Static Assignment of Complex Stochastic Tasks Using Stochastic Majorization

David Nicol Rahul Simha

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185
nicol@cs.wm.edu simha@cs.wm.edu

Don Towsley

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003
towsley@cs.umass.edu

Abstract

We consider the problem of statically assigning many tasks to a (smaller) system of homogeneous processors, where a task's structure is modeled as a branching process, and all tasks are assumed to have identical behavior. We show how the theory of majorization can be used to obtain a partial order among possible task assignments. This partial ordering is particularly useful when heterogeneous constraints are placed on the numbers of tasks that one may assign to the processors, or when the assignment problem is combinatorially complex. Our results show that if the vector of numbers of tasks assigned to each processor under one mapping is *majorized* by that of another mapping, then the former mapping is better than the latter with respect to a large number of objective functions. In particular, we show how measurements of finishing time, resource utilization, and reliability are all captured by the theory. We also examine applications of the theory to situations where tasks may exhibit different behavior, and to situations where the processors are heterogeneous.

1 Introduction

Parallel processing has emerged as an important means of achieving high computational performance. As a consequence, much research interest has been sparked in the area of efficient use of parallel computers. The problem of assigning tasks among processors to minimize processing time has already received considerable attention in the literature, e.g., [3, 4, 8, 9, 12, 19]. We consider the problem of statically assigning tasks to processors when the tasks have unknown random processing times and a certain type of stochastic structure. The structure we examine embodies the notion of one task spawning a set of others; we examine static assignments, under the assumption that all offspring of a task are executed on the same processor as the task. Static assignment is likely to be used when a task's state is large, thereby making dynamic assignment very costly in terms of communication.

This paper examines theoretical issues associated with comparing different static mappings of a set of complex stochastic tasks. In particular, we show how the theory of *majorization* can be used to derive strong results concerning the comparison of different mappings. The strength of our contribution lies in our providing a formal underpinning to the analysis of mapping complex stochastic tasks and to the optimization of a rich class of objective functions.

Previous work on load balancing or task assignment [3, 4, 7, 8, 9, 12, 19] in parallel systems may be loosely divided into three categories. The first category, with deterministic structure, involves task structures and execution times which are known prior to assignment. This case [14] includes a study of problem complexity under various constraints and heuristic algorithms for task scheduling. A second class of load balancing formulations, in which task execution times are random, is characterized by queueing-theoretic considerations [4, 16, 19]. Much of this work pertains to steady-state expectations of task delays with state-dependent [4, 19] and state-independent [16] assignment policies. Our work is closest to the third category [7, 8, 9, 13] which also takes task execution times to be random but focuses on minimizing expected processing times for a fixed set of tasks. As discussed in [9], the assumption of random execution times and a given set of tasks is justified in applications such as Monte-Carlo simulations.

Our approach to the problem differs from previous work [7, 8, 9, 13] in several ways. In this paper, we do not concern ourselves with the explicit optimization of task assignment, but rather, with the comparison between different assignments over a wide range of possible objective functions. Past approaches typically address the question: given K processors and m tasks with random execution requirements, find the assignment of tasks to processors that minimizes the expected maximum workload (or *makespan*). In this paper, we address a related question: given two assignments, when can we say that one is "better" than the other, and for what class of objective functions can we make this assertion? Our results have a simple general form. We can describe a mapping of probabilistically homogeneous tasks to processors by a vector $\boldsymbol{m}$, whose

ith component is the number of tasks assigned to the ith processor. Let $\boldsymbol{m}$ and $\boldsymbol{m}'$ describe two different mappings. Then if $\boldsymbol{m}$ can be bounded by $\boldsymbol{m}'$ using the notion of *majorization* [10] (written $\boldsymbol{m} \prec \boldsymbol{m}'$), then for all objective functions f in a class $\mathcal{C}$ we may say that the assignment described by $\boldsymbol{m}$ is better than the assignment described by $\boldsymbol{m}'$. The class $\mathcal{C}$ is often quite general, and includes many commonly used objective functions, e.g., the expected maximum workload. We note that an interest in inequalities or *stochastic orderings* can be more useful than merely searching for optimal assignments, because such orderings may be derived in a variety of cases where it is too expensive to search for an optimal assignment. Inequalities are also useful when constraints on the assignment (e.g. heterogeneous memory capacity among processors) prohibit one from adopting an otherwise obvious optimal policy. We note that stochastic orderings are of independent interest [15] and also, in some of the cases we consider, the optimal strategy is apparent from the derived ordering.

Our interest in obtaining stochastic orderings also stems from the observation that they are often the only results available for small numbers of random variables and a wide variety of distributions. Consider the fact that in [8, 9] the results are asymptotic in at least one variable m or K. In fact, in [9], the results are only asymptotically correct in both the number of tasks m and the number of processors K. These approaches are based on the use of the central limit theorem [8] and large deviation theory [9], which are among the few limit results available that hold for a variety of distributions. In contrast, our approach is concerned with finite (and possibly small) m and K and we make use of the theory of *stochastic majorization* [10]. Thus, while some of our results are not as strong (in terms of optimality) as those obtained from fundamental limit theorems, the accuracy of our results does not depend on the number of tasks or processors.

We now discuss other specific differences between our work and past efforts. Our structural model of a single task is that of a *finite branching process*: a completing task spawns a random number of subtasks. This type of behavior appears in diverse applications such as Branch-and-Bound searching algorithms [2] where the branching structure is obvious, and dynamic regridding algorithms in numerical computations [1] where sections of coarse grid serve as "tasks" which give rise to "subtasks" associated with finer grids. Furthermore, our results permit the analysis of much more complex objective functions than have typically been studied for stochastic task models. Our model differs significantly from those in [8, 9, 12]. The tasks in [9] were taken to be individual independent and identically distributed (i.i.d.) samples drawn from a common distribution, and synchronization behavior is that of periodic global synchronization. In [8] a complex task is comprised of a fixed number of tasks with random i.i.d. execution times. However, the analyses in both [9] and [8] are concerned with overheads (e.g. synchronization and communication costs) that our model does not include. In some ways the present work resembles earlier results obtained under the assumption that the workload assigned to a processor causes the processor to behave as a Markov chain [13]. Like this earlier work, our new results show how the quality of a static assignment persists across numerous stochastic transformations of the workload. The model we study in the present paper is a distinct improvement over that in [13], as the stochastic behavior of a processor is now explicitly dependent on the volume of workload it contains.

Other related research has been directed at computing the expected completion time for a *single* complex task with a possibly random acyclic structure [6, 17]. Another related publication [11] studies the problem of scheduling *sub-tasks* of a single task, where the sub-tasks form a tree. Lastly, an analytic study of load-balancing statistically homogeneous workload on a hypercube is presented in [7], where the mean and variance of the difference between the load on a processor and the average load are derived. While past research has been concerned exclusively with a single task or a given set of tasks, we also consider the joint assignment of *multiple classes* of tasks, where tasks in different classes have different probabilistic behaviors.

Our work is based on results from the study of stochastic majorization. The fundamental theory of majorization originates in the economic study of income distribution—a sort of "load" balancing. We believe majorization finds a natural application in the area of mapping parallel workload, and that one of our contributions is to demonstrate uses of this powerful theory in parallel processing. In this respect our work is similar to that in [3, 20, 18]. Application of majorization to network control is discussed in [18]. In [3] the focus is on a new stochastic ordering based on the class of symmetric, convex and L-subadditive functions with applications to routing and designing processor speeds. The load balancing emphasis in [3] is on scheduling structurally simple tasks from a queue. Majorization in steady-state queue lengths of open queueing networks is studied in [20], in which orderings are parameterized by queue utilizations. In contrast, we use the established orderings in [10] to obtain inequalities among all generations of complex tasks under different static mappings of the initial tasks.

The rest of this paper is organized as follows. In the next section, we define basic notation and present our workload model; also, we discuss the different stochastic orderings to be used throughout the paper. Section §3 contains the fundamental orderings for workloads. Section §4 discusses various objective functions of interest in parallel systems and Section §5 has some results on systems with multiple classes and processors with different speeds. Finally, we make some concluding remarks in Section §6.

2 Preliminaries

We now introduce our model of computation, important definitions and known results, and a rationale for using majorization to study the assignment problem.

2.1 Workload and System Model

We model the workload produced by a single task as a branching process [15, pp. 116-117]. At any point in time our workload consists of a number of *workload units* (WU's). Unless otherwise stated, we assume that a given WU may be executed with the same cost on any one of K homogeneous processors. For some of our results, a WU's execution cost may be random provided that all WU execution times are independent and identically distributed (i.i.d.).

