off-the-shelf clusters, it is very tempting and quite natural to use off-the-shelf (native) operating systems, with each node performing its own (temporal) scheduling. However, the lack of a coordinated schedule across the system can result in a task of a job waiting for a message from another task that is not currently scheduled on a remote node. This problem is accentuated in current cluster environments that employ user-level messaging (to reduce latencies and improve bandwidth), wherein the operating system is unaware of the task waiting for a message. Multiprocessors have traditionally addressed this problem using a technique called Coscheduling/Gang Scheduling [18], wherein tasks of a job are scheduled on their respective nodes during the same time quantum.

[1]In this paper, we use the term *tasks* to refer to the processes (as defined by the native operating system) constituting an application to avoid any ambiguities with *stochastic processes*.

However, gang scheduling is not a very attractive/scalable option for off-the-shelf clusters, since it requires periodic synchronization across the nodes to coordinate the effort. Longer time quanta to offset this cost can decrease the responsiveness of the system.

As a result, there have been recent efforts [5, 19, 15] to design a new class of scheduling mechanisms – broadly referred to as "dynamic coscheduling" – which approximate coscheduled execution without explicitly synchronizing the nodes. These techniques use local events (e.g., message arrival) to estimate what is happening at remote nodes, and adjust their local schedules accordingly. They are able to simply adjust task priorities to achieve this goal, leaving it to the native operating system to do the actual scheduling. These scheduling mechanisms have been shown to be easier to implement, incur less overheads, and result in more effective schedules than exact coscheduling in a specific computing environment [24].

The design and performance spaces of dynamic coscheduling mechanisms are quite complex, and a myriad of heuristics are possible to schedule the tasks. Without a unifying performance evaluation framework, it is very difficult to evaluate the benefits and limitations of these different heuristics/mechanisms. Only a few very recent studies [15, 24, 1] have even attempted to compare their merits. With so many different parameters for the parallel system and workload, it is difficult to perform an extensive performance evaluation across the entire spectrum. As a result, it is difficult to say which approaches perform best, and under what conditions, and thus many important design and performance questions remain open. Previous studies have used a set of static jobs [19, 5, 15] or a very specific dynamic arrival pattern [24] on a small-scale parallel system. While one could use experimentation [15] and simulation [1, 24] to study small-scale systems, the suitability of these mechanisms for large-scale systems has not been explored, which is one of the motivating factors for our present study. Further, what is the effect of varying the relative fraction of computation (requiring only CPU resources), communication and I/O components in the application, which represents different parallel workloads? Each dynamic coscheduling mechanism has tunable parameters that can have a significant effect on performance. How can we select values for these parameters to get the best performance from the parallel system? As the underlying hardware or operating system changes (leading to different context-switch costs, interrupt costs, speeding up the computation, faster I/O, etc.), how do the relative benefits of the different alternatives compare? Answers to such questions are crucial for the design of an efficient cluster environment, and a unified performance evaluation framework is needed to answer these and other fundamental design and performance questions.

This paper therefore fills a crucial void in the performance evaluation of emerging parallel scheduling strategies by developing and exploiting the first analytical models and analysis (to our knowledge) that accurately capture the execution of dynamically coscheduled parallel systems. We formulate a general mathematical model of this class of scheduling strategies within a unified framework that allows us to investigate a wide range of parallel computing environments. This general framework supports incorporating different scheduling heuristics, workload parameters, system parameters and overheads, policy parameters, etc. We derive a matrix-analytic analysis of the parallel scheduling models based on a stochastic decomposition, which leads to a vastly reduced state-space representation and in some cases is asymptotically exact as the number of processors increases. Measures of certain aspects of the dynamic parallel system behavior are calculated in terms of the decomposed model solution, and a fixed-point iteration is used to obtain the final solution. In addition to mean job response time and maximum system throughput measures, our analysis provides detailed statistics (e.g., time spent in certain system/application states) that can be used to explain the overall results, to isolate hotspots (for subsequent optimization), and to gain a better understanding of the benefits and limitations of different dynamic coscheduling approaches.

A large number of numerical experiments were conducted, which were used to make detailed comparisons among various design and performance space issues/tradeoffs. We first demonstrate the accuracy of our approach by showing the results to be in excellent agreement with those from detailed simulations of a relatively small parallel system, often being within 5% and always less than 10%. As previously noted, the accuracy of our approach increases with the size of the system and our primary interests are in large-scale parallel systems consisting of many computing nodes. Given the complex dynamic interactions among the different aspects of both the parallel computing environment and the scheduling strategy, these results provide considerable evidence of the benefits of our approach to investigate the fundamental design and performance problems at hand. Our numerous experiments also readily provide and quantify the behavior of very large-scale systems, which has not been possible in any of the previous experimental and simulation-based research studies. Using our models and analysis, we are able for the first time to evaluate the benefits and limitations of the three previously proposed dynamic coscheduling mechanisms under a wide range of different workload and system configurations, thus complementing and considerably extending very recent experimental and simulation-based dynamic coscheduling studies. The advantages of such a flexible and configurable model are demonstrated by conducting studies with different computing environments to explore and understand the fundamental design and performance tradeoffs associated with the various dynamic coscheduling strategies. While the motivation for the model comes from the suitability of dynamic coscheduling mechanisms for cluster environments, the model itself is very generic and can be used to examine the behavior of these mechanisms on a diverse set of parallel and distributed systems subject to a diverse set of workloads.

The remainder of this paper is organized as follows. The next section provides a brief overview of the dynamic coscheduling mechanisms. Section 3 presents the parallel system environment used in our study and defines the corresponding stochastic models. We then derive a mathematical analysis of these parallel dynamic coscheduling models. Section 5 presents a representative sample of the results from our numerical experiments based on this analysis. Finally, Section 6 summarizes the contributions of this work and identifies directions for future research.

2. DYNAMIC COSCHEDULING POLICIES

Exact coscheduling or Gang Scheduling (henceforth referred to as just coscheduling) ensures that the tasks of a job are scheduled on their respective nodes (processors) at the same time. This requires a coordinated effort (requiring synchronization), incurring an additional overhead. Increasing time quanta to offset this cost can decrease the responsiveness of the system. One of the advantages of coscheduling is that the network is also virtualized for each job (and is context switched along with the job), making it possible for the task to directly deposit and pick up messages to/from the network interface (user-level messaging). However, recent network innovations [4, 23] have made it possible to virtualize the network by just using the virtual memory system, and it is thus possible to perform user-level networking even if different jobs are running on the different nodes without compromising protection. This has led to the exploration of dynamic coscheduling mechanisms that try to approximate coscheduled execution without the need for synchronization among the nodes. In this section we briefly review

the three previously proposed dynamic coscheduling mechanisms, namely Spin-Block (SB), Demand-based Coscheduling (DCS) and Periodic Boost (PB). We also describe the system (called Local Scheduling (LS)) which does not attempt to do any coscheduling between the nodes, and leaves it to the native operating system at each node to time share its processor.

2.1 Local Scheduling (LS)

In this mechanism, the native operating system is left to schedule the tasks at each node, with no coordinated efforts across them. Most off-the-shelf/commercial operating system schedulers (including Solaris, Linux, Windows NT, System V UNIX) use some version of the multi-level feedback queue to implement time sharing. There are a certain number of priority levels, with jobs waiting at each level, and a time quantum associated with that level. The job at the head of the highest level is selected to execute for that time quantum, and at the end of this quantum it is inserted at the tail of the next level, and the entire sequence repeats. In this paper, we associate equal time quanta for all priority levels which is the case in Windows NT. When a task initiates an I/O operation, it relinquishes the CPU, and is put in a blocked state. Upon I/O completion, it typically receives a priority boost and is scheduled again.

2.2 Spin Block (SB)

Versions of this mechanism have been considered in the context of implicit coscheduling [5, 2] and demand-based coscheduling [19]. In this scheme, a task spins on a message receive for a fixed amount of time (*spin time*) before blocking itself. The rationale here is that if the message arrives in a reasonable amount of time (spin time), the sender task is also currently scheduled and the receiver should hold on to the CPU to increase the likelihood of executing in the near future when the sender task is also executing. Otherwise, it should block so that CPU cycles are not wasted. When the message does arrive, the task is woken up, and consequently gets a boost in priority to get scheduled again. There are costs associated with blocking and waking up (requires an interrupt on message arrival). Both interrupt costs and spin time will have a bearing on the resulting performance, and need to be studied. Our model is flexible enough to allow either fixed or variable (adaptive) spin times and the model derivation allows for incorporating several heuristics that can be used to tune the spin times adaptively. However, in the results part of this paper, we focus on a fixed spin model for comparisons, with the hope of gaining insights that can be used to design both fixed and variable (adaptive) spin time mechanisms.

2.3 Demand-based Coscheduling (DCS)

Demand-based coscheduling [19] uses an incoming message to schedule the task for which it is intended, and preempts the current task if the intended receiver is not currently scheduled. The underlying rationale is that the receipt of a message denotes the higher likelihood of the sender task of that job being scheduled at the remote workstation at that time. Upon message arrival, if the intended task is not currently scheduled, an interrupt overhead is paid for re-scheduling that task. The *interrupt* costs will have a bearing on the resulting performance.

2.4 Periodic Boost (PB)

This mechanism has been proposed as an interrupt-less alternative to address the inefficiencies arising from scheduling skews between tasks. Instead of immediately interrupting the host CPU on message arrival as in DCS, the actions are slightly delayed. A kernel activity becomes active periodically to check if there is a task with a pending (unconsumed) message, and if so it boosts the priority of that task. Even if there is no such task, but the currently scheduled one is busy waiting for a message, the activity boosts another task with useful work to do. Though interrupts are avoided, there is the fear of delaying the actions more than necessary. At the same time, making this too frequent would increase the overheads. The impact of the *frequency* of the periodic mechanism, and the *associated* costs of the activity on the resulting performance need to be studied.

