Optimal Remapping in Dynamic Bulk Synchronous Computations via a Stochastic Control Approach

G. George Yin, *Fellow*, *IEEE*, Cheng-Zhong Xu, *Senior Member*, *IEEE*, and Le Yi Wang, *Senior Member*, *IEEE*

Abstract—A bulk synchronous computation proceeds in phases that are separated by barrier synchronization. For dynamic bulk synchronous computations that exhibit varying phase-wise computational requirements, remapping at runtime is an effective approach to ensure parallel efficiency. This paper introduces a novel remapping strategy for computations whose workload changes can be modeled as a Markov chain. The use of Markovian model allows us to treat statistical dependence and more complex structure than the usual independent identically distributed random variable assumptions. Our models are quite general and we do not need to impose conditions on the dynamics of the underlying process other than the transition probability matrix. It is shown that optimal remapping can be formulated as a binary decision process: remap or not at a given synchronizing instant. The optimal strategy is then developed for long lasted computations by employing optimal stopping rules in a stochastic control framework. The existence of optimal controls is established. Necessary and sufficient conditions for the optimality are obtained. Furthermore, a policy iteration algorithm is devised to reduce computational complexity and enhance fast convergence to the desired optimal control.

Index Terms—Mapping, remapping, optimal stopping, bulk synchronous computation, stochastic control.

1 Introduction

One of the most important issues in parallel computing is assignment, or mapping, of the workload to processing nodes of a parallel computer. The primary performance goals of mapping are to balance the workload among the processors, to minimize the interprocessor communication, and to reduce the runtime overhead of managing the assignment [6], [28]. This can be accomplished statically or dynamically, depending upon the nature and predictability of the computation. Static assignment works for applications that have static and predictable computation and communication characteristics. For applications that have unpredictable characteristics, dynamic scheduling must be used at runtime so as to adapt to the change of the workload. A special form of dynamic scheduling is *dynamic remapping*, the subject of this paper, which redistributes the workload among processing nodes during the execution time.

In general, achieving the optimal performance goals of mapping is computationally intractable due to numerical complexities. However, a bulk synchronous computation lends itself to a fairly structured approach to mapping. Its processes proceed in phases that are separated by global synchronization. During each phase, they perform calculations independently and then communicate new results with their data-dependent peers. Due to the need synchronization between steps, the duration or execution time of a step is determined by the most heavily loaded processor, which in turn determines the total execution time of the application.

Bulk synchronous computations may exhibit varying phase-wise computational requirements as the computation proceeds. This can occur in applications where the behavior of the physical objects being modeled changes with time. For example, a molecular dynamics program simulates the dynamic interactions among all atoms in a system of interest for a period of time. For each time step, the simulation calculates the forces between atoms, the energy of the whole structure, and the movements of atoms. Since atoms tend to move around, the amount of workloads with different parts of the system will change from one step to another as the atoms change their spatial positions. Dynamic computations can also appear in solution-adaptive problems. For example, a computational fluid dynamics code calculates the velocity and pressure of grid points for the purpose of deriving their structural and dynamic properties. In simulations that use adaptive gridding to adjust the scale of resolution as the simulation progresses, computational workloads associated with different parts of a grid may change from phase to phase. Dynamic bulk synchronous computations can also appear in multiprogrammed distributed systems. Since the resources available to a job may change as other jobs join and leave, even a static bulk synchronous computation may exhibit nondeterministic computational characteristics.

Due to its popularity in many application areas, dynamic bulk synchronous models have long been the subject of

• *G.G. Yin is with the Department of Mathematics, Wayne State University, Detroit, MI 48202. E-mail: gyin@math.wayne.edu.*
• *C.-Z. Xu and L.Y. Wang are with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202. E-mail: czxu@wayne.edu, lywang@ece.eng.wayne.edu.*

Manuscript received 28 Sept. 2000; revised 29 June 2001; accepted 5 Mar. 2002.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 112919.

extensive research. Marinescu and Rice quantitatively analyzed the effects of the load imbalance of a bulk synchronous computation, assuming that the execution times of processes in an iteration are independent and identically distributed random variables [15]. Agrawal and Chakradhar modeled VLSI logic simulations as the dynamic bulk synchronous model [1]. By assuming the workload of a processor within a time step to be a binomial random variable, they derived an accurate estimate of the parallel simulation time. Madala and Sinclair derived asymptotic approximates and upper bounds for the average execution time of a bulk synchronous computation [14]. Similar results were obtained by Peterson and Chamberlain when they modeled a general class of discrete event simulation as the dynamic bulk synchronous model [23].

There were also studies on dynamic remapping with an objective of lessening the penalty due to synchronization and load imbalances [4], [7], [10], [11], [17], [22], [24], [25], [26], [29]. Dynamic remapping strategies were demonstrated effective when they were applied to solution-adaptive CFD [11], [26], multistage image understanding systems [4], and Monte Carlo dynamical simulations [10], [25]. Dynamic remapping was shown to be important for applications that have no inherent synchronization requirements as well [18]. Nicol and Ciardo experimented with dynamic remapping at artificial synchronization points created with real-time clocks. There were also time-efficient distributed remapping strategies that redecompose the problem domain in parallel based on the previous distributions [29]. They are not sufficient to produce the highest quality partitions because of the lack of global knowledge about the problem domain. The loss of their quality could be compensated for by its smaller runtime overhead in periodic remapping.

Since dynamic remapping incurs nonnegligible runtime overhead, a more critical issue is when to remap so that its performance gain will not be outweighed by its overhead. It can be either specified by the programmer in the form of a directive or determined automatically by the compiler.

For applications with gradually varying resource demands, Nicol and Saltz [20] proposed a simple provably good invocation policy, Stop-At-Rise (SAR). Moon and Saltz [16] applied the SAR invocation policy, coupled with an elegant chain-structured partitioner and a recursive coordinate bisection (RCB) partitioner, to three-dimensional direct Monte Carlo simulation methods. An alternative to the SAR is periodic remapping. Xu and Lau [27], [28] and Nicol and Ciardo [18] experimented with *periodic remapping* policies against the remapping periodicity in parallel skeleton of images, WaTor simulation, and discrete state-space generations. Moon and Saltz [16] compared the periodic policy with the SAR strategy in their Monte Carlo simulations and found that the best fixed interval policy was able to deliver comparable performance to the SAR through the periodic policy. There are also application-specific remapping policies that invocate remapping in response to abrupt changes of the computations (e.g., grid refinement in the CFD code) or changes of the system resources [5], [4], [19].

The literature is littered with heuristic invocation policies. There were few rigorous analyses of the impact of the invocation policies. In [9], Fong et al. studied the periodic remapping policy, focusing on the impact of remapping frequencies. They formulated the problem as sequential stochastic optimization and derived optimal remapping frequencies for additive and multiplicative workload change models with known and unknown probabilistic distributions. In general, Nicol and Saltz modeled the decision process of "when to remap" as a stochastic dynamic programming problem [20]. They defined a cost function with respect to the remapping overhead and potential gains over a finite number of steps. They derived optimal remapping controls by recursively updating the cost function according to the Bellman's optimality principle [2]. Due to the high complexities of dynamic programming, their analytical approach is limited to small scale systems, small workload state spaces, and a small number of computation steps.

It is noticed that the effects of a remapping operation normally will not last for too many steps in dynamic bulk synchronous computations. Any uniform distribution generated by a remapping operation will soon become imbalanced again as the computation proceeds and the computation will evolve into a similar state to the one prior to remapping. We consider the remapping problem over computation steps $[0, N]$. For sufficiently large N, it can be approximated by that of $[0, \infty)$.

With this regard, we formulate the remapping problem for long lasted bulk synchronous computations as an optimal stopping problem in stochastic control theory. It deals with a free-time problem, namely, how to control (remapping) until a desired target set (workload balance) is reached. By converting the optimal stopping problem to an equivalent binary (i.e., remap or not) decision over an infinity horizon, we derive conditions for the existence of optimal strategies and present necessary and sufficient conditions for the optimality. Searches for the optimal controls are reduced to finding fixed points of a policy iteration equation. This iterative process converges at a geometric rate.

The analytical approach in [20] was based on an assumption of multiple independent Markov chain workload evolution models. That is, the workload of a process changes in a way that its new value depends only on its own current load. This assumption does not hold in many applications. For example, in molecular dynamics applications, the workload evolution of a process is explicitly dependent on its own load and on the loads of other processes because the atoms would move across partitions.