We assume a processor executes one WU at a time. Upon completion of its execution a WU instantaneously spawns a new set of WU's, which are added to the processor's queue of pending WUs. Each WU belongs to some *generation*; an initial WU belongs to generation 1; the progeny of a generation q WU belong to generation $q+1$.

Consider the evolution of an initial WU. Let N_q denote the number of WUs in its q-th generation. We assume that the workload produced in the q-th generation is given as

$$N_q = \sum_{j=1}^{N_{q-1}} Z_{j,q}, \tag{1}$$

where $N_1 = 1$ and where $Z_{j,q}$ is the number of WUs generated by the j-th WU in the $(q-1)$-th generation. We assume that $\{Z_{j,q}, 1 \leq q \leq K\}_{j=1}^{\infty}$ is an independent and identically distributed (i.i.d.) sequence of random variables (r.v.'s) and that every WU is executed on the same processor as its parent. The following notation will be employed:

- K - the number of processors.
- c - the number of job classes. Each job in class i, $1 \leq i \leq c$, is a branching process. Every job in a class has the same branching distribution, at every generation.
- A multiclass assignment is described by a $c \times K$ matrix $\underline{\mathbf{m}}$ where the $(i,k)th$ element $m_{i,k}$ denotes the number of class i tasks assigned to processor k. When there is a single class we will use the notation $\mathbf{m}$ to describe the assignment. m_i is the ith component of $\mathbf{m}$.
- For single class problems, let $N_q(k, \mathbf{m})$ be the random number of generation q WUs spawned from initial WU's, on processor k under mapping vector $\mathbf{m}$. The random vector (across processors) of generation q WUs induced under $\mathbf{m}$ is denoted $\mathbf{N}_q(\mathbf{m})$.

The theory we develop permits us to compare different mappings under a variety of objective functions $f : \mathbb{R}^K \to \mathbb{R}$. Our results focus on comparing values of $E\left[f(\mathbf{N}_q(\mathbf{m}))\right]$ by deriving conditions for inequalities involving initial task assignments $\mathbf{m}$. Most of these are of the following form: given two assignments $\mathbf{m}$ and $\mathbf{m}'$ where $\mathbf{m} \prec \mathbf{m}'$, then $E\left[f(\mathbf{N}_q(\mathbf{m}))\right] \leq E\left[f(\mathbf{N}_q(\mathbf{m}'))\right]$ for all generations q. Functions f for which such results can be derived include the class of all symmetric[1] convex functions. This class contains the maximum operator and all powers of the maximum, the sum operator, and the product operator. Thus a single comparison between the assignment vectors $\mathbf{m}$ and $\mathbf{m}'$ vectors can yield a wealth of information about the comparative behaviors of complex stochastic tasks under the two mappings. Note that the maximum function or *makespan* ($\max_k N_q(k, \mathbf{m})$) is the most commonly used performance metric in the load balancing of parallel systems. We turn next to the definition of this $\prec$-ordering, *majorization*.

2.2 Stochastic Ordering and majorization

We now define the *majorization* partial ordering $\prec$; for convenience we use notation slightly different from that in [10].

Definition 2.1 (majorization) *A vector* $\mathbf{x}$ *is majorized by vector* $\mathbf{y}$, *written as* $\boldsymbol{x} \prec \boldsymbol{y}$, *iff*

1. $\sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]}$, $k = 1, \cdots, K-1$,
2. $\sum_{i=1}^{K} x_{[i]} = \sum_{i=1}^{K} y_{[i]}$.

where the notation $x_{[i]}$ *is taken to be the* i*-th largest element of* $\boldsymbol{x}$.

Definition 2.2 (Schur-convex function) *A function* $f : \mathbb{R}^K \to \mathbb{R}$ *is said to be Schur-convex if* $\boldsymbol{x} \prec \boldsymbol{y}$ *in* $\mathbb{R}^K$ *implies* $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ *in* $\mathbb{R}$.

Examples of Schur convex functions include $f(\boldsymbol{x}) = \max x_i$ and $f(\boldsymbol{x}) = \sum g(x_i)$, $\forall$ convex $g : \mathbb{R} \to \mathbb{R}$.

Let $\mathcal{C}_0$ be the class of increasing functions from $\mathbb{R}^K$ onto $\mathbb{R}$. The well-known stochastic ordering between random variables [15] is defined as follows. For random vectors $\mathbf{X}$ and $\mathbf{Y}$ with distribution functions F and G respectively,

$$\mathbf{X} \leq_{st} \mathbf{Y} \quad \text{iff} \int_{\mathbb{R}^K} f(\boldsymbol{x}) dF(\boldsymbol{x}) \leq \int_{\mathbb{R}^K} f(\boldsymbol{x}) dG(\mathbf{x}) \quad \forall f \in \mathcal{C}_0$$

such that the integrals are well defined. In this manner, majorization over deterministic quantities is extended to random variables by using an appropriate class of functions:

$$\begin{aligned} \mathcal{C}_1 &= \{sc\} &&= \{f : \mathbb{R}^K \to \mathbb{R} | f \text{ is Schur-convex } \}, \\ \mathcal{C}_2 &= \{cas\} &&= \{f : \mathbb{R}^K \to \mathbb{R} | f \text{ is convex and symmetric } \}. \end{aligned}$$

[1] f is symmetric if $f(\mathbf{X}) = f(\Pi\mathbf{X})$ for all permutations Π.

These define respectively the the *Schur convex* partial ordering, denoted by $\prec_{scx}$ and the *convex symmetric* partial ordering, denoted by $\prec_{cas}$ (the notation $\prec_{E_1}$ and $\prec_{E_2}$ is used in [10]). Thus, $\mathbf{X} \prec_{sc} \mathbf{Y}$ iff

$$\int_{\mathbb{R}^K} f(\boldsymbol{x})dF(\boldsymbol{x}) \leq \int_{\mathbb{R}^K} f(\boldsymbol{x})dG(\mathbf{x}) \quad \forall f \in \mathcal{C}_1$$

and $\mathbf{X} \prec_{cas} \mathbf{Y}$ iff

$$\int_{\mathbb{R}^K} f(\boldsymbol{x})dF(\boldsymbol{x}) \leq \int_{\mathbb{R}^K} f(\boldsymbol{x})dG(\mathbf{x}) \quad \forall f \in \mathcal{C}_2.$$

Note that $\mathcal{C}_2 \subset \mathcal{C}_1$ and thus, $\prec_{sc}$ is a stronger ordering than $\prec_{cas}$. Throughout this paper we will compare assignments by ordering their resulting random workloads; for example, for assignments $\boldsymbol{m}$ and $\boldsymbol{m}'$ which result in workloads $\mathbf{N}_q(\boldsymbol{m})$ and $\mathbf{N}_q(\boldsymbol{m}')$, we will determine conditions under which, for example, $\mathbf{N}_1(\boldsymbol{m}) \prec_{sc} \mathbf{N}_q(\boldsymbol{m}')$. Then, using the theory of majorization we conclude that for any Schur-convex functions f (such as the maximum function), $E\left[f(\mathbf{N}_1(\boldsymbol{m}))\right] \leq E\left[f(\mathbf{N}_q(\boldsymbol{m}'))\right]$. Next, we require the notion of a random vector with *exchangeable* components [10]:

Definition 2.3 (exchangeable components) *A random vector $\mathbf{X} = (X_1, \ldots, X_K)$ is said to have exchangeable components if the joint distribution of $X_1, \ldots, X_K$ is invariant under permutations of its components.*

Note that if two random variables X and Y have the same distribution we write $X =_{st} Y$. Some of our results require the following important result, referenced in [10] (page 283).

Lemma 2.1 (Strassen) *Two random vectors $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^K$ satisfy $\boldsymbol{X} \prec_{sc} \boldsymbol{Y}$ if and only if there exist two random vectors $\tilde{\boldsymbol{X}}$ and $\tilde{\boldsymbol{Y}}$ defined on a common probability space such that $\boldsymbol{X} =_{st} \tilde{\boldsymbol{X}}$, $\boldsymbol{Y} =_{st} \tilde{\boldsymbol{Y}}$, and $\tilde{\boldsymbol{X}} \prec \tilde{\boldsymbol{Y}}$ almost surely (a.s.).*

Other properties concerning these orderings can be found in [10], Chapter 11. Next we prove a result that we later use for $\prec_{cas}$ orderings.