3. SYSTEM AND MODELS

Our stochastic models of the parallel systems and workloads of interest in this study are based on the broad spectrum of parallel computing environments found in practice [7, 11, 12, 21]. Similarly, our stochastic models of the above dynamic coscheduling mechanisms are based on actual implementations described in the research literature [19, 15], and the costs used for the various scheduling actions have also been drawn from experimental results.

3.1 Parallel System

We consider a parallel computer system that implements the dynamic coscheduling strategies of Section 2 to allocate system resources to the applications submitted for execution. The parallel system consists of P identical processors, each capable of executing any of the parallel tasks comprising an application. Every processor can operate at a maximum multiprogramming level (MPL), which is usually governed by resource availability (e.g., memory/swap space) to provide reasonable overall performance for the executing jobs. Processors are interconnected by a network that has both a latency and a contention component.

Upon arrival, jobs either specify a certain number of processors that they require (which is equal to the number of tasks in that job), or a range of the number of processors on which they can execute. Each of the tasks belonging to the job is then assigned to a processor that is not already operating at its maximum MPL. Let M denote this maximum MPL for a processor in the system. Even if only one of the tasks cannot be assigned, the job waits in a queue and the scheduler assigns jobs from this queue in a first-come first-serve (FCFS) order to a processor partition of the desired size.

As noted in the introduction, our primary focus in this paper is on temporal scheduling. We mostly consider spatial scheduling with respect to its impact on our modeling approach.

Another primary focus in this paper is on large-scale parallel computer systems. These computing environments can increase the dynamic interactions and complexities of dynamic coscheduling strategies. These parallel environments are also of great interest and play an important role in many scientific and commercial applications. Moreover, it is both difficult and expensive to either experiment with and/or simulate such large-scale computing environments, and thus our analytic models can play an even more important role in better understanding dynamic coscheduling in these environments. We further note that, while our approach provides very good approximations for relatively small-scale systems, an exact formulation (see Section 4) can be used together with some of the results of our analysis to examine even smaller parallel systems.

3.2 Parallel Workload

Parallel jobs arrive to the system from an exogenous source at rate λ. The interarrival times of these jobs are assumed to be independent and identically distributed (i.i.d.) according to the phase-type distribution PH$(\underline{\alpha}, \mathcal{S}^A)$ of order m^A with mean $\lambda^{-1} = -\underline{\alpha}(\mathcal{S}^A)^{-1}\mathbf{e}$ and coefficient of variation c^A, where $\mathbf{e}$ is used to denote the column vector of appropriate dimension containing all ones. An arrival is placed in the FCFS system queue when all M

time-sharing slots are filled in each of the processor partitions of the desired size. The time-sharing quantum lengths and the context-switch overheads at each processor are respectively assumed to be i.i.d following the phase-type distributions PH($\underline{\chi}, \mathcal{S}^Q$) and PH($\underline{\xi}, \mathcal{S}^O$) of orders m^Q and m^O having means $\tau^{-1} = -\underline{\chi}(\mathcal{S}^Q)^{-1}\mathbf{e}$ and $\delta^{-1} = -\underline{\xi}(\mathcal{S}^O)^{-1}\mathbf{e}$, and coefficients of variation c^Q and c^O.

The applications comprising the system workload consist of parallel tasks, each of which alternates among several stages of execution in an iterative manner. The number of iterations for each application, N_A, is assumed to follow a (shifted) geometric distribution with parameter p_{N_A}, i.e., $\mathsf{P}[N_A = 1 + n] = (1-p_{N_A})^n p_{N_A}, n \in \mathbb{Z}_+$. We consider a general class of parallel applications in which each iteration consists of a computation stage, followed by an I/O stage, followed by a communication stage (i.e., sending and receiving messages among tasks). The service times of these per-iteration computation, I/O and communication stages are respectively assumed to be i.i.d. according to the order m^B, m^I and m^C phase-type distributions PH($\underline{\beta}, \mathcal{S}^B$), PH($\underline{\eta}, \mathcal{S}^I$) and PH($\underline{\zeta}, \mathcal{S}^C$) with means $\mu^{-1} = -\underline{\beta}(\mathcal{S}^B)^{-1}\mathbf{e}$, $\nu^{-1} = -\underline{\eta}(\mathcal{S}^I)^{-1}\mathbf{e}$ and $\gamma^{-1} = -\underline{\zeta}(\mathcal{S}^C)^{-1}\mathbf{e}$, and coefficients of variation c^B, c^I and c^C. As part of the communication stage, a task may also have to wait for the receipt of a message from a peer parallel task. We also consider an all-to-all communication strategy, where a task may have to wait for messages from all other tasks before proceeding. All of the above stochastic sequences are assumed to be mutually independent.

The probability distributions assumed for the mathematical model parameters are important in that they determine both the generality of our solution and the usefulness of our model in practice. The use of phase-type distributions [16, 13] for all model parameters is of theoretical importance in that we exploit their properties to derive solutions of various instances of our general stochastic scheduling models. It is also of practical importance in that, since this class of distributions is dense within the set of probability distributions on $[0, \infty)$, any distribution on this space for the parallel environments of interest can in principle be represented arbitrarily closely by a phase-type distribution. Moreover, a considerable body of research has examined the fitting of phase-type distributions to empirical data, and a number of algorithms have been developed for doing so; e.g., see [3, 8, 10, 17] and the references cited therein. This includes recent work that has considered effectively approximating heavy-tailed distributions with instances of the class of phase-type distributions in order to analyze performance models. By appropriately setting the parameters of our models, a wide range of parallel application and system environments can be investigated.

4. MATHEMATICAL ANALYSIS

In this section we present a general mathematical analysis of the foregoing parallel scheduling models within a unified framework that allows us to investigate all classes of parallel application and system environments of interest using a single formulation. To the best of our knowledge, there is no previous work that has analytically modeled and analyzed parallel systems under dynamic coscheduling policies. The reasons for this center around the inherent complexity of such a mathematical model given the dynamic and complex interactions among the parallel processors, making an exact analysis intractable. After illustrating some of these problems, we present a very different and novel approach to address the fundamental problems involved in the mathematical modeling and analysis of this complex parallel system. Our analysis is based on a general form of stochastic decomposition in which we derive distributional characterizations of the dynamic interactions and dependencies among the parallel processors under various dynamic coscheduling policies. We then exploit this distributional characterization to obtain a reduced state-space representation, for which we derive a matrix-analytic analysis based on a fixed-point iteration. Our analysis can be shown to be asymptotically exact in the limit as $P \to \infty$ for some versions of dynamic coscheduling [22], and numerous simulation-based experiments demonstrate that this approximation is very accurate even for relatively small-scale parallel systems though our primary interest is on the class of large-scale, high-performance parallel systems.

The parallel dynamic coscheduling models can be represented by the continuous-time stochastic process $\{X(t)\,;\, t \in \mathbb{R}_+\}$, defined on the infinite, multi-dimensional state space given by $\Omega_X = \bigcup_{i=0}^{\infty} \Omega_X^i$ where $\Omega_X^0 \equiv \{(0, j^A) \mid j^A \in \{1, \ldots, m^A\}\}$, $\Omega_X^i \equiv \{(i, j^A, \overline{j}_1^B, \ldots, \overline{j}_P^B) \mid i \in \mathbb{Z}^+,\, j^A \in \{1, \ldots, m^A\}\}$, $\overline{j}_k^B \equiv (i_k, \overline{j}_k^R, j_{k,1}^B, \ldots, j_{k,\min\{i_k,M\}}^B, j_k^Q)$, $i_k \in \{0, \ldots, \min\{i, M\}\}$, $\overline{j}_k^R(\ell) \in \{1, \ldots, \min\{i_k, M\}\}$, $j_{k,\ell}^B \in \{1, \ldots, \sigma_{N_\phi}\}$, $j_k^Q \in \{1, \ldots, m^Q + m^O\}$, $k \in \{1, \ldots, P\}$, $\sigma_n \equiv \sum_{\ell=1}^{n} \phi_\ell$. The state-vector variable i denotes the total number of parallel jobs in the system; j^A denotes the phase of the interarrival process; i_k denotes the number of parallel jobs assigned to processor k; $\overline{j}_k^R(\ell)$ denotes the current index of the ℓ^{th} priority ordered parallel job assigned to processor k (where $\overline{j}_k^R(1)$ denotes the index of the parallel job currently using processor k) when $i_k \geq 1$ and it is defined to be equal to 1 when $i_k = 0$; $j_{k,\ell}^B$ denotes the phase of the *overall service process* (defined below) for the ℓ^{th} job assigned to processor k; j_k^Q denotes the phase of the quantum length process (including context-switch overhead) at processor k; ϕ_ℓ denotes the number of phases in the phase-type distribution for the ℓ^{th} execution stage of the overall service process; and N_ϕ denotes the number of execution stages comprising the overall service process.

The above (exact) formulation exploits known closure properties of phase-type distributions [13], most notably that the convolution of phase-type distributions is also of phase type. As a result, we can generally capture the various stages of execution (e.g., computation, I/O and communication) for the classes of parallel applications of interest via a single phase-type distribution that represents the appropriate combinations of the phase-type distributions for each of these stages of execution as well as other system behavior (e.g., the impact of a task waiting to receive a message). We refer to this combined process as the *overall service process*. Moreover, as described in more detail below, we note that one of the general keys of our approach consists of capturing the various correlations and dynamic behaviors of the parallel system and scheduling policy in the construction and use of this overall service process.