Another main contribution of this paper is that our remapping formulation eliminates the assumption of independent workload evolutions between processes. By modeling the workload evolutions of different processes collectively as a single vector-valued Markov chain, the workload of a process can depend on its own load as well as that of others.

The rest of the paper is arranged as follows: Section 2 characterizes the bulk synchronous computations as a Markov chain workload evolution model. Section 3 establishes the connection of remapping with that of optimal

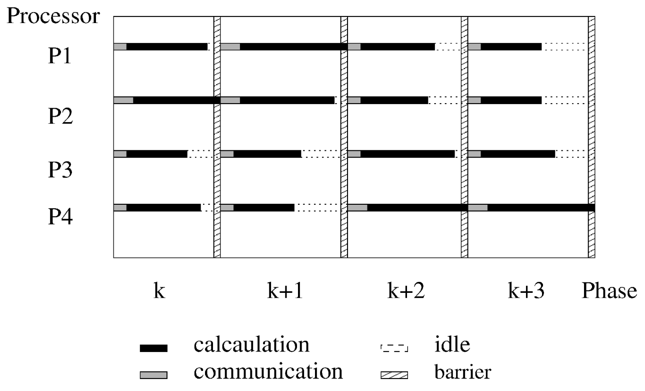

Fig. 1. An illustration of dynamic bulk synchronous computations.

stopping in stochastic control. The optimal remapping strategy is then treated in Section 4. We illustrate the remapping strategy through a concrete example in Section 5. Section 6 concludes the paper with remarks on future work.

2 Bulk Synchronous Computations

Consider a bulk synchronous computation that is comprised of r processes labeled from 1 to r. Assume the processes are running in parallel in a multiprogrammed distributed system with the same number of processing nodes. The computation proceeds in steps $k = 1, 2, \dots$, separated by barrier synchronization. The processes may exhibit step-wise computational requirements as the computation proceeds. Resources allocated to the computation can also vary with time as other jobs join and leave. Fig. 1 shows the computation model with four processes running on different processors. The horizontal scale corresponds to computation phases; the vertical lines represent barrier synchronization operations which are due to begin.

Molecular dynamics (MD) is a typical example of such bulk synchronous computations. It simulates the dynamic interactions between atoms in a system of interest step by step for microscopic insights into the structural and dynamical properties of materials. At each step, the simulation calculates the forces between atoms, the energy of the whole structure, and the movements of atoms. Fig. 2a shows the workload change of a molecular dynamic program running on a cluster of three dedicated 4-way SMP servers. By contrast, Fig. 2b plots the results on the cluster when there is one cpu-bound disturbance job running. Although the disturbance job was single-threaded and occupying one processor at a time, it led to severe load imbalance between the processors. The bulk synchronous computation slowed down significantly due to the barrier synchronization between phases.

Let w_k^i be the workload of process i at step k and c_k^i be the capacity of its corresponding processing node. The execution time of process i at step k, t_k^i is proportional to the workload w_k^i relative to the capacity c_k^i (or proportional to the scaled workload), that is, $t_k^i \sim w_k^i / c_k^i$. Since the duration of the kth step of the computation is determined by the maximum of t_k^i for $i = 1, 2, \dots, r$, the total elapsed time of the computation for N steps, denoted by T, is given by

$$T = \sum_{k=1}^{N} \max\{t_k^i, i = 1, 2, \dots, r\}. \tag{1}$$

As the computation proceeds, the workload w_k^i and the capacity c_k^i may evolve dynamically and randomly with step k. As a result, the execution time of each process t_k^i becomes nondeterministic. The objective of remapping is to redistribute the processes' scaled workload at runtime so as to reduce the duration of each step and minimize the elapsed time. Assuming the random processes $\{w_k^i\}$ and $\{c_k^i\}$, $r = 1, 2, \dots, r$, are independent, we refer to w_k^i as normalized workload relative to c_k^i henceforth. Since t_k^i is solely determined by w_k^i, we will use the terms of execution time and workload of processes interchangeably.

2.1 Remapping

The remapping problem involves how to remap and when to remap. The first issue is to balance the process workload at a minimum cost. The second issue is to determine the invocation conditions under which benefits of remapping would not be outweighed by its runtime cost.

Assume that the computation starts with an initial workload distribution w_1^i for $i = 1, \dots, r$ at time $k = 1$. Use a vector $w_k = (w_k^1, \dots, w_k^r)$ to denote the workload distribution at step k. The workload distribution w_k changes with time randomly, following the progressing of computation tasks, and can also be altered by remapping. Denote the workload distributions just before the remapping and right after the remapping at step k by vectors w_k^- and w_k^+, respectively, where $w_k^{\pm} = (w_k^{\pm,1}, \dots, w_k^{\pm,r})$. Assume the underlying computation suspends while remapping, that is, there is no workload generation or consumption during the procedure of remapping. A remapping operation then defines a map:

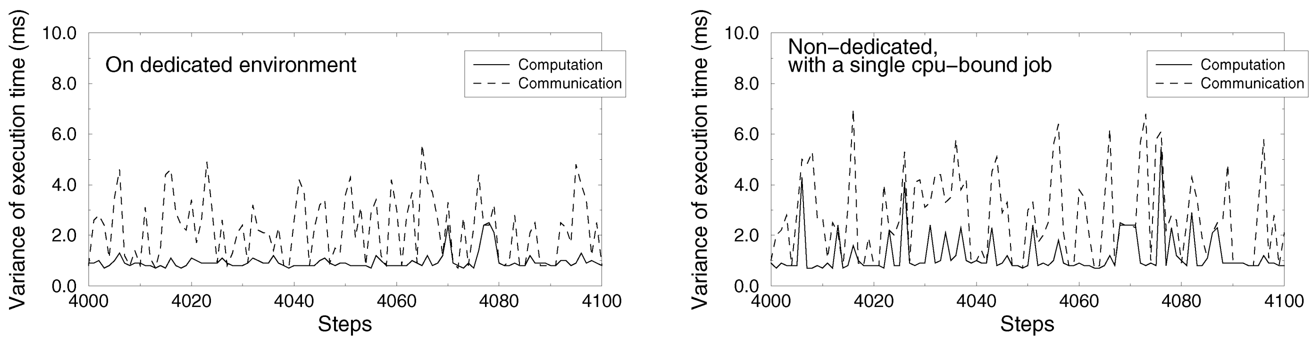

Fig. 2. Workload change on dedicated and multiprogrammed systems.

$$w_k^+ = \mathcal{R}(w_k^-) \text{ satisfying } \sum_{i=1}^{r} w_k^{+,i} = \sum_{i=1}^{r} w_k^{-,i}. \tag{2}$$

To develop a viable remapping strategy, we introduce a measure, $\phi(w)$, of workload imbalance among processors, normalized with their capacities. Generically, $\phi(w_k) \geq 0$ is a mapping from the scaled workload distribution $w_k = (w_k^1, \ldots, w_k^r)$ to a nonnegative real value. Let w_k^0 denote the average of the workload at step k and $\overline{w}_k$ be a uniform workload distribution. That is,

$$\begin{aligned} w_k^0 &= \frac{1}{r}\sum_{i=1}^{r} w_k^i \text{ and} \\ \overline{w}_k &= (w_k^0, \ldots, w_k^0). \end{aligned} \tag{3}$$

Then, the measure of load imbalance ϕ has the following properties:

$$\phi(w_k)\begin{cases} = 0 & \text{if } w_k = \overline{w}_k, \\ > 0, & \text{if } w_k \neq \overline{w}_k. \end{cases}$$

Among the many possible choices, the maximum imbalance,

$$\phi(w_k) = \|w_k - \overline{w}_k\|_\infty = \max_{i=1,\ldots,r} |w_k^i - w_k^0|, \tag{4}$$

and the average imbalance,

$$\phi(w_k) = \|w_k - \overline{w}_k\|_2 = \sqrt{\sum_{i=1}^{r} (w_k^i - w_k^0)^2}, \tag{5}$$

are used most often. The remapping reduces imbalance in that $\phi(w_k^+) \leq \phi(w_k^-)$. Thus, $\phi(w_k^-) - \phi(w_k^+)$ is a measure of imbalance reduction by remapping. After the remapping, the system starts anew, i.e., the initial condition is reset from w_k^- to w_k^+. Then the system is running with the new initial condition w_k^+. In particular, w_k^+ is reset to $\overline{w}$ when a total remapping is implemented. Since the actual workload w_k^+ after remapping may not be an integer, w_k^+ will be quantized, say, by rounding to its nearest integer, as the renewed initial state of workload after remapping.

