NASA Contractor Report 189716

ICASE Report No. 92–51

ICASE

STATIC ASSIGNMENT OF COMPLEX STOCHASTIC TASKS USING STOCHASTIC MAJORIZATION

David Nicol
Rahul Simha
Don Towsley

Contract Nos. NAS1–18605 and NAS1–19480
October 1992

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23681–0001

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

STATIC ASSIGNMENT OF COMPLEX STOCHASTIC TASKS USING STOCHASTIC MAJORIZATION

David Nicol[1] *and Rahul Simha*[2]

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185

Don Towsley

Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003

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. 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 show how the theory may be applied to the problem of partitioning a pool of processors for distribution among parallelizable tasks.

[1] This research was supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-18605 and NAS1-19480 while the first author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001. Research also supported in part by NSF ASC 8819393.

[2] Research supported by NSF NCR-8907909.

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, 18]. 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, 18] 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. In 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, 18]. Much of this work pertains to steady-state expectations of task delays with state-dependent [4, 18] 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 C we may say that the assignment described by $\boldsymbol{m}$ is better than the assignment described by $\boldsymbol{m}'$. The class 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 n or K. In fact, in [9], the results are only asymptotically correct in both the number of tasks n 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) n 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 *branching process*: a completing process spawns a random number of subprocesses. 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 "processes" which give rise to "subprocesses" 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, 19]. 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 [19], 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 applies the theory to the problem of partitioning a pool of processors among a set of parallelizable tasks. Section §6 summarizes our work.

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], as follows. The task begins with a single *work unit* (WU) of computation. The WU is executed; upon its completion a random number of other WUs are created, and placed in the task's work list. The initial WU can thus be thought of as containing the "seeds" for a number of additional WUs, possibly zero, each of which similarly contain the seeds for additional WUs, and so on. One of the first generation WUs may then be executed, and its children (which are 2^{nd} generation WUs) spawned and placed in the task's work list. The number of children a WU spawns is assumed to be random, chosen from a probability distribution known as the *branching distribution.* The process is repeated until the task's work list is empty. The task workload is comprised of all computation related to all WUs ultimately descended from the initial task WU.

We assume that the order of WU execution in no way affects the spawning of children: a WU in the work list is destined to spawn some j children, regardless of the length of time it spends in the list. This is easily understood if one views the WU generation as reflecting some intrinsic structural property of the problem, e.g., the branching of a search tree. Because of this independence, every WU belongs to some "generation" which is independent of execution order. The initial WU is in generation 0; all children spawned by a generation 1 WU are in generation 2, and so on.

We assume that a given WU may be executed with the same constant cost on any one of K homogeneous processors, and that every WU is executed on the same processor as is its parent. Therefore, we map all computation associated with a task when we map the task's initial WU.

Consider the evolution of an initial task WU. Let N_q denote the number of WUs in its q-th generation. The size of the q-th generation is given as

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

where $N_0 = 1$ 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 a sequence of independent and identically distributed (i.i.d.) random variables (r.v.'s). The following notation will be employed:

- K the number of processors.
- n - the number of initial task WUs.
- $\boldsymbol{m}$ - an integer assignment vector whose i^{th} component m_i gives the number of WUs assigned to the i^{th} processor.
- N_q - the size of generation q, descended from a single initial WU (when the branching distribution is understood). For any subset $A \subseteq \mathbb{N}$, S_A is the sum of all sizes of generations $i \in A$:

$$S_A = \sum_{i \in A} N_i.$$

- $f^{(j)}$ – the j^{th} convolution of a probability mass function f. If X is a random variable, we will also use $X^{(j)}$ to denote a sum of j independent instances of X.
- $\boldsymbol{W}_q(\boldsymbol{m})$ – the random vector of generation q WUs resulting from assignment vector $\boldsymbol{m}$:

$$\boldsymbol{W}_q(\boldsymbol{m}) = (N_q^{(m_1)}, \ldots, N_q^{(m_K)}).$$

We denote the i^{th} component of $\boldsymbol{W}_q(\boldsymbol{m})$ by $(\boldsymbol{W}_q(\boldsymbol{m}))_i$. The notation is extended to arbitrary subsets $A \subseteq \mathbb{N}$ by