A primary difficulty in analyzing the process $\{X(t)\,;\, t \in \mathbb{R}_+\}$ that records the state of each processor in the system is the size and complexity of its state space Ω_X. This is because the stochastic process is infinite in multiple dimensions and it contains no structure that can be exploited to deal with the combinatorial explosion of the multi-dimensional state space in order to obtain an exact solution. To create a tractable formulation, we first partition the system into (disjoint) sets of processors that are executing the same collection of parallel applications. We approximate each of these processor partitions by assuming that the distribution of the state of each processor in a given partition is stochastically independent and identical to the distribution of the states of the other processors in the partition. For each processor partition, the corresponding decomposed stochastic process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$, representing each individual processor in the partition, then can be solved in isolation by modifying its overall service process to reflect the distributional behavior of the other processors in the partition. Thus, the complex

dependencies among the processors of a partition and their complex dynamic interactions are probabilistically captured, in part, via the overall service process. Performance measures for the entire partition can be obtained by analyzing the model of a single processor in this manner via a fixed-point iteration, and performance measures for the entire parallel system are obtained by combining the solutions for each of the partitions in the system.

This formulation of our approach assumes a specific parallel computing environment in which each of the processors allocated to an application has the same MPL. This is often the case in many large-scale parallel systems that are spatially partitioned together with some type of packing scheme to fill holes in the time-sharing slots. However, our approach also can be used to handle large-scale parallel systems in which the set of processors executing an application has different numbers of applications assigned to them. This is achieved by further partitioning the set of processors allocated to an application based on their MPL. Our approach is then used to solve in isolation the decomposed stochastic process representing each individual processor in each of these subpartitions by modifying its overall service process to reflect the distributional behavior of the other processors in the subpartition, as well as the processors in the other subpartitions. The fixed-point iteration of our approach is extended in a relatively straightforward manner to handle these subpartitions, and the final solution is obtained as described above.

We note that our analysis can include important aspects of real parallel computing environments such as the effects of increased contention for system resources (e.g., the network) which can be incurred by the parallel system as a result of various application and scheduling actions. Due to space limitations, however, this analysis is not presented here and we refer the interested reader to [22].

4.1 Matrix-Analytic Analysis

Consider a particular processor from a specific processor partition. The state space of the corresponding process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$ is given by $\Omega_Y = \bigcup_{i=0}^{\infty} \Omega_Y^i$ where $\Omega_Y^0 \equiv \{(0, j^A) \mid j^A \in \{1, \ldots, m^A\}\}$, $\Omega_Y^i \equiv \{(i, j^A, \overline{j}^R, j_1^B, \ldots, j_{\min\{i,M\}}^B, j^Q) \mid i \in \mathbb{Z}^+,\ j^A \in \{1, \ldots, m^A\},\ \overline{j}^R(\ell) \in \{1, \ldots, \min\{i, M\}\},\ j_k^B \in \{1, \ldots, \sigma_{N_\phi}\},\ j^Q \in \{1, \ldots, m^Q + m^O\}\}$. The state-vector variable i denotes the number of parallel jobs at the given processor; j^A denotes the phase of the interarrival process; $\overline{j}^R(\ell)$ denotes the current index of the ℓ^{th} priority ordered parallel job assigned to the processor (where $\overline{j}^R(1)$ denotes the index of the parallel job currently using the processor) when $i \geq 1$ and it is defined to be equal to 1 when $i = 0$; j_k^B denotes the phase of the overall service process for the k^{th} job assigned to the processor; and j^Q denotes the phase of the quantum length process (including context-switch overhead) at the processor. Recall that N_ϕ denotes the number of execution stages comprising the overall service process, that ϕ_ℓ denotes the number of phases in the phase-type distribution for the ℓ^{th} execution stage of the overall service process, $1 \leq \ell \leq N_\phi$, and that $\sigma_n \equiv \sum_{\ell=1}^{n} \phi_\ell$. We shall refer to Ω_Y^i as *level* i.

Let $y_{i,d} \in \Omega_Y^i$, $1 \leq d \leq D_i$, $i \in \mathbb{Z}_+$, be a lexicographic ordering of the elements of level i, and define $D \equiv \sum_{i=0}^{M-1} D_i$, where D_i denotes the cardinality of level i. We then define

$$\boldsymbol{\pi} \equiv (\boldsymbol{\pi}_0,\ \boldsymbol{\pi}_1,\ \boldsymbol{\pi}_2,\ \ldots), \tag{1}$$

$$\boldsymbol{\pi}_i \equiv (\pi(y_{i,1}),\ \pi(y_{i,2}),\ \ldots,\ \pi(y_{i,D_i})), \tag{2}$$

$$\pi(y_{i,d}) \equiv \lim_{t \to \infty} \mathsf{P}[\, Y(t) = y_{i,d}\,], \tag{3}$$

for $i \in \mathbb{Z}_+$, $y_{i,d} \in \Omega_Y^i$, $1 \leq d \leq D_i$. The limiting probability vector $\boldsymbol{\pi}$ is the stationary distribution for the stochastic process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$, and the value of each of its components $\pi(y_{i,d})$ can be easily shown to represent the long-run proportion of time the system spends in state $y_{i,d} \in \Omega_Y^i$, a property which we exploit later in this section and in Section 5. Assuming this process to be irreducible and positive recurrent, the invariant probability vector is uniquely determined by solving $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ and $\boldsymbol{\pi}\mathbf{e} = 1$, where $\mathbf{Q}$ is the infinitesimal generator matrix for the process.

The generator matrix $\mathbf{Q}$, organized in the same order as the elements of the stationary vector $\boldsymbol{\pi}$, has a structure given by

$$\mathbf{Q} = \begin{bmatrix} B_{00} & B_{01} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots \\ B_{10} & B_{11} & A_0 & \mathbf{0} & \mathbf{0} & \ldots \\ \mathbf{0} & A_2 & A_1 & A_0 & \mathbf{0} & \ldots \\ \mathbf{0} & \mathbf{0} & A_2 & A_1 & A_0 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{4}$$

where B_{00}, B_{01}, B_{10}, B_{11} and A_n, $n = 0, 1, 2$, are finite matrices of dimensions $D \times D$, $D \times D_M$, $D_M \times D$, $D_M \times D_M$ and $D_M \times D_M$, respectively. Furthermore, the matrices corresponding to the non-homogeneous boundary of Ω_Y have structures given by

$$B_{00} = \begin{bmatrix} \Psi_0 & \Lambda_0 & \cdots & \mathbf{0} & \mathbf{0} \\ \Phi_1 & \Psi_1 & \cdots & & \\ \mathbf{0} & \Phi_2 & \cdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & & \\ \mathbf{0} & \mathbf{0} & \cdots & \Phi_{M-1} & \Psi_{M-1} \end{bmatrix}, \quad B_{11} = \Psi_M, \tag{5}$$

$$B_{01} = [\mathbf{0} \cdots \mathbf{0}\, \Lambda_{M-1}]^T, \qquad B_{10} = [\mathbf{0} \cdots \mathbf{0}\, \Phi_M], \tag{6}$$

where Φ_i, Ψ_i and Λ_i have dimensions $D_i \times D_{i-1}$, $D_i \times D_i$ and $D_i \times D_{i+1}$, respectively. The components of the stationary probability vector $\boldsymbol{\pi}$ are then given by [16]

$$(\boldsymbol{\pi}_0,\ \boldsymbol{\pi}_1,\ \ldots,\ \boldsymbol{\pi}_M) \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} + RA_2 \end{bmatrix} = \mathbf{0}, \tag{7}$$

$$\boldsymbol{\pi}_{M+k} = \boldsymbol{\pi}_M R^k, \quad k \in \mathbb{Z}_+, \tag{8}$$

$$(\boldsymbol{\pi}_0,\ \boldsymbol{\pi}_1,\ \ldots,\ \boldsymbol{\pi}_{M-1})\,\mathbf{e} + \boldsymbol{\pi}_M (\mathbf{I} - R)^{-1}\mathbf{e} = 1, \tag{9}$$

where R is the minimal non-negative solution of the equation

$$R^2 A_2 + R A_1 + A_0 = \mathbf{0}, \tag{10}$$

and $\mathbf{I}$ is the identity matrix of order D_M.

While these standard matrix-analytic results make it possible for us to numerically solve the stochastic process in a very efficient manner, instances of the model with large parameter values can cause the dimensions of the boundary submatrices in equation (4) to become quite large. Following the results derived in [20] for infinite quasi-birth-death processes (QBDs), we next establish a theorem that significantly reduces the time and space complexities of computing equations (7) and (9). We also point out that the results in [20] for (both) infinite (and finite) QBDs were derived independently of the analogous results for finite QBDs obtained in [9].