Note that the cost of remapping generally depends on the desired imbalance reduction. When the total remapping becomes unaffordable, a partial remapping can be a choice for better tradeoff between the cost and remapping benefits. Optimal decisions for the best choice of partial remapping operations are beyond the scope of this paper. The reader is referred to [30] for an empirical study on the relationship between the remapping cost and quality. This paper focuses on the issue of when to remap, assuming total remapping is implemented.

2.2 Workload Evolution: Markov Chain Models

A remapping operation resets any workload distribution w_k to a uniform distribution. Its runtime overhead aside, effects of the remapping are also determined by the computation characteristics. For computations following a highly dynamic workload evolution model, remapping is hardly beneficial because the uniform distribution it generates would become severely imbalanced quickly. Instead, an alternative scatter decomposition approach was shown to be effective for such highly dynamic computations [21].

Dynamic remapping was demonstrated to be viable in computations under gradual workload evolution models. Gradual changes of w_k at each step ensure that the remapping cost would be amortized by the benefits in subsequent steps. A popular and elegant gradual workload evolution model uses a Markov chain formulation. A discrete-time finite-state Markov chain is a sequence $\{x_k\}$ of random variables that take values in a finite set of scalars or vectors. No matter the values are scalars or vectors, use the collection of indices $\{1, 2, \ldots, m\}$ to denote the set of states $\mathcal{M}$ for notational simplicity. The Markov property requires that the conditional probabilities satisfy

$$\begin{aligned} &\Pr(x_{k+1} = j | x_1 = i_1, x_2 = i_2, \ldots, x_{k-1} = i_{k-1}, x_k = i) \\ &\quad = p(x_{k+1} = j | x_k = i), \ \forall k \geq 0. \end{aligned}$$

That is, the transition probability of the next state, x_{k+1}, depends only on the current state x_k, not on the remote past. When $\Pr(x_{k+1} = j | x_k = i) = p_{ij}$ is independent of k, the corresponding Markov chain is said to be homogeneous (or the chain has stationary transition probabilities).

For our model, we assume that the workload sequence $\{w_k\}$ is a vector-valued Markov chain over finite state space with a stationary transition probability matrix $P = (p_{ij})$. We also assume that the transition probabilities are unaffected by remapping. The use of Markovian model allows us to treat statistical dependence and more complex structure than the usual independent identically distributed random variable assumptions. Our models are quite general and we do not need the dynamics of the underlying process other than the transition probability matrix.

To illustrate, consider the following examples, in which $\{\xi_k\}$ is a sequence of vector-valued or matrix-valued independent and identically distributed random variables, representing the workload change from time $k-1$ to k.

Example 2.1. Consider the following random walk model:

$$w_{k+1} = w_k + \xi_k.$$

Using such a model as the workload is the focus of [9].

Example 2.2. More generally, workload changes can be modeled as a discrete-time dynamic system

$$w_{k+1} = f(w_k, \xi_k), \tag{6}$$

where $f(\cdot, \cdot)$ is an appropriate function, representing the pattern of workload changes. The model (6) includes the additive changes $f(w_k, \xi_k) = w_k + \xi_k$ and multiplicative changes $f(w_k, \xi_k) = \xi_k^1 w_k + \xi_k^2$ as special cases. Note that, in the multiplicative case, for each k, ξ_k^1 is an $r \times r$ random matrix and ξ_k^2 is a random r-vector.

Molecular dynamics is an example of the Markov chain model. We quantify the workload of a process in terms of the number of atoms residing in a partition associated with the process. Computation over each atom is a basic execution unit. Since the atoms may move across partitions, the workload of processes changes gradually as the simulation proceeds. Because the next workload distribution, w_{k+1}, depends only on the current state w_k, not on the remote past, the workload sequence $\{w_k\}$ obeys a Markov chain model. The workload change is independent of simulation step k and the number of atoms is limited. Hence, the Markov chain is homogeneous with a finite state space. With the atoms moving from one partition to another, the workload of a process in the next step should depend on its own load, as well as the others, although, collectively the vector-valued $\{w_k\}$ obeys the Markovian model. Nicol and Saltz studied the remapping problem, assuming multiple independent Markov chains. This paper goes beyond to more relaxed computation models.

For molecular dynamics programs running on a multiprogrammed system, we normalize the workload with respect to the capacity of processing nodes and round the workload to integers. As a result, for each $i = 1, \dots, r$, the workload w_k^i will take positive integer values and have a finite state space. Without loss of generality, the state space can be ordered and indexed by integers. The Markov state space $\mathcal{M}$ consists of all possible values of w, where w is an $\mathbb{R}^r$-valued vector with the jth component representing the workload of the process j. Note that our formulation using finite state Markov chains is mostly for simplicity of presentation. Countable state space Markov chains can be treated by essentially the same techniques as discussed in this paper, with some modifications.

3 Formulation of the Remapping Problem

Assume that the processes are initially balanced. As the computation proceeds, workloads are generated or consumed. Remapping decisions can be symbolically represented by a sequence of binary-valued functions $\{u_k\}$ such that $u_k = 0$ means a remapping action is taken at time $t = k$, and $u_k = 1$ indicates that no remapping is executed at time k. As a result, the pursuit of optimal remapping strategy can be equivalently expressed as the search for optimal sequence $\{u_k\}$.

For long-lasted computations with an infinite horizon, i.e., $N = \infty$, the decisions are causal in the sense that u_k can be a function of past workload trajectories, as well as the statistical information of the dynamic system (6), but not future workload values. This is analogous to a gambler engaged in a series of games. The gambler cannot foresee (or predict) his or her future. As a result, his or her decision is based only on the information accumulated up to current time. At each time k, the gambler makes a decision on if he or she should continue to play or to stop (preserving his or her accumulated fortune or preventing further loss). His or her objective is to choose the stopping rule to optimize the expected fortune.

3.1 Sequential Stopping Time

The search for an optimal sequence $\{u_k\}$ can be formulated as a sequential stopping time problem. Define a random time τ by

$$\tau = k \text{ if and only if } u_k(w_1, \dots, w_k) = 0. \tag{7}$$

Such a τ is nonanticipative (i.e., $\tau \le k$ is completely determined by observing $w_1, \dots, w_k$). Since it depends only on the information up to the present, not on the future, τ is a stopping time [13]. Starting at time $t = 1$, we search for the first optimal stopping time $\tau_1 > 1$ at which $u_{\tau_1} = 0$, namely, remapping is to be implemented at τ_1. Then, starting at τ_1, the system initial condition is reset to a new value, the dynamic system is renewed, and the optimal stopping decision for the second stopping time τ_2 is pursued, and so on. Under the assumption that the stochastic behavior of the workload distribution change is not affected by remapping, the subsequent remapping time can be found sequentially in a similar manner. Thus, it suffices to study the first remapping time. The resulting optimal remapping strategies for a Markov chain in the state space $\mathcal{M}$ will be expressed in an optimal state-dependent form: Find a stopping set $\mathcal{S} \subseteq \mathcal{M}$ such that u_k switches its values when w_k crosses the boundary of $\mathcal{S}$.

The main task of developing an optimal remapping strategy is now reduced to the formulation of $\mathcal{S}$. To make this problem nonsingular, we shall assume that the stopping time is finite with probability one, i.e., $\tau < \infty$ w. p. 1. That is, a remapping will happen in finite time. Under this framework, a remapping rule is characterized by a sequence $\{u_k\}$.

The optimal stopping time depends on the workloads up to the current time and cannot be determined beforehand. Suppose that $\{u_k\}$ is a sequence of decision rules which control the transition probabilities of the Markov chain. For each choice of the $\{u_k\}$, the transition probabilities $p_{ij}(u_k)$ are fixed. The family of possible Markov processes, each of which corresponds to a fixed sequence of controls or decision rules, is known as a controlled Markov chain. In what follows, we consider $\{w_k\}$ as a controlled Markov chain.

3.2 Optimal Remapping

To find the optimal stopping rule, we transform the optimal stopping problem to an equivalent binary decision problem in which the cost function depends only on the states w_k, i.e., a cost function that depends only on if the workload is in the stopping set. Subsequently, the workloads are naturally divided into two disjoint sets, a continuation set $\mathcal{C}$ and a stopping set $\mathcal{S}$ satisfying $\mathcal{M} = \mathcal{C} \cup \mathcal{S}$. When $w_k \in \mathcal{C}$, the dynamic system continues to run. If, however, w_k is in the complement of $\mathcal{C}$, namely, the stopping set $\mathcal{S}$, a remapping is carried out.