Lemma 2.2 *Let X,Y be nonnegative random variables, and $\boldsymbol{Z} = (Z_1, Z_2)$ be a random vector with nonnegative exchangeable components. Assume that X, $\bar{X}$, Y, $\bar{Y}$ and $\boldsymbol{Z}$ are independent r.v.'s with $X =_{st} \bar{X}$ and $Y =_{st} \bar{Y}$. Define $\boldsymbol{U} = (X, Y)$ and $\boldsymbol{V} = (\bar{X} + \bar{Y}, 0)$. Then*

$$\boldsymbol{Z} + \boldsymbol{U} \prec_{cas} \boldsymbol{Z} + \boldsymbol{V}.$$

Proof. Let $\phi : \mathbb{R}_+^2 \to \mathbb{R}$ be a convex symmetric function. Define the function $\psi : \mathbb{R}_+^2 \to \mathbb{R}$ as $\psi(\boldsymbol{a}) = E[\phi(\boldsymbol{Z} + \boldsymbol{a})]$ $\forall \boldsymbol{a} \in \mathbb{R}_+^2$. Since $\boldsymbol{Z}$ has exchangeable components, ψ is also a convex symmetric function.

Define $\boldsymbol{V}' = (X+Y, 0)$. Now $\boldsymbol{U} \prec \boldsymbol{V}'$ a.s. from which it follows

$$\begin{aligned} \psi(\boldsymbol{U}) \leq \psi(\boldsymbol{V}') &\Rightarrow E[\psi(\boldsymbol{U})] \leq E[\psi(\boldsymbol{V}')], \\ &\Rightarrow E[\phi(\boldsymbol{Z}+\boldsymbol{V}')] \leq E[\phi(\boldsymbol{Z}+\boldsymbol{V}')], \\ &\Rightarrow \boldsymbol{Z}+\boldsymbol{U} \prec_{cas} \boldsymbol{Z}+\boldsymbol{V}'. \end{aligned}$$

The result follows by noting that $E[\psi(\boldsymbol{V}')] = E[\psi(\boldsymbol{V})]$. ∎

Corollary 2.1 *Let X,$\bar{X}$,Y and $\bar{Y}$ be nonnegative random variables such that $X =_{st} \bar{X}$ and $Y =_{st} \bar{Y}$; let $\boldsymbol{Z} = (Z_1, Z_2, \cdots, Z_n) \in (\mathbb{R}_+^K$ be a r.v. with exchangeable components. Assume that X,$\bar{X}$,Y,$\bar{Y}$ and $\boldsymbol{Z}$ are mutually independent and define $\boldsymbol{U} = (X, Y, 0, \cdots, 0)$ and $\boldsymbol{V} = (\bar{X}+\bar{Y}, 0, \cdots, 0)$. Then*

$$\boldsymbol{Z}+\boldsymbol{U} \prec_{cas} \boldsymbol{Z}+\boldsymbol{V}.$$

Proof: Let $f : \mathbb{R}^K \to \mathbb{R}$ be a symmetric convex function. Now, f is symmetric and convex in the first two arguments. Therefore, we can condition the values of Z_j to be z_j, $3 \leq j \leq K$, and apply the previous lemma to obtain

$$E_{X,Y,Z_1,Z_2}[f(\boldsymbol{U})|Z_3 = z_3, \cdots, Z_K = z_K] \leq E_{X,Y,Z_1,Z_2}[f(\boldsymbol{V})|Z_3 = z_3, \cdots, Z_K = z_K]$$

Removal of the conditioning on Z_j, $3 \leq j \leq K$ yields the desired result. ∎

Before leaving this section we consider the ordering $\leq_{st}$ in the case $K = 1$. First, if X and Y are real valued r.v.'s, then

$$X \leq_{st} Y \quad \text{iff} \quad \Pr[X > t] \leq \Pr[Y > t], \quad \forall t.$$

A necessary and sufficient condition for $X \leq_{st} Y$ is that there exist r.v.'s $\tilde{X}$ and $\tilde{Y}$ with the same distributions as X and Y respectively, such that $\tilde{X} \leq \tilde{Y}$ [15][page 255].

Finally, we state a few results about *likelihood-ratio* orderings which we will use later. Consider non-negative integer valued r.v.'s X and Y with probability mass functions f and g.

Definition 2.4 (likelihood ratio) *We say that X is smaller than Y in likelihood ratio, written as $X \leq_{lr} Y$, iff*

$$\frac{g(n)}{f(n)} \leq \frac{g(m)}{f(m)}, \quad 0 \leq n \leq m, \quad n, m \in \mathbb{N}.$$

This definition is used to characterize a class of distribution functions in the following manner.

Definition 2.5 (ILR) *The non-negative integer valued r.v. X is said to have increasing likelihood ratio (ILR) iff*

$$c_1 + X \leq_{lr} c_2 + X, \quad 0 \leq c_1 \leq c_2.$$

Last, we define a *totally positive convolution*:

Definition 2.6 (totally positive convolution) *Let X be a discrete non-negative r.v. with probability distribution F and let X_j denote the j-fold convolution of X (with distribution F_j). If*

$$F_i(s)F_j(t) \geq F_i(t)F_j(s) \qquad \text{whenever } i < j \text{ and } s < t. \tag{2}$$

then the convolutions X_i are said to be totally positive (our definition is derived from the more general definition of totally positive functions – see [10][page 487]).

Our results use two properties of totally positive convolutions, both of which are consequences of Theorem 18.A.4.a in [10]. We first state a discrete version of this useful theorem and then describe how they are applied to compounded totally positive convolutions.

Theorem 2.1 *Let $\beta(\gamma, y)$ and $\alpha(\theta, x)$ be the probability mass functions of two totally positive convolutions, the first a γ-fold convolution and the second a θ-fold convolution. Then*

$$\delta(\theta, y) = \sum_{\gamma} \beta(\gamma, y)\alpha(\theta, \gamma)$$

is a probability mass function of a totally positive (θ-fold) convolution.

Proof: Theorem 18.A.4.a in [10] implies that $\delta(\theta, y)$ is totally positive. That it is a probability mass function is a consequence of appropriately interpreting the arguments of the functions so that $\delta(\theta, y)$ is either a convolution or a compounding.

Lemma 2.3 (compounding) *Let W_θ and X_γ be two discrete totally positive convolutions. Then the compounded r.v. $Z_\theta = X_{W_\theta}$ is also a totally positive convolution.*

Proof: Let $\alpha(\theta, z) = P[W_\theta = z]$ and $\beta(\gamma, y) = P[X_\gamma = y]$. Then,

$$\begin{aligned}
\delta(\theta, y) &= P[Z_\theta = y] \\
&= \sum_{\gamma} P[X_\gamma = y | W_\theta = \gamma] P[W_\theta = \gamma] \\
&= \sum_{\gamma} \beta(\gamma, y)\alpha(\theta, \gamma)
\end{aligned}$$

which by Theorem 2.1 is the probability mass function of a totally positive convolution – Z_θ. ∎

Lemma 2.4 *Suppose Z_θ is a totally positive convolution. Then $Y_\theta = Z_\theta + \theta$ is also a totally positive convolution.*

Proof: Let $\alpha(\theta, y) = P[Z_\theta = y]$ and $\nu(\theta, z) = P[Y_\theta = z]$. Then, $\nu(\theta, z) = \alpha(\theta, z - \theta)$. Now Y_θ will be totally positive if the following is satisfied:

$$\nu(\theta_1, z_1)\nu(\theta_2, z_2) \geq \nu(\theta_1, z_2)\nu(\theta_2, z_1)$$

whenever $\theta_1 < \theta_2$ and $z_1 < z_2$. This will be true if

$$\alpha(\theta_1, z_1 - \theta_1)\alpha(\theta_2, z_2 - \theta_2) \geq \alpha(\theta_1, z_2 - \theta_2)\alpha(\theta_2, z_1 - \theta_1)$$

and both $\theta_1 < \theta_2$ and $z_1 - \theta_1 < z_2 - \theta_2$. **I'm not sure if the last bit is always true**.

The following properties [10, 15] will be useful:

- if $X, Y \in \mathbb{Z}$ are independent r.v.'s both having increasing likelihood ratio, then $X + Y$ has increasing likelihood ratio, and
- if $X, Y \in \mathbb{Z}$, $X \leq_{lr} Y$, then $X \leq_{st} Y$.

3 Branching and stochastic majorization

In this section we establish conditions under which either $\prec_{cas}$ or $\prec_{sc}$ orderings can be established between the workload vectors of the k-th generation of two initial workload vectors. We begin with the $\prec_{sc}$ ordering. Our results for the $\prec_{sc}$ ordering is based on the following theorem [10][J.2, page 101], which we state for discrete non-negative r.v.'s.