THEOREM 4.1. *Let $\mathbf{Q}$ be irreducible and in the form of (4) through (6). If the minimal non-negative solution R of equation (10) satisfies $sp(R) < 1$, and if there exists a positive probability vector $(\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_M)$ satisfying equation (7), then the components of this probability vector are given by*

$$\boldsymbol{\pi}_k = -\boldsymbol{\pi}_{k+1}\, \Phi_{k+1}\, \widetilde{R}_k^{-1}, \qquad 0 \leq k \leq M-1, \tag{11}$$

$$\boldsymbol{\pi}_M = -\boldsymbol{\pi}_{M-1}\, \Lambda_{M-1}\, \widetilde{R}_M^{-1}, \tag{12}$$

where $\widetilde{R}_0 \equiv \Psi_0$, $\widetilde{R}_k \equiv \Psi_k - \Phi_k \widetilde{R}_{k-1}^{-1} \Lambda_{k-1}$, $1 \leq k \leq M-1$, $\widetilde{R}_M \equiv \Psi_M + RA_2$. Furthermore, when $M > 1$, the vector $\boldsymbol{\pi}_{M-1}$

can be determined up to a multiplicative constant by solving

$$\boldsymbol{\pi}_n \, [\, \omega_n \,] \;=\; \mathbf{0}, \tag{13}$$

$$\boldsymbol{\pi}_n \, \mathbf{e} \;=\; \theta, \qquad \theta \;>\; 0, \tag{14}$$

where $n = M-1$ *and* $\omega_{M-1} = \Psi_{M-1} - \Phi_{M-1}\widetilde{R}_{M-2}^{-1}\Lambda_{M-2} - \Lambda_{M-1}\widetilde{R}_M^{-1}\Phi_M$. *Otherwise, when* $M = 1$, *the vector* $\boldsymbol{\pi}_1$ *can be determined up to a multiplicative constant by solving equations (13) and (14) where* $n = 1$ *and* $\omega_1 = \widetilde{R}_1 - \Phi_1\widetilde{R}_0^{-1}\Lambda_0$. *In either case, the vector* $(\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_M)$ *then can be obtained from (11), (12) and the normalizing equation (9).*

Proof: From known matrix-analytic results, the stationary probability vector $\boldsymbol{\pi}$ is given by equations (7), (8) and (9). Substitution of (5) and (6) into equation (7) shows that the generator matrix for the boundary has the structure

$$\widetilde{\mathbf{Q}} = \begin{bmatrix} \Psi_0 & \Lambda_0 & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \Phi_1 & \Psi_1 & \cdots & & & \mathbf{0} \\ \mathbf{0} & \Phi_2 & \cdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & & & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \Phi_{M-1} & \Psi_{M-1} & \Lambda_{M-1} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \Phi_M & \Psi_M + RA_2 \end{bmatrix}, \tag{15}$$

where we have made use of (8). The invariant vector $(\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_M)$ then satisfies, up to a multiplicative constant, the equations

$$\boldsymbol{\pi}_0\Psi_0 \;+\; \boldsymbol{\pi}_1\Phi_1 \;=\; \mathbf{0}, \tag{16}$$

$$\boldsymbol{\pi}_{k-1}\Lambda_{k-1} \;+\; \boldsymbol{\pi}_k\Psi_k \;+\; \boldsymbol{\pi}_{k+1}\Phi_{k+1} \;=\; \mathbf{0}, \tag{17}$$

$$\boldsymbol{\pi}_{M-1}\Lambda_{M-1} \;+\; \boldsymbol{\pi}_M(\Psi_M + RA_2) \;=\; \mathbf{0}, \tag{18}$$

$$(\boldsymbol{\pi}_0, \cdots, \boldsymbol{\pi}_M)\,\mathbf{e} \;=\; \theta \;>\; 0, \tag{19}$$

for $1 \leq k \leq M-1$. Upon substituting $\widetilde{R}_0, \ldots, \widetilde{R}_M$ into (16) – (18), we obtain equations (11) and (12). The non-singularity of the matrices $\widetilde{R}_k$, $0 \leq k \leq M$, follows from the properties of the irreducible generator matrix and the submatrices in (4) and (15). Substitution of (11) and (12) into (17) for $k = M-1 > 0$ yields equation (13) with $n = M-1$, which then can be used together with (14) to obtain the vector $\boldsymbol{\pi}_{M-1}$ up to a multiplicative constant when $M > 1$. Similarly, when $M = 1$, equation (13) with $n = 1$ is obtained by substituting (11) and $\widetilde{R}_M$ into (18). The remaining components of the vector $(\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_M)$ are determined up to the same multiplicative constant via recurrence using equations (11) and (12). The vector $\boldsymbol{\pi}$ is then uniquely determined by the normalizing equation (9). □

The use of Theorem 4.1 to calculate the probability vector $\boldsymbol{\pi}$ significantly reduces the computational complexity over that of numerically solving equations (7) and (9) directly. In particular, this approach makes it possible to obtain the boundary components of the invariant vector (up to a multiplicative constant) by solving $M+1$ matrix equations of (time and space) complexity $\mathrm{O}(D_i^2)$, $0 \leq i \leq M$, as opposed to solving a single matrix equation of (time and space) complexity $\mathrm{O}((\sum_{i=0}^{M} D_i)^2)$. This makes it possible for us to compute solutions for large instances of the dynamic coscheduling models that are otherwise prohibitively expensive. Moreover, the algorithm (based on Theorem 4.1) used to compute these solutions is numerically stable across a wide spectrum of model parameters. In fact, throughout all of the numerous experiments performed as part of our study, some of which are presented in Section 5, we encountered no numerical stability problems.

4.2 Generator Matrix

The elements of the generator matrix are constructed based on the above formulation and the dynamic transitions among the stages of behavior for the parallel system, which also depend upon the specific version of dynamic coscheduling of interest. A key aspect of our approach consists of capturing these dynamic behaviors (including correlations among the processors) in the overall service process based on their probability distributions as well as the system dynamics, and constructing the stochastic process of the parallel system to probabilistically include these behaviors under the dynamic coscheduling policy. To simply clarify the presentation of our approach, and in the interest of space, we briefly consider a generic example consisting of M parallel applications that time-share a processor partition under a simple policy (very similar to LS) that allocates time-slices to every application in a round-robin fashion. We refer the interested reader to [22] for additional details.

The first three stages of the overall service process for each application represent the computation, I/O and communication stages of execution, and thus $\phi_1 = m^B$, $\phi_2 = m^I$ and $\phi_3 = m^C$ (refer back to the definitions of Ω_Y and corresponding variables). In the present example, we assume that the communication stage of each task consists of sending a *single* message to one other task followed by receiving a *single* message from one other task. (Note that more general cases of sending/receiving multiple messages are addressed within the context of our approach in Section 4.3.) We further assume that the communication stage is comprised of the total processor demands for the send operation and the portion of the processor demands for the receive operation up to the point of checking if the message from another task has arrived; the processor demands of the remainder of the receive operation are included as part of the computation stage of the next iteration. For this reason, we shall henceforth use "send stage" and "communication stage" interchangeably, unless noted otherwise. A context switch occurs on an I/O operation. At the end of the I/O stage for each iteration, the job completes and departs the system with probability p_{N_A}, and otherwise it proceeds to the send stage. Upon completion of this communication stage, the application immediately enters the computation stage of the next iteration if the message to be received has already arrived, which occurs with probability $\widehat{p}_A(\cdot)$. On the other hand, if the message to be received has not arrived, then the sender of the message can be in one of four (generic) states, namely computation, I/O, send and waiting for a message from yet another processor. The corresponding events that the sender of the message is active in the computation, I/O or send stages occur with probability $\widehat{p}_B(\cdot)$, $\widehat{p}_I(\cdot)$ and $\widehat{p}_C(\cdot)$, respectively.

Let T_S denote the maximum number of iterations, at any given time, that a waiting task can be ahead of the task that will be sending the corresponding message, under the dynamic coscheduling policy being examined. To further clarify the presentation, suppose that $T_S = 0$ in our generic example which implies that the sender of a message to any task in a waiting stage must be in the same iteration (albeit in an earlier stage) as the waiting task. (Note that the general case for T_S is considered in Section 4.3.) We therefore have $\phi_4 = m^B$, $\phi_6 = m^C$ and $N_\phi = 6$. The measures $\widehat{p}_A(\cdot)$, $\widehat{p}_B(\cdot)$, $\widehat{p}_I(\cdot)$, $\widehat{p}_C(\cdot)$ and the order ϕ_5 phase-type distribution for stage 5 (as well as the measures $\underline{\beta}'_{(\cdot)}$, $\underline{\eta}'_{(\cdot)}$ and $\underline{\zeta}'_{(\cdot)}$ used below) are all calculated from our model solution and form the basis for a fixed-point iteration that is described in Section 4.3. More general communication strategies are handled in a similar manner. For example, in the case of an all-to-all communication strategy, we derive a formula for the distribution of time that the task waits for messages from all other tasks (based on distributions conditioned on the state of the system), and then we use a phase-type distribution to estimate (or bound) this waiting time distribution.

The corresponding set of transition rates for this generic example is obtained in a straightforward manner based on the formulation and analysis described above. We note that the general case of these transition rules are similarly obtained in a straightforward manner. Moreover, the corresponding set of general rules for other types of applications and dynamic coscheduling policies can be constructed in a similar fashion within the context of our approach. This is in fact exactly what we did for each of the coscheduling policies defined in Section 2, which was then used together with our analysis to generate the numerical results presented in Section 5. In particular, LS is quite similar to the general case of the policy considered above, where the vector $\overline{j}^R(\ell)$ is used to give a priority boost to each job immediately upon the completion of its I/O operation. A similar approach is used to address SB, with the addition of extending the phase-type distribution for the communication stage to include spin times and interrupt costs (as part of the corresponding receive operation). Somewhat more complex priority orderings are maintained in the vector $\overline{j}^R(\ell)$ for the DCS and PB strategies in order to properly capture the local execution of applications under these schemes. In the interest of space we omit these details of our procedure for constructing the generator matrix elements; see [22].