Based on this model, the controls become a binary action, either continuing or stopping (remapping). Consider a sequence of controls $\{u_k\}$ such that $u_k = u(w_k)$ with

$$u(w) = \begin{cases} 0, & \text{there is a remapping when the state is } w, \\ 1, & \text{otherwise.} \end{cases} \tag{8}$$

As was alluded to before, the computation is more or less running an indefinitely long time, the problem under consideration is then naturally formulated as an infinite horizon one in which the advantage of dealing with a "steady state" problem in lieu of a time-dependent problem is particularly pronounced and the computations involved are substantially simpler than that of the finite-horizon counter part.

Let $\phi(w)$ and $\eta(w)$ denote the execution time of a step and the runtime cost for a remapping, respectively, when the state is in w. The step execution time $\phi(w)$ is the average execution time of a step, plus the penalty of load imbalance $\psi(w)$, as defined in (4). Given a workload distribution $w = (3, 5, 1)$, we have $\psi(w) = 2$ and $\phi(w) = 5$. We define a time-independent cost function as

$$c(w, u) \overset{\text{def}}{=} \begin{cases} \eta(w), & \text{if } u = 0 \\ \phi(w), & \text{if } u = 1, \end{cases} \tag{9}$$

where $\eta(\cdot)$ and $\phi(\cdot)$ are some continuous functions. It follows that finding the optimal remapping time is equivalent to finding the optimal control that minimizes

$$J(u,i)=E_i\sum_{k=1}^{\infty}c(w_k,u_k), \tag{10}$$

where E_i denotes the expectation taken with the initial data $w_1=\alpha_i$. Recall that we have decided to use i, the index of the state of the underlying Markov chain, to represent the vector-valued state α_i. It is also noted that $c(w,u)$ excludes the execution time $\psi(w)$ in the case of remapping. This is because a remapping operation drives the state w into a new state and the cost of the new state is to be included in the case of no-remapping.

To facilitate the subsequent theoretical development, we assume that: The Markov chain $\{w_k\}$ has a finite state space $\mathcal{M}$, which includes absorbing states $0_1,\dots,0_{m_0}$. The interpretation corresponding to our system is that each of these absorbing states is a balanced state. Every other state has a positive transition probability to one of the absorbing states. To illustrate, for example, we suppose that the bulk synchronous computation is comprised of two processes and the possible maximum workload value of each process is b. Then, the states $(1,1)$, $(2,2)$, ..., (b,b) are absorbing states (or balanced states) and all other b^2-b states are transient, which will eventually be "absorbed" into one of these absorbing states by using our control strategy.

With the index notation convention, the state space of the controlled Markov chain can be written as

$$\begin{aligned}\mathcal{M}&=\mathcal{C}\cup\mathcal{S}\\&=\{1,2,\dots,m_1\}\cup\{0_1,0_2,\cdots 0_{m_0}\},\end{aligned}$$

with $m=m_0+m_1$. For any $x\in\mathcal{M}$, a control given by (8) is used. Taking into consideration the decision variables, we represent the corresponding transition probability matrices of the controlled Markov chain by $P(u)=\{p_{ij}(u(i))\}$ with

$$\begin{cases}p_{i0_j}(0)=1, & j=1,\dots,m_0,\\ p_{0_j0_j}(u(0))=1 & j=1,\dots,m_0.\end{cases}$$

That is, if $u(i)=0$, $p_{i0_j}(u(i))=1$ and, regardless of the value of $u(0(i))$, $p_{0(i)0(i)}(u(0(i)))=1$. The controlled Markov chain defined on $\mathcal{M}$ represents the process of the actual workload evolution on $\mathcal{M}$ as long as the process continues without remapping. As soon as a remapping action is taken, the Markov chain enters one of the absorbing states (or balanced state). Throughout the paper, assume that $P(\cdot)$ is a continuous function.

Consider the controls over computations in steps $[0,N]$, for each $i\in\mathcal{M}$,

$$J(u,i)=E_i\sum_{k=1}^{N}c(w_k,u_k).$$

We will rewrite it in a vector form to simplify the derivation of policy iteration type results. The policy iteration procedure is a frequently used numerical scheme. It proceeds recursively. At each iteration, one chooses an improved control policy as compared to the previous iteration.

Let $u^k(i)$ denote the control when there are k-units of time left to go when the state is in i. It follows that $u_k(i)=u^{N-k}(i)$. Denote $u^k=(u^k(1),\dots,u^k(m))'$ and use μ^k to denote the collection of controls $\mu^k=(u^k,\dots,u^1)$. We have that $\mu^k=(u^k,\mu^{k-1})=(u^k,u^{k-1},\mu^{k-2})$, etc. Denote by $E_i^{\mu^k}$ the expectation taking with initial $w_1=\alpha_i$ and policy μ^k being used. We obtain that

$$\begin{aligned}J(\mu^N,i)&=E_i^{\mu^N}\sum_{k=1}^{N}c(w_k,u_k(w_k))\\&=E_i^{\mu^N}\left(c(i,u^N(i)\right)\\&\quad+E_i^{\mu^N}\left(\sum_{k=1}^{N-1}c(w_k,u^{N-k}(w_k))|w_1\right)\\&=c(i,u^N(i))+E_i^{\mu^N}J(\mu^{N-1},w_1)\\&=c(i,u^N(i))+\sum_j p_{ij}(u^N(i))J(\mu^{N-1},j).\end{aligned}$$

The derivation is essentially a restatement of Bellman's optimality principle (see [2], [8], [12]). The dynamic programming techniques decompose the problem into a sequence of simpler minimization problems that are carried out over the control space rather than over a space of functions. Loosely, the principle of optimality states: Suppose that $(u_0^*,u_1^*,\dots,u_{n-1}^*)$ is an optimal control for the basic problem. Consider the subproblem in which we are at state w_j at time j and wish to minimize the cost-to-go function from time j to time n. Then, the truncated control law $(u_j^*,u_{j+1}^*,\dots,u_{n-1}^*)$ is also optimal for the subproblem.

Denote

$$\begin{aligned}&J(\mu^k)=(J(\mu^k,1),J(\mu^k,2),\dots,J(\mu^k,m))\\&\tilde{P}(u)=(p_{ij}(u(i)),i,j\geq 1)\in\mathbb{R}^{m\times m}\\&C(u^k)=(c(1,u^k(1)),\dots,c(m,u^k(m)))\\&\widehat{\eta}(u)=(\eta(1),\eta(2),\dots,\eta(m)).\end{aligned}\tag{11}$$

Note that $\tilde{P}(u)$ is taken from the transition matrix $P(u)$ with the absorbing states deleted. We rewrite the cost function $J(\mu,i)$ in a vector form as

$$J(\mu^{k+1})=\tilde{P}(u)J(\mu^k)+C(u). \tag{12}$$

For the controls over an infinite horizon $[0,\infty)$, using the idea of cost function expansion, we can write it in terms of the stopping rule as $J(\tau)$ with

$$J(\tau)=\tilde{P}(u)J(\tau)+C(u). \tag{13}$$

Equation (13) implies a policy iteration procedure. Its fixed point is an optimal control. In the following, we analyze the optimality of the control. We often write τ as τ_u in what follows to signal the stopping time depending on the control u.

4 Optimality and Policy Iteration Algorithm

There are three key issues associated with the optimal control (13), namely, the necessary and sufficient conditions for optimality, the existence of optimal controls, and algorithms for obtaining the optimal control. The rest of this section will be devoted to resolve each of these issues. The development follows closely the methodology of Kushner [12]. The main assumptions to be used are:

(A) Assume the Markov chain $\{w_k\}$ has a finite state $\mathcal{M}$ that contains absorbing states $0_1,0_2,\dots,0_{m_0}$. There exists a $0<c_0<1$ and $l\leq m_1$ such that the l-step transition probability

$$\sum_{j=1}^{m_0} p_{i0_j}^{(l)}(u,u,\dots,u) \geq c_0 > 0 \tag{14}$$

for each u and each i.

Note that $p_{ij}^{(l)}(u,\dots,u)$ denotes the l-step transition probabilities with control policy $(u,\dots,u)$ used. Condition (A) indicates that, for some l, the l-step transition probability from any state i to the set of absorbing states is strictly positive. In remapping, this merely states that the optimal remapping should result in a strategy in which at least one remapping will occur at a finite time with a positive probability. Otherwise, the optimization problem does not have a finite solution.