Theorem 3.1 (Proschan and Sethuraman) *Let X be a discrete non-negative r.v. with probability distribution F and let X_i denote the i-fold convolution of X (with distribution F_i). Assume that each X_i is totally positive. Let $\boldsymbol{m} = (m_1, \ldots, m_k)$ be a vector of nonnegative integers, and let $f : \mathbb{R}^K \to \mathbb{R}$ be a Schur-convex function. Then*

$$\begin{aligned} \gamma(\boldsymbol{m}) &= \sum_{\boldsymbol{x} \in \mathbb{Z}^K} f(\boldsymbol{x}) \prod_{1 \leq i \leq K} F_{m_i}(x_i) \\ &\equiv E\left[f(X_{m_1}, \ldots, X_{m_K})\right] \end{aligned}$$

is a Schur-convex function in $\boldsymbol{m}$.

We now determine a sufficient condition for totally positive convolutions.

Lemma 3.1 *If F has increasing likelihood ratio, Then convolutions of F satisfy equation (2), i.e., are totally positive.*

Proof: The proof is based on the fact that the convolution of two ILR r.v.'s is also an ILR r.v. Consider integers $i < j$. We may write $X_j = X_{j-i} + X_i$. Since $X_i \geq 0$ and X_i is ILR for each i, we may take $c_1 = 0$ and $c_2 = X_{j-i}$ in definition 2.5 to get

$$X_i \leq_{lr} X_i + X_{j-i} = X_j$$

Next, we substitute X_i and X_j in the definition of likelihood ratio ordering:

$$\frac{P\left[X_j = s\right]}{P\left[X_i = s\right]} \leq \frac{P\left[X_j = t\right]}{P\left[X_i = t\right]}, \quad s < t,$$

which gives us the desired result, equation (2). ∎

Thus, a sufficient condition for convolutions to be totally positive is that the starting random variable be ILR. This is true of the Normal distribution, Uniform, Poisson, Geometric and Binomial distributions. and certain variants of the Gamma, Beta, and Weibull distributions [10]. Using Theorem 3.1 we obtain our $\prec_{sc}$ ordering result.

Theorem 3.2 *Suppose a WU's workload is described as an independent sample from a nonnegative ILR probability distribution. Let $\boldsymbol{m}$,$\boldsymbol{m}'$ be two mapping vectors such that $\boldsymbol{m} \prec \boldsymbol{m}'$. Then the random vector of processor workloads satisfy $\prec_{sc}$:*

$$\boldsymbol{W}(\boldsymbol{m}) \prec_{sc} \boldsymbol{W}(\boldsymbol{m}').$$

Note that we have used $\mathbf{W}(\mathbf{m})$ to denote the random workload generated by assignment $\mathbf{m}$. Usually, this will be proportional to $\mathbf{N}_q(\mathbf{m})$; however, we use $\mathbf{W}(\mathbf{m})$ to emphasize the generality of Theorem 3.2. The above theorem provides us with the $\prec_{sc}$ ordering for the first generation workload vector. In order to obtain a similar result for the k-th generation workload vector, we make use of a closure property of $\prec_{sc}$ undr compounding. Let $\mathbf{X}_\theta$ be a random vector with distribution function $F^{(\theta)}$ parameterized by the vector θ; in our case we are interested in the workload distribution where θ is the mapping vector. We say that the family $\{F^{(\theta)}\}$ of distributions is parameterized to preserve $\prec_{sc}$ if $\theta \prec \theta'$ implies that $\mathbf{X}_\theta \prec_{sc} \mathbf{X}_{\theta'}$. A useful result concerning compounding is ([10], page 285):

Lemma 3.2 ([10], page 285) *If $\{F^{(\theta)}, \theta \in A\}$ is a family of distributions parameterized to preserve $\prec_{sc}$ and if θ is treated as a random variable with distribution function $G^{(\gamma)}$ which is also from a family $\{G^{(\gamma)}, \gamma \in B\}$ parameterized to preserve $\prec_{sc}$, then the family of compound distribution functions $\{H^{(\gamma)}, \gamma \in B\}$,*

$$H^{(\gamma)}(\boldsymbol{x}) = \int F^{(\theta)}(\boldsymbol{x}) dG^{(\gamma)}(\theta)$$

preserves $\prec_{sc}$ in γ.

Applied to our problem, the random vector θ describes the number of generation $q-1$ WUs generated on each processor. Conditioned on this vector, $F^{(\theta)}$ is the joint distribution of generation q WU's generated on each processor. Furthermore, we know that this joint distribution preserves $\prec_{sc}$ in θ (as was seen for the first generation). We can therefore apply Lemma 3.2 in an induction argument to prove that for all q, $\boldsymbol{N}_q(\boldsymbol{m}) \prec_{sc} \boldsymbol{N}_q(\boldsymbol{m}')$ whenever $\boldsymbol{m} \prec \boldsymbol{m}'$.

Theorem 3.3 *Consider a set of branching WUs, all with common branching distribution F. Suppose that F is ILR. For any mapping vector $\boldsymbol{m}$, let $\boldsymbol{N}_q(\boldsymbol{m})$ denote the vector of generation q WU's on each processor. Then for any $q \in \mathbb{N}$*

$$\boldsymbol{N}_q(\boldsymbol{m}) \prec_{sc} \boldsymbol{N}_q(\boldsymbol{m}') \qquad \text{whenever } \boldsymbol{m} \prec \boldsymbol{m}'.$$

We now consider the $\prec_{cas}$ ordering. In this case, we are able to consider more general branching processes and more general workload vector constructions. First, we focus on the workload vectors associated with the first generation. Let $\boldsymbol{m}'$ be any mapping vector where there are processors i, j such that $m'_i > m'_j$; let $\boldsymbol{W}(\boldsymbol{m}')$ be the random workload vector that $\boldsymbol{m}'$ induces.

Lemma 3.3 *Suppose a WU's workload is described as an independent sample from a nonnegative probability distribution. Let $\boldsymbol{m}$,$\boldsymbol{m}'$ be two mapping vectors such that $\boldsymbol{m} \prec \boldsymbol{m}'$. Then the random vector of processor workloads satisfy:*

$$\boldsymbol{W}(\boldsymbol{m}) \prec_{cas} \boldsymbol{W}(\boldsymbol{m}').$$

Proof. If $\boldsymbol{m} \prec \boldsymbol{m}'$ then Lemma 2.B.1 in [10] implies the existence of vectors $\mathbf{u}_1 = \boldsymbol{m}, \mathbf{u}_2, \ldots, \mathbf{u}_n = \boldsymbol{m}'$ such that $\mathbf{u}_1 \prec \ldots \prec \mathbf{u}_n$ and, $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ differ in only two components. Without loss of generality we will assume that $\boldsymbol{m}$ and $\boldsymbol{m}'$ differ only in the first two components and, further, that $m_1 \geq m_2$ and $m'_1 \geq m'_2$. Thus, $\boldsymbol{m} \prec \boldsymbol{m}'$ implies $(m_1, m_2) \prec (m'_1, m'_2)$. Next, let $W(m_i)$ be the workload due to m_i initial WU's. Define independent r.v.'s Z_1, Z_2, X and Y such that

$$Z_1 =_{st} Z_2 =_{st} W(m'_2)$$

and

$$\begin{aligned} X &= W(m_1 - m'_2) \\ Y &= W(m_2 - m'_2) \\ \bar{X} + \bar{Y} &= W(m'_1 - m'_2). \end{aligned}$$

Note that $X + Y =_{st} \bar{X} + \bar{Y}$. The application of corollary 2.1 yields $\boldsymbol{W}(\boldsymbol{m}) \prec_{cas} \boldsymbol{W}(\boldsymbol{m}')$. ∎

This result generalizes readily to the k-th generation by simply treating it as the first generation of an entirely different branching process. A more general result is the following.

Theorem 3.4 *Consider a set of branching WUs, all with common branching distribution F. For any mapping vector $\boldsymbol{m}$, let $\boldsymbol{N}_k(\boldsymbol{m})$ denote the vector of generation k WU's on each processor. Then for any set of integers A,*

$$\sum_{q \in A} \boldsymbol{N}_q(\boldsymbol{m}) \prec_{cas} \sum_{q \in A} \boldsymbol{N}_q(\boldsymbol{m}') \qquad \textit{whenever } \boldsymbol{m} \prec \boldsymbol{m}',$$

where $\sum$ denotes component-wise addition.

Proof. The q-th generation workloads spawned by all of the original WU's are i.i.d. r.v.'s. Furthermore, the sum of the workloads in the generations contained in A spawned by each of the original WU's form an i.i.d. set of r.v.'s. One can define a new branching process for which these workload sums can be treated as the first generation offspring for the original WU's. Consequently, lemma 3.3 applies to yield the desired result. ∎

The main two results of these section show that stochastic branching preserves stochastic majorization in the branching generations. The section to follow illustrates how these results can be fruitfully applied.