4.3 Fixed-Point Iteration

Equations (7) – (9) and Theorem 4.1 form the solution to our decomposed stochastic model, in terms of the matrix R, provided that certain aspects of the parallel system behavior are known. For the generic example of the previous section, these unknown measures include $\widehat{p}_A, \widehat{p}_B, \widehat{p}_I, \widehat{p}_C, \underline{\beta}', \underline{\eta}', \underline{\zeta}'$ and the order ϕ_5 phase-type distribution for stage 5. Estimates of these measures of the dynamic system behavior are calculated in terms of the decomposed model solution, and a fixed-point iteration is used to obtain the final solution for the processor partition. We now briefly describe our approach, initially focusing on the generic example of the previous section and then turning to the more general case.

Let $h(y)$ be the index of the state $y \in \Omega_Y$ in the lexicographic ordering of the elements of Ω_Y, and let $\mathbf{P} = \mathbf{I} + \mathbf{Q}/\max\{-\mathbf{Q}[k,k]\}$ be the transition probability matrix for the uniformized version of the process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$. We define $\mathcal{D}_\ell(k,u) = \{y \mid y \in \Omega_Y,\, i \geq 1,\, \overline{j}^R(1) = k,\, 1 \leq j^Q \leq m^Q,\, \sigma_{\ell-1}+1 \leq j_k^B \leq \sigma_\ell,\, j_k^B = \sigma_{\ell-1}+u\}$, $\ell = 1,3,4,6$, $\mathcal{D}_\ell(k,u) = \{y \mid y \in \Omega_Y,\, i \geq 1,\, \sigma_{\ell-1}+1 \leq j_k^B \leq \sigma_\ell,\, j_k^B = \sigma_{\ell-1}+u\}$, $\ell = 2,5$, for $k = 1,\ldots,M$ and $u = 1,\ldots,\phi_\ell$. Also, define $\widehat{\mathcal{D}}_\ell(k,u) = \{z \mid z \in \Omega_Y,\, \mathbf{Q}[h(z),h(y)] > 0,\, y \in \mathcal{D}_\ell(k,u)\}$, $\mathcal{D}_\ell(k) = \bigcup_{u=1}^{\phi_\ell} \mathcal{D}_\ell(k,u)$, $\widehat{\mathcal{D}}_\ell(k) = \bigcup_{u=1}^{\phi_\ell} \widehat{\mathcal{D}}_\ell(k,u)$, for $\ell = 1,\ldots,6$, $k = 1,\ldots,M$, $u = 1,\ldots,\phi_\ell$.

To calculate the vectors $\widehat{p}_B$, $\widehat{p}_I$ and $\widehat{p}_C$ in terms of the decomposed model solution, recall that these measures represent the set of probabilities that the sender of the message being waited for (by the receiving task being modeled) is active in the computation, I/O and send stages, respectively. Due to the assumptions of stochastically independent and identical processors as part of our approximate matrix-analytic solution, these probability vectors then can be respectively expressed as

$$\widehat{p}_B(k) = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{M-1})\widehat{v}_{1,k}^b + \boldsymbol{\pi}_M(\mathbf{I}-R)^{-1}\widehat{v}_{1,k}^r, \quad (20)$$

$$\widehat{p}_I(k) = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{M-1})\widehat{v}_{2,k}^b + \boldsymbol{\pi}_M(\mathbf{I}-R)^{-1}\widehat{v}_{2,k}^r, \quad (21)$$

$$\widehat{p}_C(k) = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{M-1})\widehat{v}_{3,k}^b + \boldsymbol{\pi}_M(\mathbf{I}-R)^{-1}\widehat{v}_{3,k}^r, \quad (22)$$

where the n^{th} element of the vector $\widehat{v}_{\ell,k}^b$ (resp., $\widehat{v}_{\ell,k}^r$) is 1 if the n^{th} state of the boundary (resp., level M) is in $\mathcal{D}_\ell(k)$, and otherwise is 0, $\ell = 1,2,3$. The vector $\widehat{p}_A$ represents the probabilities that the modeled task immediately enters the computation stage of the next iteration upon completion of the communication stage because the message to be received has already arrived. Since $T_S = 0$, the sender of this message must either be in stages 4, 5 or 6, and thus we have the following expression for $\widehat{p}_A$:

$$\widehat{p}_A(k) = \sum_{\ell=4}^{6} (\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_{M-1})\,\widehat{v}_{\ell,k}^b + \boldsymbol{\pi}_M(\mathbf{I}-R)^{-1}\,\widehat{v}_{\ell,k}^r. \quad (23)$$

Analogously, the initial probability vectors $\underline{\beta}_k'$, $\underline{\eta}_k'$ and $\underline{\zeta}_k'$ for the stage 4, 5 and 6 phase-type distributions can be written as

$$\underline{\beta}_k'(u) = \frac{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_1(k,u),\, y \in \mathcal{D}_1(k,u)} \pi(z)\,\mathbf{P}[h(z),h(y)]}{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_1(k),\, y \in \mathcal{D}_1(k)} \pi(z)\,\mathbf{P}[h(z),h(y)]}, \quad (24)$$

$$\underline{\eta}_k'(u) = \frac{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_2(k,u),\, y \in \mathcal{D}_2(k,u)} \pi(z)\,\mathbf{P}[h(z),h(y)]}{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_2(k),\, y \in \mathcal{D}_2(k)} \pi(z)\,\mathbf{P}[h(z),h(y)]}, \quad (25)$$

$$\underline{\zeta}_k'(u) = \frac{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_3(k,u),\, y \in \mathcal{D}_3(k,u)} \pi(z)\,\mathbf{P}[h(z),h(y)]}{\displaystyle\sum_{z \in \widehat{\mathcal{D}}_3(k),\, y \in \mathcal{D}_3(k)} \pi(z)\,\mathbf{P}[h(z),h(y)]}, \quad (26)$$

respectively. These expressions can be easily represented in closed-form with the use of binary vectors in the same manner as above for the other measures.

Finally, the phase-type distributions for stage 5 represent the times that the sender (of the message being waited for by the receiving task being modeled) spends in the I/O stage up until entering the send stage and gaining access to the processor. We construct these distributions from the decomposed model solution by examining the time until absorption in a new Markov process $\{Y'(t)\,;\, t \in \mathbb{R}_+\}$ obtained from the original process $\{Y(t)\,;\, t \in \mathbb{R}_+\}$ as follows. All of the states in $\mathcal{D}_3(k)$ are made to be absorbing states. We then make all of the remaining (reachable) states transient with the initial probability vector for every level i given by $\underline{\eta}_k'\boldsymbol{\pi}_i\mathbf{e}$, $1 \leq k \leq \min\{i, M\}$. The distributions $\text{PH}(\underline{\eta}_k', \mathcal{S}_k^{I'})$ then can be obtained either directly from the process $\{Y'(t)\,;\, t \in \mathbb{R}_+\}$ constructed in this fashion or by fitting a phase-type distribution to match the first several moments of the time until absorption in this process $\{Y'(t)\,;\, t \in \mathbb{R}_+\}$, using any of the best known methods for doing so; e.g., see [3, 8, 10, 17] and the references cited therein.

Our approach to handling general instances of the stochastic dynamic coscheduling model depends in part upon the specific value of T_S for the system environment of interest. When T_S is relatively small, we simply expand our approach above to capture the cases in which a series of up to T_S additional senders are waiting for messages from other processors by repeating T_S times the execution stages 4, 5 and 6. For example, the ℓ^{th} set of waiting stages, consisting of execution stages $4+3\ell$, $5+3\ell$ and $6+3\ell$, are entered according to a set of rules analogous to those governing transitions from stage 3 to stages 4, 5 and 6 where each transition rule is multiplied by the probability $(1-\widehat{p}_A(\overline{j}^R(1)))^\ell$, which represents the probability that the sender of the message to be received by the task being modeled is itself waiting on a series of ℓ processors to send their corresponding messages, $\ell = 0,\ldots,T_S$. (Recall that $\overline{j}^R(1)$ is the index of the executing task.) A set of transition rules analogous to those governing transitions from stage 6 to stage 1 are also appropriately constructed for each of these sets of waiting stages to make

the transition to the next set of waiting stages in the series. On the other hand, when T_S is relatively large, then we can simply use the execution stages 4, 5 and 6 of our original approach together with a form of the geometric distribution. In particular, upon completing stage 6, the system returns to stages 4, 5 and 6 with probability $(1-\widehat{p}_A(\overline{j}^R(1)))$ (following appropriately modified versions of the set of rules governing transitions from stage 3 to stages 4, 5 and 6) and otherwise enters stage 1 according to appropriately modified versions of the set of rules governing transitions from stage 6 to stage 1. Of course, both approaches can be used in combination. Finally, the sending and receiving of multiple messages is accommodated as follows. The sending of multiple messages is simply incorporated in the corresponding phase-type distribution(s) of the model. Our approach is extended to handle the case of receiving messages from multiple tasks by replacing the expressions and arguments provided above with versions of these expressions and arguments based on the appropriate order statistics. As previously noted, in the case of all-to-all communication, we derive an expression for the distribution of time that the task waits for messages from all other tasks and use a phase-type distribution to estimate this waiting time distribution.

Let $\kappa = (\widehat{p}_A, \widehat{p}_B, \widehat{p}_I, \widehat{p}_C, \underline{\beta}', \underline{\eta}', \underline{\varsigma}')$. Note that equations (20) – (26), which determine the value of κ, are expressed in terms of the decomposed model solution. Hence, we use a fixed-point iteration to solve the stochastic process as follows. Initial values are chosen for κ and the components of the stationary probability vector $\boldsymbol{\pi}$ are obtained using our matrix-analytic analysis. This solution yields new values for κ via the above equations and the model is solved again with these new values. This iterative procedure continues until the differences between the values of an iteration and those of the previous iteration are arbitrarily small. Numerous experiments were performed with this fixed-point iteration. We note that the fixed-point iteration always converged quite rapidly, and that in all of the cases tested, changing the initial values had no effect on the calculated fixed-point, i.e., the model solution was insensitive to the initial values chosen for κ.