4.1 Optimality

This section gives criteria of optimality for remapping. Some of the important questions to answer are: What conditions should the optimal cost satisfy? How can one check and determine if optimality is reached? We present such results here and provide necessary and sufficient conditions for the optimal cost.

The technical developments rely on the properties of the cost function. We shall use the idea of contraction mappings to carry out the discussion. Recall that a linear operator T is a contraction if $|T| < 1$. If T is a contraction, its eigenvalues are strictly inside the unit circle and the equation $x = Tx + b$ for a given constant b has a unique solution.

Lemma 4.1. *Under (A), corresponding to the transition probability matrix $P(u)$ of the controlled Markov workload process, $\tilde{P}^{m_1}(u)$ is a contraction uniformly in u and $\prod_{j=1}^{m_1}\tilde{P}(u^j)$ is a contraction uniformly in u^j, $j \leq n$. In addition,*

$$\prod_{j=1}^{n}\tilde{P}(u^j) \to 0 \quad \text{as } n \to \infty \tag{15}$$

uniformly in u^j, $j \leq n$.

Proof. To prove the first assertion, we note that, for each u, by virtue of (A),

$$\begin{aligned}|\tilde{P}^{m_1}(u)| &= \max_i \sum_{j\geq 1} p_{ij}^{(m_1)}(u,\dots,u)\\ &= \max_i [1 - \sum_{j=1}^{m_0} p_{i0_j}^{(m_1)}(u,\dots,u)]\\ &\leq 1 - c_0 < 1.\end{aligned}$$

Moreover the bound holds uniformly in u. Thus, $\tilde{P}^{m_1}(u)$ is a contraction uniform in u. Next consider $\prod_{j=1}^{m_1}\tilde{P}(u^j)$. Since the product contains only finitely many terms, choose $\tilde{u}$ such that

$$\max_{u^j}\left|\prod_{j=1}^{m_1}\tilde{P}(u^j)\right| \leq \left|\tilde{P}^{m_1}(\tilde{u})\right|.$$

Condition (A) and the preceding proof yields that $\tilde{P}^{m_1}(\tilde{u})$ is a contraction and

$$\begin{aligned}\left|\prod_{j=1}^{n}\tilde{P}(u^j)\right| &\leq \left|\tilde{P}^{m_1}(\tilde{u})\right|\\ &\leq 1 - c_0 < 1,\end{aligned} \tag{16}$$

and the bound holds uniform in u^j, $j \leq n$. Thus, $\prod_{j=1}^{m_1}\tilde{P}(u^j)$ is a contraction uniform in u^j with $j \leq n$. In fact, it can be verified that $|\prod_{j=\ell m_1+1}^{(\ell+1)m_1}\tilde{P}(u^j)| \leq 1 - c_0$ by using the same techniques. Thus,

$$\begin{aligned}a_{\ell+1} &\overset{\text{def}}{=} \left|\prod_{j=1}^{(\ell+1)m_1}\tilde{P}(u^j)\right|\\ &\leq \left|\prod_{j=\ell m_1+1}^{(\ell+1)m_1}\tilde{P}(u^j)\right|\left|\prod_{j=1}^{\ell m_1}\tilde{P}(u^j)\right|\\ &\leq (1-c_0)a_\ell\\ &\leq (1-c_0)^\ell a_1\\ &\to 0 \text{ as } \ell \to \infty.\end{aligned}$$

That is, $\prod_{j=1}^{\ell m_1}\tilde{P}(u^j) \to 0$ as $\ell \to \infty$. Finally, for any n, there is an ℓ such that, as $e \to 0$,

$$\left|\prod_{j=1}^{n}\tilde{P}(u^j)\right| = \left|\prod_{j=\ell m_1+1}^{n}\tilde{P}(u^j)\prod_{j=1}^{\ell m_1}\tilde{P}(u^j)\right| \to 0.$$

Thus, (15) follows. □

The purpose of this lemma is to prepare us for studying limit behavior of the cost and optimality. The above results ensure that, under the boundedness of the state space of the Markov chain and (14), the transition matrices eventually diminishes in the long run. The rational, is that, by looking at definition in (11), $\tilde{P}(u)$ essentially represents the transition probabilities of the transient states. Next, it is natural to ask: Will the cost be finite? We have formulated a stopping rule problem, but do such desired stopping rules really exist? If the stopping time is not finite, then the result will be useless. In what follows, we show that the cost, in fact is finite, together with the existence of finite stopping time.

Lemma 4.2. *Suppose (A) holds. Then,*

1. *The expected stopping time $E_i\tau_u$, for any initial state i, is finite. $E_i\tau_u < \infty$.*
2. *There is a unique solution to*

$$J = \tilde{P}(u)J + C(u), \tag{17}$$

and the solution is bounded.

Proof. By virtue of (A), the proof of [12, Theorem 8, p. 120] implies that $E_i\tau_u < \infty$ for each i. Thus, 1 holds.

In view of the definition of the running cost function, for each $i \in \mathcal{M}$ (recall the index convention), $\phi(w) \leq K_1$ for some $K_1 > 0$. So, the running cost is finite for each i. According to [12, Theorem 8, p. 120], $E_i\tau_u < \infty$ and the finite running cost for each i lead to that there is a unique uniformly bounded solution of (17). The lemma is thus proven. □

The above two lemmas prove the regularity of the cost function defined in the preceding section. We are now in a position to provide necessary and sufficient conditions for the optimal cost.

Propostion 4.3. *Assume that (A) is satisfied. Let u^* be a remapping control strategy and $J(\tau_u) = J$ be a uniformly* *bounded vector-valued cost function. The control u^* is optimal if and only if*

$$\begin{aligned} J &= \min_u[\tilde{P}(u)J + C(u)] \\ &= \tilde{P}(u^*)J + C(u^*) \\ &= \min(\widehat{\eta}, \tilde{P}(u)J + C), \end{aligned} \tag{18}$$

where, for a vector v, $\min(v)$ is taken component-wise.

Remark 4.4. The first line of (18) merely indicates that if J is the minimal cost, it must be a solution of this equation. The second line makes it clear that u^* is the optimal remapping rule. The last line of (18) is a restatement of Bellman's principle of optimality. One compares the cost of remapping with $\widehat{\eta}$ defined in (11) with that of continuing job processing and picks out the smaller one between these two alternatives.

Proof. This result gives a necessary and sufficient condition for optimality. We concentrate on the first two lines of (18). Iterating on

$$J = \tilde{P}(u^*)J + C(u^*)$$

leads to

$$J = \tilde{P}^n(u^*)J + \sum_{j=0}^{n-1} \tilde{P}^j(u^*)C(u^*).$$

By virtue of Lemma 4.1, $\tilde{P}^n(u^*) \to 0$ and $E_i\tau_{u^*} < \infty$. It then follows that $J = J(\tau_{u^*})$. For any control u with a finite cost, $\tilde{P}^n(u) \to 0$. By using the first line of (18),

$$J \le \tilde{P}(u)J + C(u) \le \tilde{P}^n(u)J + \sum_{j=0}^{n-1} \tilde{P}^j(u)C(u) \to J(\tau_u),$$

so u^* is optimal.

To prove the converse, suppose J is an optimal cost and u^* is the corresponding optimal control, but (18) does not hold. By Lemma 4.2, J is uniformly bounded. Suppose that there is a control u with finite cost such that

$$J \ge \tilde{P}(u)J + C(u)$$

and there is at least one component i_0 for which the strict inequality holds. Again, iterating on the above inequality, yields that

$$J \ge \tilde{P}^n(u)J + \sum_{j=0}^{n-1} \tilde{P}^j(u)C(u) \to J(\tau_u)$$

and $J > J(\tau_u)$. This, however, contradicts the optimality.

Finally, if we start at state i, we can either stop and pay a cost $\eta(i)$ or continue and pay an average cost $E_iJ(w_1)$. The minimal cost is the smaller of these two. Thus, the last line (18) follows. □

4.2 Existence of Optimal Control

Proposition 4.5. *Assume (A). Then, there is a unique solution to (18) and the optimal remapping solution is given by the solution of (18).*

Proof. Suppose that there are two solutions J and $\tilde{J}$ to (18) with the corresponding controls u and $\tilde{u}$. Then,

$$\begin{aligned} J &= \tilde{P}(u)J + C(u) \le \tilde{P}(\tilde{u})J + C(\tilde{u}) \\ \tilde{J} &= \tilde{P}(\tilde{u})\tilde{J} + C(\tilde{u}) \le \tilde{P}(u)\tilde{J} + C(u). \end{aligned}$$

This implies that $\tilde{P}^n(u)(J - \tilde{J}) \le \tilde{P}^n(\tilde{u})(J - \tilde{J})$. Since $\tilde{P}^n(u)$ and $\tilde{P}^n(\tilde{u})$ are contractions, by Lemma 4.1, we must have $J = \tilde{J}$ and the uniqueness is obtained.