4 Objective Functions

We will now establish that a number of interesting objective functions are either Schur convex or convex symmetric functions of the workload vector. These objective functions include finishing time under different synchronization schemes, the space-time product, and overall reliability. This diversity of application demonstrates the utility of the theory.

4.1 Finishing Time

One use of majorization is to show that whenever $\boldsymbol{m} \prec \boldsymbol{m}'$, the computation's expected finishing time under $\boldsymbol{m}$ is better than that under $\boldsymbol{m}'$. This type of result follows immediately from Theorem 3.4. For example, envision a computation where the tasks must periodically synchronize, say, after every generation (this is typical of tasks associated with numerical computations). If the WUs each have unit execution time, then $\max_k\{\boldsymbol{N}_q(k, \boldsymbol{m})\}$ time is required to complete all generation q WUs spawned by $\boldsymbol{m}$. The max operator is convex and symmetric, so the expected time between each pair of synchronizations is better under $\boldsymbol{m}$ than under $\boldsymbol{m}'$. This same result holds true if the WU execution times are random, and i.i.d. . Furthermore, the standard deviation of a vector's components is a Schur-convex function [10][page 71], showing that if the branching distribution is ILR the variation in "time-to-synchronization" is smaller under $\boldsymbol{m}$ than under $\boldsymbol{m}'$. More precisely, if

$$f(\mathbf{N}_q(\mathbf{m})) \;=\; \sum_{k=1}^{K} \left(N_q(k,\mathbf{m}) - \bar{N}_q(\mathbf{m})\right)^2$$

where

$$\bar{N}_q(\mathbf{m}) = \frac{1}{K}\sum_{k=1}^{K} N_q(k,\mathbf{m})$$

then $\boldsymbol{m} \prec \boldsymbol{m}'$ implies $E\left[f(\mathbf{N}_q(\mathbf{m}))\right] \leq E\left[f(\mathbf{N}_q(\boldsymbol{m}'))\right]$.

One can also treat the situation where work units descended from different initial WUs have no need to explicitly synchronize with each other. Once a processor is loaded with its initial WUs it is free to execute those work units and all their progeny, with a single global synchronization to terminate the set of parallel tasks. Since we have assumed that each task is a finite branching process, the number of generations is bounded, i.e., $\exists M : \forall q > M \; N_q(k,\mathbf{m}) = 0 \; \forall k$. Thus we may take $A = \{1,\ldots,M\}$ in Theorem 3.4 to obtain a $\prec_{cas}$ ordering. The ith component of $\sum_{q\in A} \mathbf{N}_q(\boldsymbol{m})$ is then the total number of WUs processed at processor i, and is hence its finishing time.

4.2 Functions of Queue Length

In our model, when a WU completes its execution it generates its children and places them on the processor's queue. Following this, another WU is selected to be executed. There is thus a storage cost associated with executing complex tasks; more generally, we show here how stochastic majorization can be applied to objective functions based on measuring queue lengths at every time step. A simple example of this is the computation's total space-time product, when processors are free to execute their WUs without synchronization. We will show that if $\boldsymbol{m} \prec \boldsymbol{m}'$, then under any branching distribution the expected space-time product under $\boldsymbol{m}$ is no greater than that under $\boldsymbol{m}'$. More general results are possible if the branching distribution is ILR: given any increasing convex cost function of queue length, the expected cost under $\boldsymbol{m}$ is no greater than that under $\boldsymbol{m}'$.

Under the model assumptions we have made, the probabilistic behavior of a processor's queue is completely independent of the queueing discipline used. We will assume that the queueing discipline is Smallest-Generation-First (SGF): no unit of workload from generation $q+1$ is executed before any unit from generation q. We also assume that the execution of a WU takes unit time.

Let $s : \mathbb{R} \rightarrow \mathbb{R}$ be the identity function. If a processor has k WUs in queue at time t, we will ascribe cost $s(k) = k$ to that processor, at that time. The computation's space-time product is the sum of this cost over all time-steps and all processors (defined more precisely below). Our analysis of this cost hinges on the following observation: if processor i has exactly $N_q(i, \boldsymbol{m})$ WU units in generation q, then under the SGF queueing discipline *at some point in time the queue will have exactly $N_q(i, \boldsymbol{m})$ WUs.* In particular, at the point where the first WU of generation q is served the queue consists entirely of generation q WUs, and contains all of them. We will then analyze the expected behavior of the space-time product over the interval of time during which the generation q WUs are processed.

Suppose $N_q(i, \boldsymbol{m}) = r$. The processing of the kth WU in generation q ($k = 1, \ldots, r$) produces a random number X_k of WU units, who join the queue. The random queue length observed during the processing of the kth WU is $r - (k-1) + \sum_{j=1}^{k-1} X_j$, with expectation $r + (k-1)(E[X] - 1)$. The expected space-time product in this processor, over the interval of time when generation q WUs are executed, is $r^2 + r(r-1)(E[X]-1)/2$. Let us now concurrently measure the expected space-time products of all processors when they process generation q WUs (of course, there is no assumption or requirement that the processors be executing generation q WUs at the same time, the concurrency here is purely mathematical) Define the function $T : \mathbb{R}^K \rightarrow \mathbb{R}$ by

$$T(r_1, \ldots, r_K) = \sum_{i=1}^{K} r_i^2 + \frac{r_i(r_i+1)(E[X]-1)}{2}.$$

Observe that T is convex over $\mathbb{R}^r$ (its Hessian is positive definite); thus, since $N_q(\boldsymbol{m})$ preserves the $\prec_{cas}$ ordering, we know that

$$E[T(\mathbf{N}_q(\boldsymbol{m}))] \leq E[T(\mathbf{N}_q(\boldsymbol{m}'))] \qquad \text{whenever } \boldsymbol{m} \prec \boldsymbol{m}'.$$

The computation's total expected space-time product is the sum over all generations of this expectation. Thus the computation achieves a better expected space-time cost under $\boldsymbol{m}$ than under $\boldsymbol{m}'$. This result is true given any nonnegative branching distribution whose mean is less than 1 and whose second moment is finite (a condition which ensures the existence of the expectation). Thus we have

Theorem 4.1 *Let $\mathcal{S}(\boldsymbol{m})$ be the space-time product under mapping vector $\boldsymbol{m}$, the function which charges unit cost for each WU in queue, each time period. Then under any branching distribution with mean less than 1, and finite second moment*

$$E[\mathcal{S}(\boldsymbol{m})] \leq E[\mathcal{S}(\boldsymbol{m}')] \qquad \textit{whenever } \boldsymbol{m} \prec \boldsymbol{m}'.$$

Other space-usage functions are also intuitive. For example, one might have to store WU states on disk whenever the queue length exceeds a threshold L; furthermore, once L is exceeded the cost might be superlinear, owing to fragmentation costs. A candidate cost function is

$$s(k) = \begin{cases} 0 & \text{if } k \leq L \\ (L-k)^{1+\epsilon} & \text{if } k > L \end{cases}$$

where $\epsilon \geq 0$. In order to deal with functions such as this, we will have to use stronger results associated with ILR branching distributions and the $\prec_{sc}$ ordering.

Suppose that $s(k)$ is increasing and convex in k. Following the same argument used with the space-time product, the expected cost of processing the kth WU in generation q, given $N_q(i, \boldsymbol{m}) = r$ is

$$\phi(k, r) = E[s(r - (k-1) + \sum_{j=1}^{k-1} X_j)].$$

Here ϕ is convex in r, because for any convex γ and random variable Z, the expectation $E[\gamma(a+Z)]$ is convex in a (assuming the expectation exists). Now the expected space cost of processing all r members of generation q is

$$C(r) = \sum_{k=1}^{r} \phi(k, r).$$

$C(r)$ is a convex function of r. To demonstrate this it suffices to show that $C(r+2)+C(r) \geq 2C(r+1)$ for all n. Since ϕ is convex in r we have $\phi(k, r+2) + \phi(k, r) \geq 2\phi(k, r+1)$ for all $k = 1, \ldots r$. This observation reduces the problem to a demonstration of

$$\phi(r+2, r+2) + \phi(r+1, r+2) \geq 2\phi(r+1, r+1).$$

The increasing nature of $s(r)$ establishes that both $\phi(r+2, r+2)$ and $\phi(r+1, r+2)$ dominate $\phi(r+1, r+1)$, thereby proving the convexity of $C(r)$.