4.4 Performance Measures

Various performance measures of interest can be obtained from the components of the stationary vector $\boldsymbol{\pi}$. In particular, the mean number of parallel jobs in the partition can be expressed as

$$\overline{\mathcal{N}} = \sum_{k=1}^{M-1} k\boldsymbol{\pi}_k\mathbf{e} + M\boldsymbol{\pi}_M(\mathbf{I}-R)^{-1}\mathbf{e} + \boldsymbol{\pi}_M R(\mathbf{I}-R)^{-2}\mathbf{e}. \quad (27)$$

Using Little's law [14] and (27), the mean response time of a parallel job in the partition then can be calculated as $\overline{\mathcal{T}} = \lambda^{-1}\overline{\mathcal{N}}$.

Another set of performance measures of interest is the long-run proportion of time that a processor spends performing computation, I/O, communication, context switching, and some form of waiting. These measures can be expressed as

$$\widetilde{p}^x \;=\; (\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_{M-1})\, v_b^x \;+\; \boldsymbol{\pi}_M(\mathbf{I}-R)^{-1} v_r^x, \qquad (28)$$

where $x \in \{B, I, C, O, W\}$ such that B: computation; I: I/O; C: communication; O: context switching; W: waiting. Here we use the notation that the n^{th} position of the vector v_b^x (resp., v_r^x) contains a 1 if the corresponding state of the boundary (resp., level M) represents when the processor is performing operations of type x, and otherwise contains a 0.

5. RESULTS

We now apply the mathematical analysis of Section 4 to study the performance characteristics of the dynamic coscheduling strategies defined in Section 2 within the context of the parallel system and workload models of Section 3. A large number of numerical experiments have been conducted, and a representative set of results are presented herein. However, due to space limitations, we omit those results that explore the impact of certain parameters on the performance of the dynamic coscheduling strategies. This includes context-switch costs, job duration, and resource contention. We refer the interested reader to [22] for these additional results.

Parameter	Value(s)
M	3
δ^{-1} (Context Switching Cost) I (Interrupt Cost)	200 us 50 us
τ^{-1} (Quantum length)	20 ms
γ^{-1}	185.48 us
γ'	γ
Priority Changes	3 us
Check for a Message	2 us
PB interval	1 ms
Fixed Maximum Spin time (SB)	200 us
Message Size	4096 bytes
Communication Pattern	Nearest Neighbor

Table 1: Default Parameters Used in the Results

Some of the default parameter values used for our base-case results are provided in Table 1, and any deviations from this base case will be explicitly noted. Many of these values have been drawn from actual system platforms.

5.1 Validation

Given that our analysis derived in Section 4.1 considers an approximation of the (exact) stochastic process $\{X(t)\,;\, t \in \mathbb{R}_+\}$ defined at the beginning of Section 4, we first must validate the results of our analysis against detailed simulations to demonstrate the accuracy of our approach. Figure 1 presents the relative errors of our models and analysis as a function of the arrival rate λ for a 32-processor system. A representative sample of the results are provided for all four dynamic coscheduling strategies under computation-intensive, communication-intensive and I/O-intensive workloads. In each case, our model is in excellent agreement with detailed simulations of a relatively small-scale parallel system, often being within 5% and always less than 10%. Recall that the accuracy of our approach increases with the size of the system and that our primary interests are in large-scale parallel systems consisting of many computing nodes. With the complex dynamic interactions among the different aspects of the parallel computing environment and the scheduling strategy, these results provide considerable evidence of the benefits of our approach to investigate the fundamental design and performance problems at hand.

5.2 Impact of Load

Figure 2 provides a representative sample of the results for investigating the effects of the arrival rate on the mean job response time under three types of workloads. In general, the differences between the schemes are less significant with lower communication in the workload, as is to be expected. Even in the CPU and I/O intensive workloads, the LS and DCS mechanisms saturate earlier than the other two, and this becomes more apparent in the communication intensive workload. Overall, we can say that LS and DCS are not very desirable because they tend to saturate at smaller values of λ than SB and PB, and the details on the effect of the workload characteristics on the relative performance of these schemes are presented next. Note that, until now, no one has been able to study these mechanisms for dynamic job arrivals with such a broad spectrum of arrival rates.

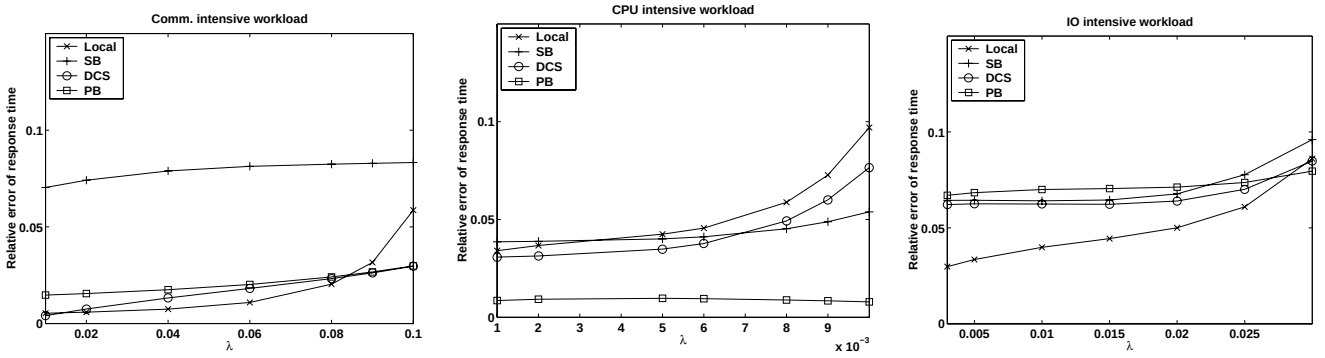
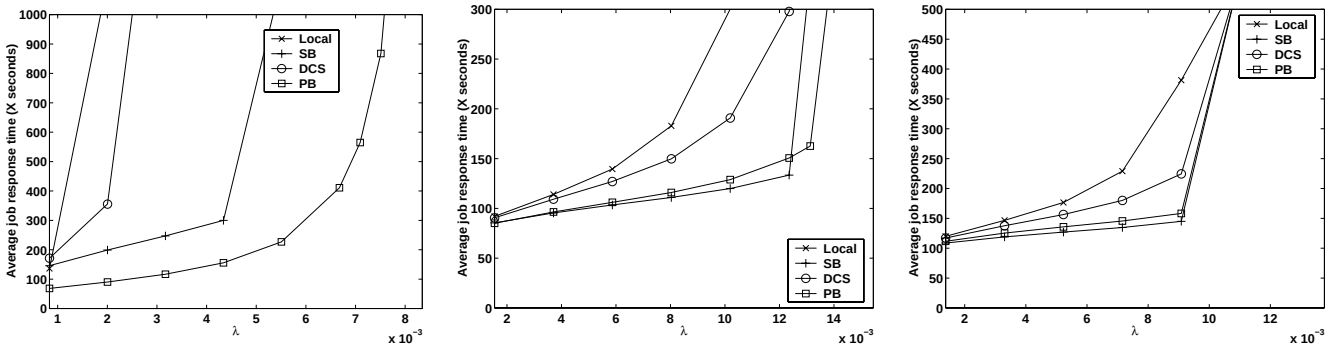


Figure 1: Validation Results.



(a) Comm. Intensive workload ($\mu^{-1} = 0.14$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 0.024$ms)

(b) CPU Intensive workload ($\mu^{-1} = 36$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 2$ms)

(c) I/O Intensive workload ($\mu^{-1} = 10$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 5$ms)

Figure 2: Impact of Load on Response Time, $\frac{1}{P_{N_A}}(\mu^{-1} + \gamma^{-1} + \nu^{-1}) = 38.19$secs.

5.3 Impact of Workload Characteristics

Since communication intensive and I/O intensive workloads are most interesting, we focus on these two workloads in Figures 3 and 4. These figures reconfirm the results in the previous subsection for varying system load, with LS and DCS performing poorly. For the communication intensive workload, PB does better than SB in terms of both throughput and response time. The benefits of PB are accentuated as the communication intensity increases. Clearly, we can see that the fraction of the time spent in communication increases with the intensity in Figures 3(e) – (f), but the increase is more gradual for PB than for SB. This can be explained based on the behavior in (b) and (c), where we can see MPL3-idle is significantly higher in SB than in PB. At such high communication intensities, blocking to relinquish the processor incurs context-switch and interrupt overheads with little benefits since there is no other work to do (everyone is blocked). PB which does not switch under those conditions does not experience the context-switch costs, and thus yields better performance.