In view of Propostion 4.3, it suffices to show that there is a minimal cost J and control u^* such that the second equality in (18) holds. Suppose that τ_u and $\tau_{\tilde{u}}$ are two stopping rules corresponding to the controls u and $\tilde{u}$, respectively. Lemma 4.2 implies that $E_i\tau_u < \infty$ and $E_i\tau_{\tilde{u}} < \infty$. As in [12], it can be shown that there is a control v such that $J(\tau_v) \le \min(J(\tau_u), J(\tau_{\tilde{u}}))$. Define $\tau_n = \tau_v \wedge n$ and denote the corresponding control by $u^{(n)}$. Then, it is easily seen that τ_n is a sequence of stopping times. It follows that $\tau_n \to \tau$, $u^{(n)} \to u$, and $w_{\tau_n} \to w_\tau$ as $n \to \infty$ w.p. 1 because

$$\begin{aligned} J(\tau_n, i) - J(\tau, i) &= E_i\left[\sum_{j=1}^{\tau_n - 1} c\left(w_j, u^{(n)}\right) - \sum_{j=1}^{\tau-1} c(w_j, u)\right] \\ &\quad + E_i(\eta(w_{\tau_n}) - \eta(w_\tau)). \end{aligned}$$

The dominated convergence theorem then implies that the right-hand side of the above equation goes to 0 and hence, $J(\tau_n, i) - J(\tau, i) \to 0$ as $n \to \infty$. □

4.3 Policy Iteration Algorithm for Optimal Remapping

It is of foremost importance to have a feasible computational procedure that enables us to obtain the optimal control. We use the idea of policy iteration in stochastic control theory; see [2], [12] for details.

The Policy Iteration Algorithm. The algorithm is fairly simple.

- Initialization step: Choose V^0 as an arbitrary and uniformly bounded vector.
- Iteration step: Construct the approximation at step $n+1$ by

$$V^{n+1} = \min_u\bigl(\hat{\eta}(u), \tilde{P}(u)V^n + C(u)\bigr). \tag{19}$$

The algorithm given above is known as *policy iteration* procedure for obtaining an optimal policy starting with an arbitrary policy. In a typical iteration of this algorithm, given a control policy u and the corresponding cost $C(u)$, one obtains an improved policy satisfying (19). The new policy is strictly better if the current policy is nonoptimal.

Proposition 4.6. *The iteration given by (19) is stable and, as $n \to \infty$, V^n converges to the minimal cost at a geometric rate.*

Proof. An equivalent form of (19) can be written as

$$V^{n+1} = \min_u(\tilde{P}(u)V^n + C(u)).$$

Let u be the optimal control satisfying (18) and $u^{(n)}$ be the nth iterate of the control corresponding to (19). Then, by the optimality,

$$\begin{aligned} &\tilde{P}(u^{(n)})V^n + C(u^{(n)}) = V^{n+1} \le \tilde{P}(u)V^n + C(u) \\ &\tilde{P}(u)J + C(u) = J \le \tilde{P}(u^{(n)})J + C(u^{(n)}). \end{aligned} \tag{20}$$

Taking the difference of the above two inequalities leads to

$$\tilde{P}(u)(J - V^n) \le J - V^{n+1} \le \tilde{P}(u^{(n)})(J - V^n).$$

Iterating on the above inequalities yields

$$\tilde{P}^{n+1}(u)(J - V^0) \le J - V^{n+1} \le \prod_{j=0}^{n} \tilde{P}(u^{(j)})(J - V^0), \quad (21)$$

where

$$\prod_{j=0}^{n} \tilde{P}(u^{(j)}) = \tilde{P}(u^{(n)})\tilde{P}(u^{(n-1)}) \cdots \tilde{P}(u^{(0)}).$$

By virtue of Lemma 4.1, $\prod_{j=0}^{n} \tilde{P}(u^{(j)}) \to 0$ as $n \to \infty$. Therefore, as $n \to \infty$, both the left-hand side and the right-hand side of (21) go to 0. The desired result follows.

The convergence rate of the iterative process can be easily seen from the structure of $\tilde{P}(u)$. Note that the transition probabilities collected in $\tilde{P}(u)$ are those for the transient states. Thus,

$$\sum_{j \ge 1} p_{ij}^n(u, \ldots, u) = 1 - \sum_{j=1}^{m_0} p_{i0_j}^n(u, \ldots, u) \le 1 - c_0.$$

Note also that $\tilde{P}^n(u) = (p_{ij}^n(u, \ldots, u), i, j \ge 1)$ is the n-step transition matrix with control policy $(u, \ldots, u)$ used. The contraction properties then imply that the rate of convergence is geometric. □

We note that each iteration involves a "policy evaluation" step whereby, given a control policy u, we obtain the corresponding cost vector $J(u)$ by solving the system of (19). This step can be time-consuming when the number of states is large. Previous studies have shown that parallel implementations for linear equations can reduce the time complexity of each policy iteration step to as little as $O(n)$ when $O(n^2)$ processors are used [3]. Under these circumstances, the policy iteration algorithm can be attractive since it typically requires few iterations for termination.

5 An Example

In this section, we present a concrete example to illustrate the optimal remapping strategy. For succinctness in presentation, we consider a computation comprised of two parallel processes and assume that the maximum process workload $w_{\max}^i = 5$. The optimal remapping strategy is readily applicable to large scale systems and large workload state spaces because of its extremely low time and space complexities.

Assume that the workload at each process changes in a random walk model

$$w_{k+1}^i = w_k^i + \xi_k$$

and that the workload change at each step follows distribution functions

$$\xi_k = \begin{cases} 1, & \text{w.p.} \quad 0.5, \\ & \qquad\qquad \text{if } w_k^i = 0, \\ 0, & \text{w.p.} \quad 0.5, \end{cases}$$
$$\xi_k = \begin{cases} -1, & \text{w.p.} \quad 0.5, \\ & \qquad\qquad \text{if } w_k^i = w_{\max}^i, \\ 0, & \text{w.p.} \quad 0.5, \end{cases} \quad (22)$$
$$\xi_k = \begin{cases} 1, & \text{w.p.} \quad 0.25, \\ 0, & \text{w.p.} \quad 0.5, \quad \text{otherwise.} \\ -1, & \text{w.p.} \quad 0.25, \end{cases}$$

This computation model can be characterized by a Markov chain over a state space

$$\mathcal{M} = \{(i, j) | i, j \in \{0, 1, 2, 3, 4, 5\}\},$$

where states (i, i) for $i = 0, 1, \ldots, 5$, are the absorbing states. Whenever the computation gets into one of the states, the computation starts anew or terminates. From the distribution functions (22), we obtain a transition matrix $P = (p_{(i_1,j_1),(i_2,j_2)})_{36 \times 36}$ of the Markov chain and

$$p_{(i_1,j_1),(i_2,j_2)} = p_{i_1,i_2} \times p_{j_1,j_2}.$$

For example,

$$p_{(1,2)(2,2)} = p_{1,2} \times p_{2,2} = 0.25 \times 0.5 = 0.125.$$

The vector-valued state space defined by the distribution functions is shown in Fig. 3. It is in the structure of two-dimensional grids. In general, the state space of a Markov chain workload evolution model for a computation with r processes is in the structure of r-dimensional grids. It is the diagonal state transitions that distinguishes this model from the multiple Markov chain model in [20]. The transitions indicate that the next state of a process depends on its current workload as well as the others.