For any convex function g, the function $T(r_1, \ldots, r_n) = \sum_{i=1}^{n} C(r_i)$ is Schur-convex. Having shown that $C(r)$ is convex we may conclude from the $\prec_{sc}$ ordering (if $\mathbf{N}_q(\mathbf{m}) \prec_{sc} \mathbf{N}_q(\boldsymbol{m}'))$ that

$$E[T(\mathbf{N}_q(\boldsymbol{m}))] \leq E[T(\mathbf{N}_q(\boldsymbol{m}'))] \qquad \text{whenever } \boldsymbol{m} \prec \boldsymbol{m}'.$$

As with the space-time product, the overall expected space-cost is obtained by summing over all generations.

4.3 Reliability

Yet another application of majorization is to the question of whether the hardware will successfully execute the entire computation. We suppose that the computation "fails" if any processor having a non-empty queue fails. Observe that this definition permits the computation to successfully complete even if a processor dies before the entire computation is finished, provided the failing processor is itself already finished. We will show that if the branching distribution is ILR and each processor has an increasing hazard rate function, then the probability of failure under $\boldsymbol{m}$ is no greater than that under $\boldsymbol{m}'$, whenever $\boldsymbol{m} \prec \boldsymbol{m}'$. Conversely, if the branching distribution is ILR and the processor failure distribution has a decreasing hazard rate function, then the reliability under $\boldsymbol{m}'$ is better than that under $\boldsymbol{m}$.

We may describe the behavior of the system under $\boldsymbol{m}$ as a random matrix $\underline{\mathbf{M}}(\boldsymbol{m})$ whose qth ($q = 1, 2, \ldots,$) row is the generation vector $\mathbf{N}_q(\boldsymbol{m})$. For every $q = 1, 2, \ldots,$ let $\underline{\mathbf{M}}_q(\boldsymbol{m})$ be the submatrix consisting of the first q rows of $\underline{\mathbf{M}}(\boldsymbol{m})$. Particular sample paths of the computation are described by appropriately sampling the components of $\underline{\mathbf{M}}(\boldsymbol{m})$.

Majorization has been extended to matrices in a number of ways, including the following. We say that matrix X is dominated by matrix Y (X and Y having the same dimensions) by the $\prec^{row}$ ordering if every row of X is majorized by the corresponding row of Y. It turns out that when the branching distribution is ILR we can use $\prec^{row}$ to dominate each sample path matrix under $\boldsymbol{m}$ by a unique equiprobable sample path under $\boldsymbol{m}'$, as we now demonstrate.

$\mathbf{N}_1(\boldsymbol{m}) = \boldsymbol{m} \prec \boldsymbol{m}' = \mathbf{N}_1(\boldsymbol{m}')$, showing that the first row of any sample path matrix under $\boldsymbol{m}$ is majorized by the first row under $\boldsymbol{m}'$. Since the branching distribution is ILR, we know that the joint distribution of the next generation under $\boldsymbol{m}$ is dominated via $\prec_{sc}$ by the next generation under $\boldsymbol{m}'$. By Strassen's lemma (Lemma 2.1), we may randomly choose second generation vectors $\mathbf{p} = (p_1, \ldots, p_K)$ and $\mathbf{q} = (q_1, \ldots, q_K)$ such that $\mathbf{p}$ has the distribution of $\mathbf{N}_2(\boldsymbol{m})$, $\mathbf{q}$ has the distribution of $\mathbf{N}_2(\boldsymbol{m}')$, and $\mathbf{p} \prec \mathbf{q}$. This implies that for every sample path matrix of the computation under $\boldsymbol{m}$ we can construct a unique equiprobable sample path for matrix $\boldsymbol{m}'$, such that the second matrix dominates the first by $\prec^{row}$. The utility of this domination will be seen by application to the following result, cited in [10]. **pages?**

Lemma 4.1 (Rinott) *A differentiable function $\phi : \mathbb{R}^{mn} \to \mathbb{R}$ satisfies*

$$\phi(X) \leq \phi(Y) \qquad \textit{for all } X \prec^{row} Y$$

if and only if ϕ is Schur-convex in each row, with all other rows being fixed.

Thus, for objective functions ϕ which satisfy these conditions, we will be able to show that $\boldsymbol{m}'$ dominates $\boldsymbol{m}$ on a sample path basis. We next show that the computation's probability of failure is such a function.

Suppose that processor i's time to failure is the random variable Z_i, with an monotone hazard rate function $\lambda(t)$. It is well known that

$$\Pr\{Z > t\} = \exp\{-\int_0^t \lambda(s)\ ds\}.$$

If $\lambda(t)$ is nondecreasing, then $-\int_0^t \lambda(s)\ ds$ is concave, which is to say that $\log \Pr\{Z > t\}$ is concave. Conversely, if $\lambda(s)$ is decreasing, then $\log \Pr\{Z > t\}$ is convex.

It follows (3.B.1.d in [10]) that when $\lambda(s)$ increases, the product

$$S(t_1, \ldots, t_K) = \prod_{i=1}^{K} \Pr\{Z_i > t_i\}$$

is Schur-concave, or equivalently, that $-S(t_1, \ldots, t_K)$ is Schur-convex. When $\lambda(s)$ decreases then $S(t_1, \ldots, t_K)$ is Schur-convex.

Let $\underline{\mathbf{M}}_q(\boldsymbol{m})$ be any sample path matrix processor counts of the first q generations under $\boldsymbol{m}$, and choose $\underline{\mathbf{M}}_q(\boldsymbol{m}')$ so that $\underline{\mathbf{M}}_q(\boldsymbol{m}) \prec^{row} \underline{\mathbf{M}}_q(\boldsymbol{m}')$. Let $M_{i,j}(\boldsymbol{m})$ and $M_{i,j}(\boldsymbol{m}')$ denote the ijth entries of $\underline{\mathbf{M}}_q(\boldsymbol{m})$ and $\underline{\mathbf{M}}_q(\boldsymbol{m}')$ respectively. For any $q \times K$ matrix $\underline{\mathbf{M}}_q$ define

$$\phi(\underline{\mathbf{M}}_q) = \prod_{i=1}^{K} \exp\{-\int_0^{\sum_{j=1}^{q} M_{i,j}} \lambda(s)\ ds\}.$$

For a given partial sample path $\underline{\mathbf{M}}_q$, $\phi(\underline{\mathbf{M}}_q)$ is the probability that the system lives long enough to process all generations up through q (assuming SGF queueing). Choose any row r, and fix all entries of all other rows. For each column i let $C_{r,i}$ be the sum of all entries in column i from rows other than r. Conditioned on the values of all rows other than r, the reliability as a function of row r is

$$\phi_M(M_{r,1}, \ldots, M_{r,K}) = \prod_{i=1}^{K} \exp\{-\int_0^{C_{r,i}+M_{r,j}} \lambda(s)\ ds\}.$$

As we have seen, if $\lambda(s)$ is increasing then this function is Schur-concave, which is equivalent to saying that its negation is Schur-convex. If $\lambda(s)$ is decreasing then the function is Schur-convex. Thus, when $\lambda(s)$ increases the function $-\phi$ satisfies the conditions of Lemma 4.1, demonstrating that $\phi(\underline{\mathbf{M}}_q(\boldsymbol{m})) \geq \phi(\underline{\mathbf{M}}_q(\boldsymbol{m}'))$. The converse is true when $\lambda(s)$ decreases. The probability that the system completes without failure is found by taking the limiting value of this function as $q \to \infty$.

Theorem 4.2 *Suppose the time to failure in each processor is independent and identically distributed with a monotone hazard rate function. If $\boldsymbol{m} \prec \boldsymbol{m}'$ and the hazard rate increases, then the probability of completing the computation without failure is no greater under $\boldsymbol{m}'$ than under $\boldsymbol{m}$. If the hazard rate decreases the probability of failure under $\boldsymbol{m}'$ is no greater than that under $\boldsymbol{m}$.*

5 Heterogeneity

All of our results so far have concerned probabilistically homogeneous tasks, and homogeneous processors. The theory of majorization requires symmetry in the objective function or components of random vectors, a requirement usually violated when heterogeneity is introduced. In this final section we consider two situations exhibiting heterogeneity: that of heterogeneous classes of WUs, and that of heterogeneous processors.

We consider two types of heterogeneous computations; one has special structure, while the other is general. In the general case we permit a WU to belong to one of c classes; each class has a unique branching distribution. We will illustrate the difficulties faced in assigning such heterogeneous tasks, but then show how majorization can be used to improve certain types of assignments.

In the second case we consider tasks which can be viewed as compositions of the uniform WUs we have already studied. Such tasks are comprised of a sequence of stages, each with random nonnegative integer weight. The stage weights behave as a Markov chain, as follows. If the weight of stage i is w_i, then the weight of stage $i+1$ has distribution F_{w_i}—an w_i-fold convolution of some common distribution F. Of course, F must be integer and nonnegative. This task structure has the property that a task with initial weight w can be viewed as the convolution of w uniform branching WUs. The weight of stage j has the distribution of the sum of all generation j WUs descended from a set of w initial WUs. Because of this correspondence we are able to obtain majorizations based on the stage 1 weights of the set of heterogeneous tasks. The result is significant; whereas the optimal assignment of probabilistically uniform WUs in self-evident, the problem of finding the optimal assignment of these heterogeneous tasks is NP-Complete.