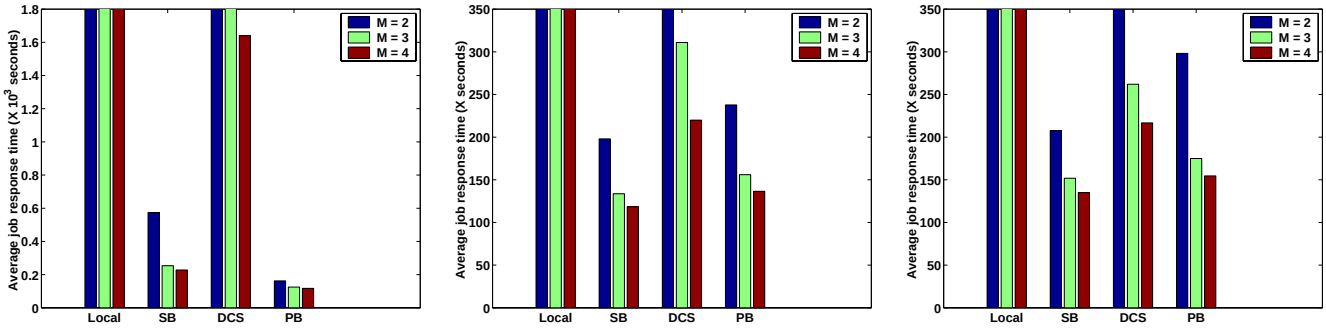
For the I/O intensive workload, the differences among the schemes are less noticeable since the performance is dictated more by the I/O in the application than by the schemes themselves (which do not behave differently for I/O). In fact, we expect all the response time curves in Figure 4(d) to converge at large I/O intensities. The reason why throughput increases and response time decreases for this case (in contrast to the communication intensive figures), is because I/O can be performed concurrently by tasks at a node (while computation/communication cannot) in this instance of our model. The profile graphs show similar behavior to that of the communication intensive workload. At this point, we can also explain why SB does marginally better than PB for I/O intensive (and, incidentally, CPU intensive) workloads. With large computation or I/O fractions, the skewness of the work to be done among the tasks of an application also increases. This can cause tasks to spin more than the message latencies in PB, while SB can limit the effect of such skewness. For the communication intensive workload, this skewness gets smaller, and PB realizes the full benefits of spinning.

5.4 Impact of Maximum MPL (M)

Figure 5 considers the impact of the maximum MPL (M) on the dynamic coscheduling strategies. Increasing M can help to reduce the processor idling (during I/O or when waiting for a message) by providing more opportunities to switch to another task and execute useful work. As a result, performance improvements due to higher values of M are more noticeable in the I/O intensive workload, or in schemes where the waiting time is high (DCS). This is also the reason why SB needs a higher value of M to become as competitive as PB for the communication intensive workload. While there are improvements in response times with increasing M, it should be noted that too high a value for M is not appropriate due to practical resource limitations and/or because of high swap costs.

5.5 Optimum Quantum length (τ^{-1})

For any scheme, a small time quantum increases the impact of context-switch overheads, while at the same time a small quantum can mitigate the effects of scheduling skews across processors (to avoid large wait times). These two contrasting factors play a role in

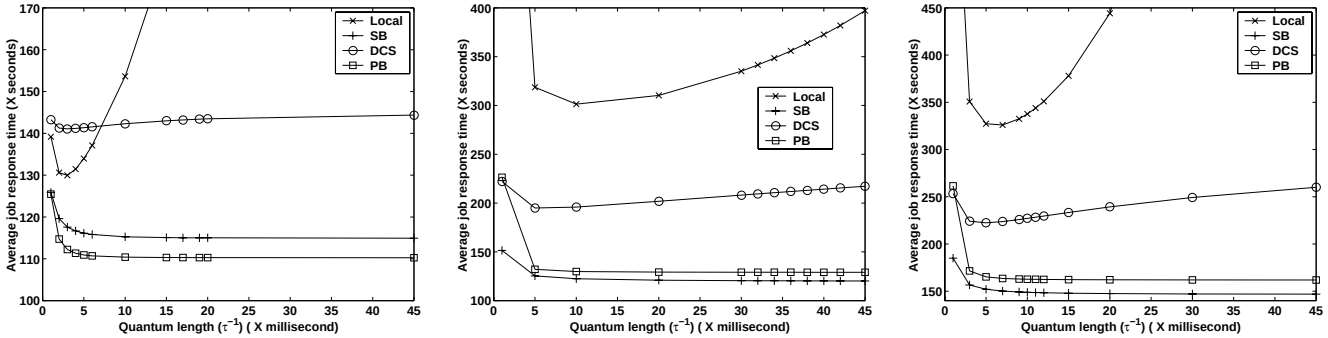


(a) Comm. Intensive workload ($\mu^{-1} = 0.14$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 0.024$ms, $\lambda = 0.003$)

(b) CPU Intensive workload ($\mu^{-1} = 36$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 2$ms, $\lambda = 0.0123$)

(c) I/O Intensive workload ($\mu^{-1} = 10$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 5$ms, $\lambda = 0.01$)

Figure 5: Impact of M on Response Time, $\frac{1}{P_{N_A}}(\mu^{-1} + \gamma^{-1} + \nu^{-1}) = 38.19$secs.



(a) Comm. Intensive workload ($\mu^{-1} = 2$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 0.34$ms, $\lambda = 0.004$)

(b) CPU Intensive workload ($\mu^{-1} = 36$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 2$ms, $\lambda = 0.0103$)

(c) I/O Intensive workload ($\mu^{-1} = 10$ms, $\gamma^{-1} = 0.19$ms, $\nu^{-1} = 5$ms, $\lambda = 0.0095$)

Figure 6: Impact of Quantum Length (τ^{-1}) on Response Time, $\frac{1}{P_{N_A}}(\mu^{-1} + \gamma^{-1} + \nu^{-1}) = 38.19$secs.

determining a good operating point for the time quantum. Figure 6 captures the effect of these factors for the four schemes on the three workload classes. In general, for the LS and DCS schemes (which are more susceptible to scheduling skews as shown in earlier results), the second factor that can mitigate scheduling skews is more important, causing the good operating points for LS and DCS in Figure 6 to be more to the left than those for SB or PB. In fact, SB and PB would prefer a long time quantum, since they do not really rely on the native operating system scheduler and perform the task switches whenever needed. As for the effect of the workload itself, CPU and I/O intensive workloads should prefer longer time quanta (because the first factor concerning context-switch overheads are more important) than the communication intensive workload.

5.6 Optimum Spin Time

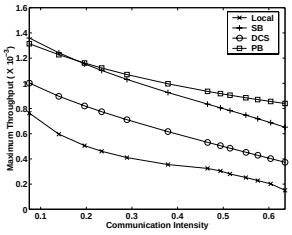
For SB, the choice of the spin time is a crucial issue. Previous studies have not explored the full impact of spin time across a broad spectrum of workloads. Figure 7(a) shows the effect of the spin time on four different workloads. The curves are normalized with respect to the performance for the default spin time. A small spin time is preferred if either the message will arrive in a very short time, or if it will take a very long time (so that we do not waste too much time spinning). A slightly larger spin time will be preferred in the other situations. The results in this figure capture this effect for the four workloads, showing that the ideal spin time (giving the lowest response time) is very sensitive to the workload. This makes the selection of a good spin time on a real system very difficult, but it further highlights the importance of the use of our models and analysis for the proper setting of this parameter in any system of interest. It also should be noted that in the results presented in previous sections, the chosen spin times are reasonably close to their ideal values (no more than 5–10% off).

5.7 Optimum PB Frequency

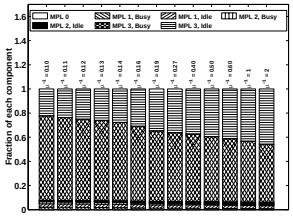
One of the important design considerations for PB is selecting the frequency of the kernel activity. Across the spectrum of workloads ranging from very high to very low communication intensities, we find that the ideal frequency of invocation lies between 0.3ms to 1ms in Figure 7(b) (the lines are normalized with respect to a 1ms frequency). These results suggest that an invocation frequency of between 0.5 to 1ms would provide good performance across the entire workload spectrum for PB.

6. CONCLUDING REMARKS

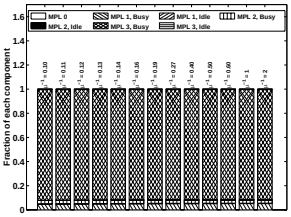
The complex interactions among the different system components and the numerous workload parameters have made it difficult until now to investigate the advantages and disadvantages of dynamic coscheduling mechanisms in great detail. All previous studies have been limited by the environments under consideration, whether it be the underlying hardware, operating system, system



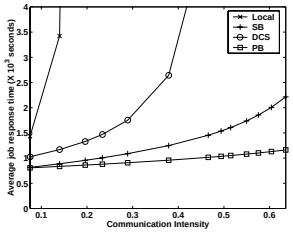
(a) Maximum Throughput



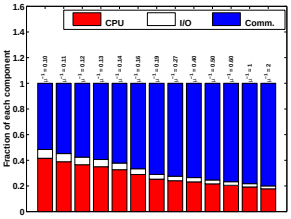
(b) System Profile at Maximum Utilization for SB (time unit is millisecond)



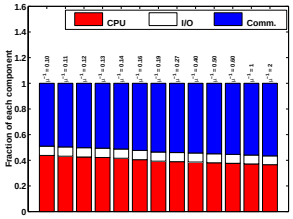
(c) System Profile at Maximum Utilization for PB (time unit is millisecond)



(d) Average Response Time ($\lambda = 0.006$)

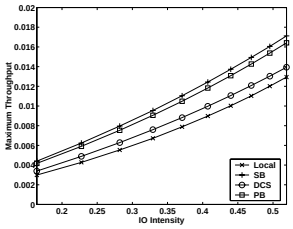


(e) Application Profile for SB (time unit is millisecond, $\lambda = 0.006$)

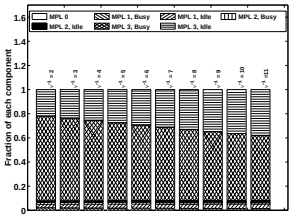


(f) Application Profile for PB (time unit is millisecond, $\lambda = 0.006$)

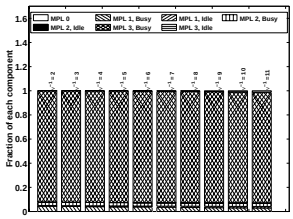
Figure 3: Impact of Comm. Intensity, $\nu^{-1}/\mu^{-1} = 0.17$, $\gamma^{-1} = 0.19$ms, $\frac{1}{P_{N_A}}(\mu^{-1} + \gamma^{-1} + \nu^{-1}) = 38.19$secs.