Assume that the grid points are numbered in a row-major fashion from left to right and from top to bottom. We obtain a transition matrix over the total number of 36 states as a direct product of two tridiagonal matrices:

$$P = \frac{1}{8} \begin{pmatrix} 2Q & 2Q & & & & \\ Q & 2Q & 2Q & & & \\ & Q & 2Q & Q & & \\ & & Q & 2Q & Q & \\ & & & Q & 2Q & Q \\ & & & & 2Q & 2Q \end{pmatrix} = \frac{1}{8} Q \otimes Q,$$

where $Q \otimes Q$ is the Kronecker product of two matrices, and

$$Q = \begin{pmatrix} 2 & 2 & & & \\ 1 & 2 & 1 & & \\ & 1 & 2 & 1 & \\ & 1 & 2 & 1 & \\ & & 2 & 2 & \end{pmatrix}.$$

By the definition of $\tilde{P}(u)$ in (11), we cross out the rows and columns corresponding to the absorbing states from P and obtain a transition matrix of the *controlled Markov chain* without remapping, $\tilde{P}(1)$. Note that $\tilde{P}(1)$ is essentially the transition matrix between imbalance states. A remapping control (i.e., u = 0) in the controlled Markov random process

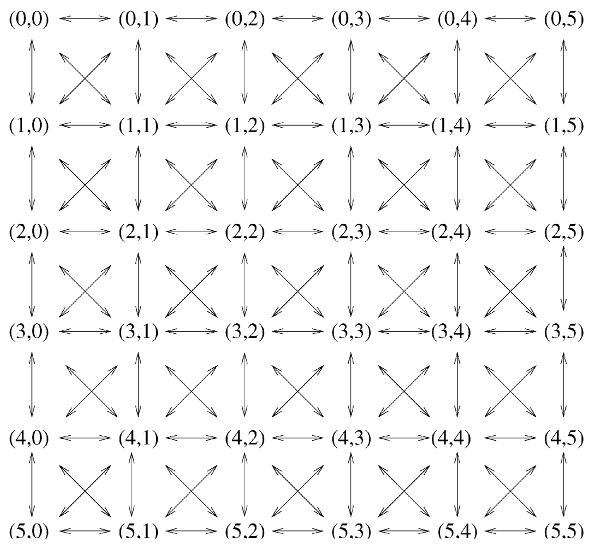

Fig. 3. Vector-valued state space of a random walk computation model.

transforms any transient (imbalance) state to an absorbing (balanced) state. Hence, the assumption (A) holds. We simply set the remapping transition matrix $\tilde{P}(0)$ to a uniform matrix $\frac{1}{36}U$, where $U = (u_{ij})_{30\times30}$ and $u_{ij} = 1$. The scaling factor $\frac{1}{36}$ is due to the fact that the controlled Markov chain contains six absorbing states out of a total of 36 states and that any transient state has a probability of 5/6 to remain in the transient set.

Next, we define the penalty of load imbalance ϕ and remapping cost η. Since the duration of each computation step is determined by the most heavily loaded process, we set the penalty to $|i-j|/2$ for a state (i, j). The remapping cost is an application dependent factor. It is measured relative to the load imbalance penalty.

Given the bulk synchronous computation settings in this example, we experimented with the policy iteration algorithm (19), assuming different remapping costs. Fig. 4 shows the remapping decisions. Shadowed states refer to those that would benefit from remapping. From this figure, it can be seen that the remapping decisions are highly accurate. For example, in Fig. 4a, where the remapping case $\eta = 1$, any load distribution with a difference of two or more workload units deserves a remapping. With the increase of remapping cost, fewer and fewer states deserve a remapping. When the remapping cost goes beyond a certain number, the computation should never carry any remapping. This is in agreement with past practical experience in the literature. Fig. 4c also reveals that state (4, 1) has more potential benefits from remapping than the boundary states (3, 0) and (5, 2), although they have the same cost. States (3, 0) and (5, 2) have higher probabilities of evolving into more balanced states.

In order to illustrate the time efficiency of the algorithm (19), we plot in Fig. 5 the number of iterations required to reach a remapping decision versus the remapping cost. It is expected that, as the remapping cost increases, the algorithm would need to look forward to more steps for making a good tradeoff between the cost and potential benefits and the number of iterations would increase accordingly. In any case, the algorithm terminates quickly in tens of iteration steps. This confirms the geometric convergence rate of the algorithm.

Finally, we point out that the policy iteration algorithm is based on a vector-valued load state space. The probability transition matrix P in this example is a block tridiagonal matrix of size $6^2 \times 6^2$. It consists of three consecutive diagonals composed of matrix blocks along the principal diagonal. The block diagonals are tridiagonal matrices and each contains at most six nonzero entries. In general, for a computation with r processes and assuming a maximum of m workload units per process, the transition probability matrix of its Markov chain is of size $m^r \times m^r$. It is a block r-diagonal matrix. That is, there are r block diagonals adjacent to the principal diagonal on each side. It is represented by direct product of a sequence of block diagonals. Since each block diagonal is a tridiagonal matrix and of size m, the storage complexity of the algorithms can be reduced to $O(m)$.

6 Concluding Remarks

In this paper, we have proposed and developed a novel remapping strategy for dynamic bulk synchronous computations whose workload changes can be modeled as a Markov chain. The use of Markovian model allows us to

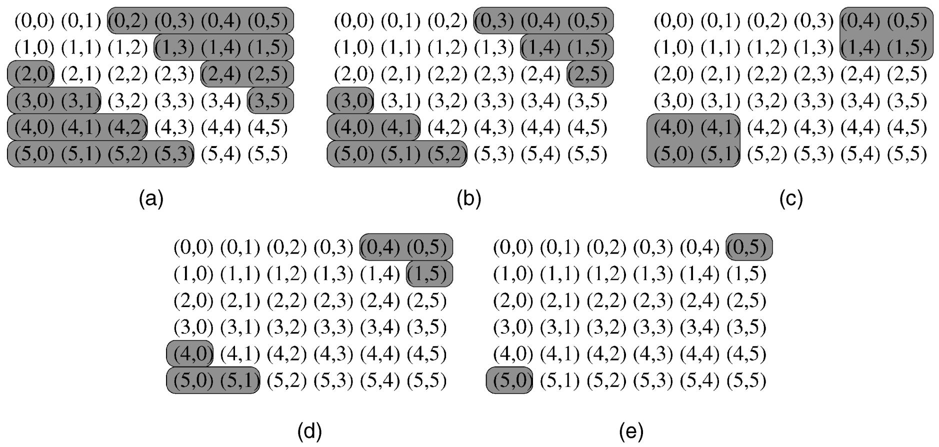

Fig. 4. Remapping decisions under different remapping cost models. (a) cost = 1, (b) cost = 2, 3, 4, (c) cost = 5, 6, (d) cost = 7, 8, 9, and (e) cost = 10, 11, 12, 13.

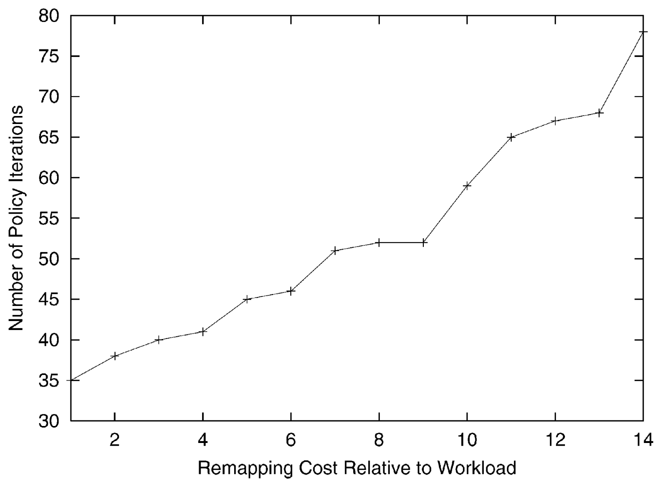

Fig. 5. Number of policy iterations versus remapping cost.

treat statistical dependence and more complex structure than the usual independent identically distributed random variable assumptions. Our models are quite general and we do not need to impose conditions on the dynamics of the underlying process other than the transition probability matrix. Optimal remapping can be formulated as a binary decision process: remap or not at a given synchronizing instant. We have developed the optimal invocation polices for long lasted computations by employing optimal stopping rules in a stochastic control framework. The existence of optimal controls is established. Necessary and sufficient conditions for the optimality are obtained. We have also devised a policy iteration algorithm to reduce computational complexity and enhance fast convergence to the desired optimal control.

In this paper, we mainly treated the case that the transition matrix is known. What if this information is not available? In practice, relative frequency and/or history of the workload may be used to identify or estimate the transition matrix. Another technique is known as adaptive control, which has an ingredient to replace the true parameter by its estimate at each step to update the control action. A more sophisticated technique is motivated from machine learning, known as Q-learning (see [13] and the references therein). Suppose that the system can be observed or measured under any choice of actions (either continuing the computation or stopping and remapping). The algorithm involves recursive estimation of the so-called Q-function $Q(i,d)$ (the cost given that we start at state α_i and action d is taken). The estimate of $Q(i,d)$ turns out to be much simpler than the estimation of the transition probabilities.