Finally, we consider the assignment of WUs with a (common) ILR branching distribution to a multiprocessor system having heterogeneous processor speeds. We prove the intuitive result that given any partitioning of the WUs, the optimal mapping is achieved by iteratively assigning the largest unassigned partition group of WUs to the fastest unassigned processor.

5.1 General Multiclass Problem

Suppose that each WU belongs to one of c classes. Each class is characterized by the branching distribution of WUs in that class. The assignment problem under this assumption is much more difficult than when WU workloads are identically distributed. Consider: if μ_B is the mean of a WU's branching distribution, then μ_B^{q-1} is the mean number of generation q descendents. Under our previous assumptions when k_1 WUs are placed on processor 1 and k_2 WUs are placed on processor 2, the ratio $E^r[N_1(1,\boldsymbol{m})]/E^r[N_1(2,\boldsymbol{m})]$ is the constant k_1/k_2 for all q and all r. This gives us a sense of how the balance in the first generation is preserved in later generations. This is not the case when WUs belong to different classes. For example, suppose we have one class 1 WU whose branching distribution has mean 0.9, and three class 2 WUs whose branching distribution has mean 0.8. If we assign the class 1 WU to one processor, and all class 2 WU's to the second, then we clearly have a load imbalance in the first generation, but by the tenth generation the mean number of WUs on the first processor is 0.34 and the mean number on the second is 0.32. Alternately, if we place two WUs on each processor the load balance is perfect for the first generation, but by the tenth generation one processor has 0.45 WUs on average while the other has 0.21. Thus we see that when multiple classes of WUs are permitted the balance fundamentally changes as the computation progresses.

Nevertheless, it is sometimes still possible to determine that one multiclass mapping is better than another. The situation occurs when there exist processors i and j and classes a and b such that processor i is assigned more class a WUs than is j, and it is assigned more class b WUs than is j. We show that a better assignment is to move one class b (or a) WU on processor i to processor j.

A c-class assignment to K processors can be described by a $c \times K$ matrix, say $\underline{\mathbf{m}}$. Row a of the matrix is the mapping vector for class a WUs. For each class a, let $W^{(a)}$ be the random "workload" associated with a WU from class a; we will use $W_i^{(a)}$ to denote the workload on processor i from class a. As before, the notion of workload is permitted to be general, so long as it is additive. The random workload $W(i, \underline{\mathbf{m}})$ for processor i under $\underline{\mathbf{m}}$ is the convolution of convolutions

$$W(i, \underline{\mathbf{m}}) = \sum_{a=1}^{c} W_i^{(a)}.$$

As before, we denote the random vector of workloads under $\underline{\mathbf{m}}$ as $\mathbf{W}(\underline{\mathbf{m}})$. Then we have the following result.

Theorem 5.1 *Let $\underline{\mathbf{m}}'$ be a multiclass assignment matrix where there exist processors i, j such that $\forall k : m'_{k,i} \geq m'_{k,j}$ and there exists an a such that $m'_{a,i} > m'_{a,j}$. Construct $\underline{\mathbf{m}}$ from $\underline{\mathbf{m}}'$ by moving one class a WU from processor i to j. Then $\mathbf{W}(\underline{\mathbf{m}}) \prec_{cas} \mathbf{W}(\underline{\mathbf{m}}')$.*

Proof: We focus on processors i and j. We may write the vector of workloads on these processors from a class b as follows: $(W_i^{(b)}, W_j^{(b)}) = (\bar{W}_j^{(b)}, W_j^{(b)}) + (W^{(b)}, 0)$ where $\bar{W}_j^{(b)} =_{st} W_j^{(b)}$ and $W^{(b)} =_{st} W_i^{(b)} - W_j^{(b)}$. Now take

$$\begin{aligned}
(Z_1, Z_2) &= \sum_b (\bar{W}_j^{(b)}, W_j^{(b)}) \\
X &= \sum_{b \neq a} W^{(b)} \\
Y &= W^{(a)}.
\end{aligned}$$

The result follows by direct application of Corollary 2.1. ∎

5.2 Heterogeneous Composition Tasks

Consider the following construction. Let F be an integer-valued nonnegative probability distribution. A task is comprised of a sequence of randomly weighted stages. The weight w_1 of the first stage is any nonnegative integer; given the weight w_q of the qth stage, the weight w_{q+1} of the following stage is sampled from a w_q-fold convolution of F. We will call this a *composition task.* Observe that a composition task terminates as soon as one of its stages takes weight 0. We are interested in the problem of mapping a set of heterogeneous composition tasks.

A composition task with initial weight 1 is equivalent to a WU with branching distribution F; one defines the weight of the task's qth stage to be the number of generation q descendents of the WU. Once this is understood it is apparent that a composition task with initial weight w is equivalent to the convolution of w WU's having branching distribution F. This latter observation permits us to use majorization to compare mappings of heterogeneous composition tasks.

Consider a set of composition tasks all constructed using the same branching distribution F. We may classify a composition task by its initial weight, say, each class a task has initial weight w_a. Let $\mathbf{w}$ be the vector whose ath component is w_a. One may use an assignment matrix $\underline{\mathbf{m}}$ to describe a multiclass mapping. Given $\underline{\mathbf{m}}$, we define the *composition weight vector* $cw(\underline{\mathbf{m}})$ to be the vector-matrix product $\mathbf{w} \cdot \underline{\mathbf{m}}$. That is, the ith component of $cw(\underline{\mathbf{m}})$ is the weighted sum of tasks assigned to processor i, where the number from each class a is weighted by w_a. Finally, for each stage q, let $\mathbf{S}_q(\underline{\mathbf{m}})$ be the vector (across processors) of the sum of all task stage q weights under mapping $\underline{\mathbf{m}}$.

Theorem 5.2 *Let F be any branching distribution, and let A be any set of positive integers. Then*

$$\sum_{q \in A} \mathbf{S}_q(\underline{\mathbf{m}}) \prec_{cas} \sum_{q \in A} \mathbf{S}_q(\underline{\mathbf{m}}') \qquad \textit{whenever } cw(\underline{\mathbf{m}}) \prec cw(\underline{\mathbf{m}}')$$

where $\sum$ denotes component-wise summation. If F is ILR, then for any positive integer q and $A = \{q\}$ we may replace $\prec_{cas}$ with $\prec_{sc}$.

Proof: The stage q weight of a class a task is equivalent to the total number of generation q descendents from w_a initial WUs with branching distribution F. Thus, any multiclass assignment of composition tasks can be viewed as an assignment of WUs, with the ith component of $cw(\underline{\mathbf{m}})$ being the number of WUs assigned to processor i under the equivalent mapping of WUs. The conclusions follow from Theorems 3.4 and 3.3. ∎

Reconsider the problem of mapping i.i.d. WUs. If there are no constraints on the mapping, then the optimal solution is obvious—put the same number of WUs on each processor. The problem is not so easy with composition tasks. One would like to find a mapping $\underline{\mathbf{m}}$ such that every component of $cw(\underline{\mathbf{m}})$ is the same; however, this is not generally possible. Restricted to two processors, the problem is identical to the Subset Sum partitioning problem, known to be NP-Complete [5]. One might seek to balance to balance the first stage by finding a matrix $\underline{\mathbf{m}}$ that minimizes the sum of squares of stage 1 weights on each processor. This problem is identically the Minimized Sum of Squares problem, also known to be NP-Complete [5]. However, the results we've developed can be used at least to compare candidate mappings in a mapping heuristic.

We may assume that the mechanics of executing a composition task differ from those of executing a WU process. For example, it is natural to assume that no storage space associated with stage $q+1$ of a task is demanded before stage q is completely executed. If there are multiple composition tasks assigned to a processor, then upon the completion of one task's stage and the subsequent storing of its next stage's initial state, another task is selected for execution. From the point of view of execution time and reliability, a system of composition tasks does not differ from the corresponding system of WUs, so that the results of sections §4.1 and §4.3 still apply. However, composition tasks have a different space-time product behavior than WU systems. We expect that results similar to those developed for WU systems exist, but also note that unlike WU systems, the order in which task stages are selected for execution *does* affect the space-time product. This difference will have to be considered in any space-time analysis.