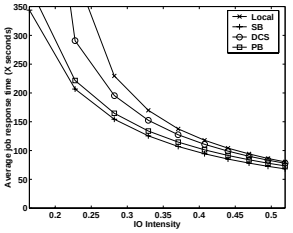
(a) Maximum Throughput



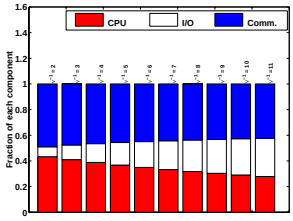
(b) System Profile at Maximum Utilization for SB (time unit is millisecond)



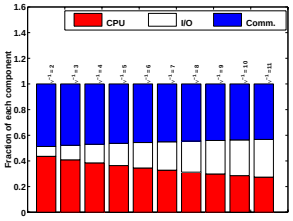
(c) System Profile at Maximum Utilization for PB (time unit is millisecond.)



(d) Average Response Time ($\lambda = 0.008$)



(e) Application Profile for SB (time unit is millisecond, $\lambda = 0.008$)



(f) Application Profile for PB (time unit is millisecond, $\lambda = 0.008$)

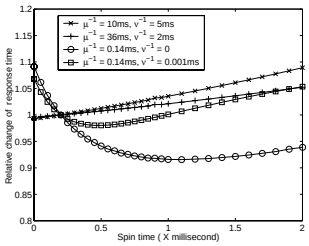
Figure 4: Impact of I/O Intensity, $\mu^{-1} = 10$ms, $\gamma^{-1} = 0.19$ms, $\frac{1}{P_{N_A}}(\mu^{-1} + \gamma^{-1} + \nu^{-1}) = 38.19$secs.

size or workload. The lack of a unifying framework for studying the benefits and limitations of dynamic coscheduling under different system conditions, workload parameters and system size has been the limiting factor in these previous studies. This paper has addressed this critical void in scheduling for parallel systems by developing and validating an accurate analytical model for studying the design and performance spaces of dynamic coscheduling mechanisms across a wide range of system and workload parameters.

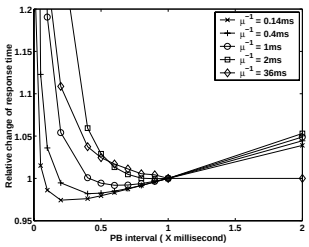
Specifically, we formulated a general mathematical model of various dynamic coscheduling strategies within a unified framework that addresses the aforementioned limitations of previous studies. We derived a matrix-analytic analysis of these scheduling models based on a stochastic decomposition, which leads to a vastly reduced state-space representation, and a fixed-point iteration, which is used to obtain the final solution. In addition to mean job response time and maximum system throughput measures, the detailed probabilistic measures from our analysis were used to help explain the overall results and to gain fundamental insights about various aspects of the different dynamic coscheduling approaches. Moreover, numerical results from our analysis were shown to be in excellent agreement with those from detailed simulations of even relatively small-scale systems, often within 5% and always less than 10%.

Numerical results from our analysis show that it is not advisable to allow the native operating systems at each node to switch between tasks at their disposition. Some amount of coordination is definitely needed. Of the three previously proposed dynamic coscheduling mechanisms, DCS does not fare as well as the others across a broad spectrum of workload and system parameters. SB and PB have their relative merits, with the latter faring better whenever the workload is more communication intensive. PB is also preferable whenever nodes are not operating at the full multiprogramming level. Our model and analysis can be used as a design tool to fine tune the parameters for these mechanisms (spin time in SB and periodic frequency in PB) to derive good operating points. With SB, the choice of the spin time is important for determining performance, while a frequency of once every 0.5–1ms provides good performance for PB across the entire workload spectrum considered. Both of these mechanisms are relatively immune to the native operating system switching activity, by taking over whenever the communication events call for coscheduling and becoming less intrusive otherwise. These mechanisms are good choices regardless of whether the system is subject to short running interactive jobs or long running parallel applications. It should be noted that the results presented here for SB are with a fixed spin time, though the model itself allows adaptive tuning of this value (which more recently has been shown to be a better alternative). Planned future research includes using our model and analysis to investigate various adaptive spin time approaches.

Some of the newer insights and contributions that this study has



(a) Spin Time for SB ($\lambda = 0.003$)



(b) PB Interval ($\lambda = 0.003$, $\nu^{-1}/\mu^{-1} = 0.17$)

Figure 7: Design Choices for SB and PB, $\frac{1}{P_{N_A}}(\mu^{-1} + \gamma^{-1} + \nu^{-1}) = 38.19$secs.

provided, which have not been possible in earlier simulations or experimental evaluations, is to be able to answer several important issues about the the optimal frequency for invoking the periodic boost mechanism, the optimal frequency of context switching (time quantum length), a direct way of calculating the optimal fixed spin time for SB without running costly simulations, and the impact of workload characteristics on these issues. Our model and analysis serve as a powerful tool for exploring these issues at little cost.

There are several interesting directions for future work. Along the scheduling front, it would be interesting to find out how best a schedule one can ever hope to achieve with a given load, to shed light on future research in this area. Further, examining differential services to different job types, and the ability of a scheduler to provide guaranteed (soft or hard) service to parallel jobs in the context of a dynamically coscheduled environment are part of our future work. Along the theoretical front, it would be interesting to extend our models and analysis of this paper to solve these problems, as well as to investigate the problem of setting various dynamic coscheduling parameters as mathematical optimization problems, based in part on the results, models, analysis and/or insights presented in this paper.

7. REFERENCES

[1] C. Anglano. A comparative evaluation of implicit coscheduling strategies for networks of workstations. *Proceedings of International Symposium on High Performance Distributed Computing*, 2000.

[2] A.C. Arpaci-Dusseau, D.E. Culler, A.M. Mainwaring. Scheduling with implicit information in distributed systems. *Proceedings of ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, 1998.

[3] S. Asmussen. Phase-type distributions and related point processes: Fitting and recent advances. *Matrix-Analytic Methods in Stochastic Models*, S.R. Chakravarthy & A.S. Alfa (eds.), 137–149, 1997.

[4] N.J. Boden et al. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[5] A.C. Dusseau, R.H. Arpaci, D.E. Culler. Effective distributed scheduling of parallel workloads. *Proceedings of ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, 25–36, 1996.

[6] D.G. Feitelson. A survey of scheduling in multiprogrammed parallel systems, Research Report RC 19790(87657), IBM Research Division, 1994.

[7] D.G. Feitelson, B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson & L. Rudolph (eds.), 337–360, 1995. Springer-Verlag LNCS Vol. 949.

[8] A. Feldmann, W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation*, 31:245–279, 1998.

[9] D. Gaver, P. Jacobs, G. Latouche. Finite birth-and-death models in randomly changing environments. *Advances in Applied Probability*, 16:715–731, 1984.

[10] A. Horvath, M. Telek. Approximating heavy tailed behaviour with phase type distributions. *Advances in Algorithmic Methods for Stochastic Models*, G. Latouche & P. Taylor (eds.), 191–214, 2000.

[11] S.G. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson & L. Rudolph (eds.), 27–40, 1996. Springer-Verlag LNCS Vol. 1162.

[12] N. Islam, A. Prodromidis, M.S. Squillante. Dynamic partitioning in different distributed-memory environments. *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson & L. Rudolph (eds.), 244–270, 1996. Springer-Verlag LNCS Vol. 1162.

[13] G. Latouche, V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[14] J.D.C. Little. A proof of the queuing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[15] S. Nagar, A. Banerjee, A. Sivasubramaniam, C.R. Das. A closer look at coscheduling approaches for a network of work stations. *Proceedings of ACM Symposium on Parallel Algorithms & Architectures*, 96–105, 1999.

[16] M.F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. The Johns Hopkins University Press, 1981.

[17] B.F. Nielsen. Modelling long-range dependent and heavy-tailed phenomena by matrix analytic methods. *Advances in Algorithmic Methods for Stochastic Models*, G. Latouche & P. Taylor (eds.), 265–278, 2000.

[18] J.K. Ousterhout. Scheduling techniques for concurrent systems. *Proceedings of International Conference on Distributed Computing Systems*, 22–30, 1982.

[19] P.G. Sobalvarro. *Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors*. PhD thesis, Dept. of Elec. Eng. & Comp. Sci., MIT, Cambridge, MA, 1997.

[20] M.S. Squillante. A matrix-analytic approach to a general class of G/G/c queues. Research Report, IBM Research Division, 1996.

[21] M.S. Squillante, F. Wang, M. Papaefthymiou. Stochastic analysis of gang scheduling in parallel and distributed systems. *Performance Evaluation*, 27&28:273–296, 1996.

[22] M.S. Squillante, Y. Zhang, A. Sivasubramaniam, N. Gautam, H. Franke, J. Moreira. Analytic modeling and analysis of dynamic coscheduling for a wide spectrum of parallel and distributed environments. Research Report, IBM Research Division, 2000.

[23] Specification for the Virtual Interface Architecture. `http://www.viarch.org`.

[24] Y. Zhang, A. Sivasubramaniam, J. Moreira, H. Franke. A simulation-based study of scheduling mechanisms for a dynamic cluster environment. *Proc. of ACM International Conference on Supercomputing*, 100–109, 2000.