Acknowledgments

This work was supported in part by the US National Science Foundation under grants DMS-9877090, CCR-9988266, ACI-0203592, and ECS-9634375. The authors would like to thank the anonymous reviewers for their detailed constructive comments and suggestions. They also thank Guosheng Li and Linfeng Zhang for assistance in the experiments.

References

[1] V.D. Agrawal and S.T. Chakradhar, "Performance Analysis of Synchronized Iterative Algorithms on Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 6, pp. 739-746, Nov. 1992.
[2] D. Bertsekas, *Dynamic Programming Deterministic and Stochastic Models.* Englewood Cliffs, N.J.: Prentice Hall, 1987.
[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, 1989.
[4] A.N. Choudhary, B. Narahari, and R. Krishnamurti, "An Efficient Heuristic Scheme for Dynamic Remapping of Parallel Computations," *Parallel Computing,* vol. 19, pp. 621-632, 1993.
[5] A.N. Choudhary and R. Ponnusamy, "Run-Time Data Decomposition for Parallel Implementation of Image Processing and Computer Vision Tasks," *Concurrency: Practice and Experience,* vol. 4, no. 4, pp. 313-334, June 1992.
[6] D. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann, 1998.
[7] G. Edjlali, G. Agrawal, A. Sussman, J. Humphries, and J. Saltz, "Compiler and Runtime Support for Programming in Adaptive Parallel Environments," *Scientific Programming,* Jan. 1997.
[8] W. Fleming and R. Rishel, *Deterministic and Stochastic Optimal Control.* New York: Springer-Verlag, 1975.
[9] F.-T. Fong, C.-Z. Xu, and L. Wang, "Optimal Periodic Remapping of Bulk Synchronous Computations on Multiprogrammed Distributed Systems." *Proc. 14th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 103-108, Apr. 2000.
[10] R.V. Hanxleden and L.R. Scott, "Load Balancing on Message Passing Architectures," *J. Parallel and Distributed Computing,* vol. 13, no. 3, pp. 312-324, Nov. 1991.
[11] J. De Keyser and D. Roose, "Multigrid with Solution-Adaptive Irregular Grids on Distributed Memory Computers," *Parallel Computing,* D.J. Evans, G.R. Joubert, and H. Liddell, eds., pp. 375-382, 1992.
[12] H. Kushner, *Introduction to Stochastic Control Theory.* New York: Holt, Rinehart and Winston, 1972.
[13] H.J. Kushner and G. Yin, *Stochastic Approximation Algorithms and Applications.* New York: Springer-Verlag, 1997.
[14] S. Madala and J.B. Sinclair, "Performance of Synchronous Parallel Algorithms with Regular Structures," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, no. 1, pp. 105-116, Jan. 1991.
[15] D.C. Marinescu and J.R. Rice, "Synchronization and Load Imbalance Effects in Distributed Memory Multiprocessor Systems," *Concurrency: Practice and Experience,* vol. 3, no. 6, pp. 593-625, Dec. 1991.
[16] B. Moon and J. Saltz, "Adaptive Runtime Support for Direct Simulation Monte Carlo Methods on Distributed Memory Architectures," *Scalable High-Performance Computing Conf.,* pp. 176-183, May 1994.
[17] J.E. Moreira, V.K. Naik, and S.P. Midkiff, "Dynamic Data Distribution and Processor Repartitioning for Irregularly Structured Computation," *J. Parallel and Distributed Computing,* vol. 50, pp. 28-60, 1998.
[18] D. Nicol and G. Ciardo, "Automated Parallelization of Discrete State-Space Generation," *J. Parallel and Distributed Computing.* vol. 47, no. 2, pp. 153-167, Dec. 1997.
[19] D.M. Nicol and P.F. Reynolds, "Optimal Dynamic Remapping of Data Parallel Computation," *IEEE Trans. Computers,* vol. 39, no. 2, pp. 206-219, Feb. 1990.
[20] D.M. Nicol and J.H. Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands," *IEEE Trans. Computers,* vol. 37, no. 9, pp. 1073-1087, Sept. 1988.
[21] D.M. Nicol and J.H. Saltz, "An Analysis of Scatter Decomposition," *IEEE Trans. Computers,* vol. 39, no. 11, pp. 1337-1345, Nov. 1990.
[22] L. Oliker and R. Biswas, "Efficient Load Balancing and Data Remapping for Adaptive Grid Calculations," *Proc. Ninth Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '97),* 1997.
[23] G.D. Peterson and R.D. Chamberlain, "Beyond Execution Time: Expanding the Use of Performance Models," *IEEE Parallel and Distributed Technology,* pp. 37-49, Summer 1994.
[24] W. Shu and M.-Y. Wu, "Runtime Incremental Parallel Scheduling (RIPS) on Distributed Memory Computers," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 6, pp. 637-649, June 1996.
[25] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 3, pp. 235-248, Mar. 1998.

[26] R.D. Williams, "Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations," *Concurrency: Practice and Experience*, vol. 3, no. 5, pp. 451-481, Oct. 1991,

[27] C.-Z. Xu and F. Lau, "Decentralized Remapping of Data-Parallel Computations with the Generalized Dimension Exchange Method," *Proc. 1994 Scalable High Performance Computing Conf.*, pp. 414-421, May 1994.

[28] C.-Z. Xu and F. Lau, *Load Balancing in Parallel Computers: Theory and Practice.* Kluwer Academic. 1997.

[29] C.-Z. Xu, F. Lau, and R. Diekmann, "Decentralized Remapping of Data Parallel Applications in Distributed Memory Multiprocessors," *Concurrency: Practice and Experience*, pp. 1351-1376, Dec. 1997.

[30] C.-Z. Xu and N. Nie, "Relaxed Implementation of Spectral Methods for Graph Partitioning," *Proc. Fifth Int'l Symp. Solving Irregular Problems in Parallel*, pp. 366-375, Aug. 1998.

Gang George Yin received the MS degree in electrical engineering and PhD degree in applied mathematics from Brown University in 1987. He is a professor of mathematics at Wayne State University. He is an IEEE fellow, a member of AMS, and SIAM. He serves on the IFAC Technical Committee on Modeling, Identification, and Signal Processing. He was an associate editor of the *IEEE Transactions on Automatic Control*, the editor of the *SIAM Activity Group on Control and Systems Theory Newsletters*, and was on the editorial board of *Stochastic Optimization & Design*. He was on the Mathematical Review Date Base Committee and various conference program committees. He was the SIAM representative to the 34th CDC, and was chairman of 1996 AMS-SIAM Summer Seminar in Applied Mathematics.

Cheng-Zhong Xu received the BS and MS degrees from Nanjing University, China, in 1986 and 1989, respectively, and the PhD degree from the University of Hong Kong, in 1993, all in computer science. He is currently an associate professor in the Department of Electrical and Computer Engineering at Wayne State University. Dr. Xu has published more than 50 peer-reviewed papers in various journals and conference proceedings in the areas of resource management in distributed and parallel systems, high performance cluster computing, mobile agent technology, and web semantics. He is the lead coauthor of the book *Load Balancing in Parallel Computers: Theory and Practice* (Kluwer Academic, 1997). He has served on the technical program committees of numerous international conferences, including IEEE HiPC '2002, IEEE ICDCS '2001, and IASTED PDCS '1999-2002. He will be a guest coeditor for a special issue on scalable web services and architecture for the *Journal of Parallel and Distributed Computing* in 2003. He is a recipient of the year 2000 Faculty Research Award at Wayne State University. He is a member of the ACM and a senior member of the IEEE.

Le Yi Wang received the PhD degree in electrical engineering from McGill University, Montreal, Canada, in 1990. Since 1990, he has been with Wayne State University, Detroit, Michigan, where he is currently a professor in the Department of Electrical and Computer Engineering. His research interests are in the areas of H-infinity optimization, complexity and information, robust control, time-varying systems, system identification, adaptive systems, hybrid and nonlinear systems, information processing and learning, as well as automotive, computer, and medical applications of control methodologies. Dr. Wang was awarded the Research Initiation Award in 1992 from the US National Science Foundation. He also received the Faculty Research Award from Wayne State University in 1992, and the College Outstanding Teaching Award from the College of Engineering, Wayne State University, in 1995. He was a keynote speaker at two international conferences. He serves on the IFAC Technical Committee on Modeling, Identification, and Signal Processing. He was an associate editor of the *IEEE Transactions on Automatic Control* and currently is an editor of the *Journal of System Sciences and Complexity* and an associate editor of the *International Journal of Control and Intelligent Systems*. He is a senior member of the IEEE.