5.3 Heterogeneous Processors

Next we consider the problem of mapping i.i.d. WUs with an ILR branching distribution onto a system where processors have heterogeneous processing rates. There are actually two components to this problem. One is the partitioning of the workload, the other is the mapping. It does not appear that one can use majorization in any natural way to attack either component. However, we can show that if the workload has already been partitioned and the branching distribution is ILR, then the optimal mapping is found by iteratively assigning the "largest" unassigned partition group to the fastest unassigned processor. Throughout this section we assume that processor i executes workload at a rate r_i. Without loss of generality we may assume that $r_1 \leq r_2 \leq \ldots \leq r_K$.

Lemma 5.1 *Let $G(q)$ be the number of generation q descendents from an WU, and let $H(q)$ be the number of descendents in generations $2, \ldots, q$. If branching distribution is ILR, then $G(q)$ and $H(q)$ are ILR.*

Proof: Theorem 18.A.4.a in [10] can be used to show that the ILR property is preserved via convolution, and compounding. The fact that $G(q)$ is ILR is demonstrated using the compounding result in an induction on q. The fact that $H(q)$ is ILR is also found in an induction, where the induction step is proven by conditioning on the size of the second generation. ∎

Now suppose that a set of initial WUs has already been partitioned into K subsets, $P_1, \ldots P_K$, with cardinalities $p_1, \ldots, p_K$, and $p_1 \leq p_2 \leq \ldots p_K$. For each subset P_i, the distribution of the number of descendents in generation q of the WUs in P_i is a p_i-fold convolution of $G(q)$. Likewise, the distribution of the number of descendents in all generations through q is a p_i-fold convolution of $H(q)$. Both distributions are ILR, because ILR is preserved by convolution. Thus we have

$$G_{p_1}(q) \leq_{LR} G_{p_2}(q) \ldots \leq_{LR} G_{p_K}(q)$$

and

$$H_{p_1}(q) \leq_{LR} H_{p_2}(q) \ldots \leq_{LR} H_{p_K}(q).$$

Let $\boldsymbol{G}(q)$ and $\boldsymbol{H}(q)$ denote the associated random vectors, ordered by subset index. We may describe the mapping of the partitioned workload to processors with a permutation matrix Π, so that the jth component of $\Pi\boldsymbol{G}(q)$ is the workload assigned to processor j. If we have a computation that synchronizes between execution of generations, then the time required for all processors to execute generation q workload is $\max\{\Pi\boldsymbol{G}(q)\}$. If the processors are free to execute without synchronization, then the time required for all processors to execute as far as the qth generation is $\max\{\Pi\boldsymbol{H}(q)\}$. We are interested in finding the permutation matrix which minimizes the expectation of these finishing times. The following result, found in [15], helps to order permutations.

Lemma 5.2 ([15],page 269) *Suppose that X and Y are independent, and that $X \leq_{LR} Y$. If $h : \mathbb{R}^K \to \mathbb{R}$ satisfies*

$$h(x_1, \ldots, x_{i-2}, x_{i-1}, x_i, x_{i+1}, \ldots, x_K) \leq h(x_1, \ldots, x_{i-2}, x_i, x_{i-1}, x_{i+1}, \ldots, x_K) \qquad \textit{whenever } x_{i-1} \leq x_i,$$

then

$$h(x_1, \ldots, x_{i-2}, X, Y, x_{i+1}, \ldots, x_K) \leq_{st} h(x_1, \ldots, x_{i-2}, Y, X, x_{i+1}, \ldots, x_K).$$

Consider a permutation Π' where there are a WUs placed on processor i, b WUs placed on processor j, $i < j$ and $a > b$. Since we have assumed that the processors are indexed in decreasing order of their processing rates, Lemma 5.2 implies that exchanging the assignments of processors i and j improves any objective function of $\boldsymbol{G}(q)$ or $\boldsymbol{H}(q)$ that satisfies the conditions of Lemma 5.2. The function

$$\phi(x_1, \ldots, x_K) = \max\{x_1/r_1, x_2/r_r, \ldots, x_K/r_K\}$$

satisfies this condition,. It corresponds to the time required to complete the execution of workload vector $\boldsymbol{x}$. It follows that the optimal assignment is to place P_i onto processor i, for $i = 1, \ldots, K$. This assignment simply puts the jth largest workload on the jth fastest processor, for $j = 1, \ldots, K$. This assignment minimizes the time between synchronizations at every generation; in the absence of synchronization it minimizes the expected time required for all processors to compute through generation q workload. In the latter case one sees that the expected finishing time is minimized, by taking the limit as $q \to \infty$. If there are constraints on the mapping which prohibit this assignment, then Lemma 5.2 can still be used to improve a candidate assignment, provided the interchange does not violate a constraint.

6 Summary

This paper has explored the application of majorization to the problem of assigning a large number of stochastically complex (but probabilistically identical) tasks onto a multiprocessor. Using a model of workload based on branching processes, the theory we apply establishes a partial ordering among possible assignment of tasks to processors. We show that the quality of an initial assignment persists through stochastic transformations of the workload, and that the ordering can be taken with respect to a wide range of objective functions including those measuring finishing time, space-usage, and reliability. The utility of the theory lies in the generality of the objective functions that can be considered, and in the fact that the partial ordering may be an extremely useful tool in situations where finding an optimal solution is combinatorially complex. Finally, we consider problems associated with heterogeneous tasks and heterogeneous processors. Despite the additional complications posed by heterogeneity, we develop results permitting one to compare certain types of assignments.

References

[1] M.J.Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations", *J. Comp. Phys.*, 53:484–512, 1984.

[2] G.Brassard and P.Bratley, *Algorithmic:Theory and Practice*, Prentice-Hall,Englewood Cliffs, NJ, 1988.

[3] C-S.Chang, "A New Ordering for Stochastic Majorization: Theory and Applications", *IBM Report RC 16028*, TJ Watson Research Center, Yorktown Heights, NY, 1990.

[4] Y-C.Chow and W.H.Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System", *IEEE Transactions on Computers*, Vol. C-28, 1979, pp. 354-361.

[5] M.Garey and D.Johnson, *Computers and Intractability*, Freeman and Company, New York, 1979.

[6] E.Gelenbe, R.Nelson, T.Phillips and A.Tantawi, "An Approximation of the Processing Time for a Random Graph Model of Parallel Computation", *Proc. Int. Conf. on Parallel Processing*, 1986, pp. 691-697.

[7] J.Hong, X.Tan and M.Chen, "From Local to Global: An Analysis of Nearest Neighbour Bazlancing on Hypercube", *ACM SIGMETRICS*, 1989, pp. 73-82.

[8] B.Indurkhya, H.S.Stone and L.Xi-Cheng, "Optimal Partitioning of Randomly Generated Distributed Programs", *IEEE Transactions on Software Engg.*, Vol. SE-12, No. 3, March 1986, pp. 483-495.

[9] C.P.Kruskal and A.Weiss, "Allocating Independent Subtasks on Parallel Processors", *IEEE Transactions on Software Engg.*, Vol. SE-11, No. 10, Oct 1985, pp. 1001-1016.

[10] A.W.Marshall and I.Olkin, 'Inequalities: Theory of Majorization and Its Applications', *Academic Press*, 1979.

[11] P.Mussi and P.Nain, "Evaluation of Parallel Execution of Program Tree Structures", *ACM SIGMETRICS*, 1984, pp. 78-87.

[12] D.M.Nicol, "Optimal Partitioning of Random Programs Across Two Processors", *IEEE Transactions on Software Engg.*, Vol. 15, No. 2, Feb 1989, pp. 134-141.

[13] D.M.Nicol and J.H.Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands", *IEEE Transactions on Computers.*, Vol. 37, No. 9, Sept. 1988, pp. 1073-1087.

[14] C.H.Papadimitriou and K.Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall, 1982.

[15] S.Ross, 'Stochastic Process', *Wiley*, 1983.

[16] A.Tantawi and D.Towsley, "Optimal Static Load Balancing in Distributed Computer Systems", *J. ACM*, Vol. 32, 1985, pp. 445-465.

[17] A.Thomasian and P.F.Bay, "Analytic Queueing Network Models for Parallel Processing of Task Systems", *IEEE Transactions on Computers*, Vol. C-35, No. 12, Dec 1986, pp. 1045-1054.

[18] D.Towsley, "Applications of Sample Path Analysis to Communication Network Control", **DON YOU GOTTA FILL THIS IN**.

[19] R.R.Weber, "On the Optimal Assignment of Customers to Parallel Servers", *J. Applied Probability*, Vol. 15, pp. 406-413.

[20] D.D.Yao, "Majorization and Arrangement Orderings in Open Queueing Networks", *Annals of Operations Research*, Vol. 9, 1987, pp. 531-543.