$$\boldsymbol{W}_A(\boldsymbol{m}) = (S_A^{(m_1)}, \ldots, S_A^{(m_K)}).$$

The theory we develop permits us to compare different mappings under a variety of objective functions $\phi : \mathbb{R}^K \rightarrow \mathbb{R}$. Our results focus on comparing values of $E\left[\phi(\boldsymbol{W}_q(\boldsymbol{m}))\right]$ by deriving conditions for inequalities involving initial task assignments $\boldsymbol{m}$. Most of these are of the following form: given two assignments $\boldsymbol{m}$ and $\boldsymbol{m}'$ where $\boldsymbol{m} \prec \boldsymbol{m}'$ (see Definition 2.1), then $E\left[\phi(\boldsymbol{W}_A(\boldsymbol{m}))\right] \leq E\left[\phi(\boldsymbol{W}_A(\boldsymbol{m}'))\right]$ for all subsets $A \subseteq \mathbb{N}$, when the expectations exist.

Applicable functions ϕ include any symmetric convex function; the maximum operator, all powers of the maximum, the sum operator, and the product operator are of particular interest. Thus a single comparison between the assignment vectors $\boldsymbol{m}$ and $\boldsymbol{m}'$ vectors can yield a wealth of information about the comparative behaviors of complex stochastic tasks under the two mappings.

Our results are applicable to two different types of processor synchronization. We study *generational synchronization* (GS) where processors engage in a barrier synchronization between each WU generation. A processor executes all WUs of a given generation, say q, then synchronizes at the barrier. It is not released until all processors have executed all their generation q WUs and reached the barrier. The process repeats for subsequent generations. This type of synchronization is appropriate when the computation for a generation q in one task may depend on results computed by a generation $q-1$ WU in another task. We also study *termination synchronization* (TS), where a processor engages in a barrier synchronization only after the work lists of all its initial tasks are empty. This is appropriate when the tasks are independent of each other, and the synchronization serves only to aggregate the final results of their respective computations.

Not surprisingly, the optimal way of assigning n tasks to K processors is usually to assign n/P to each. In the face of the obvious one may well ask why we study partial orderings. Primarily, the theory proves the optimality with respect to a large number of objective functions, thereby lending theoretical support to intuition. Secondly, the theory works even in the presence of constraints that disallow the uniform assignment, and complicate one's intuition concerning optimality. For example, memory constraints may exist that forbid one or more processors from being assigned more than n/P tasks. The theory identifies the optimal assignment under heterogeneous constraints.

We will also apply these concepts to the issue of partitioning a pool of processors among a set of complex parallelizable tasks. Here we'll take K to the be number of parallelizable tasks, and use $\boldsymbol{m}$ to describe the number of processors assigned to each. Constraints on feasible $\boldsymbol{m}$ are easily envisaged, as the assignment may need to consider "natural" partition sizes that arise from communication topology, or system usage at the time of the assignment. So again, while the optimal solution to the constraint-free version of the problem may be apparent, the theory provides a means of comparing feasible solutions.

2.2 Stochastic Ordering and Majorization

We now introduce the *majorization* partial ordering $\prec$ using notation largely taken from [10].

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

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

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 $\phi : \mathbb{R}^n \to \mathbb{R}$ is said to be Schur-convex if $\boldsymbol{x} \prec \boldsymbol{y}$ in $\mathbb{R}^n$ implies $\phi(\boldsymbol{x}) \leq \phi(\boldsymbol{y})$ in $\mathbb{R}$.*

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

Let C_0 be the class of increasing functions from $\mathbb{R}^n$ onto $\mathbb{R}$. The well-known stochastic ordering between random variables [15] is defined as follows. For random vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ with distribution functions F and G respectively,

$$\boldsymbol{X} \leq_{st} \boldsymbol{Y} \quad \text{iff} \int_{\mathbb{R}^n} \phi(\boldsymbol{x}) dF(\boldsymbol{x}) \leq \int_{\mathbb{R}^n} \phi(\boldsymbol{x}) dG(\boldsymbol{x}) \quad \forall \phi \in C_0$$

such that the integrals are well defined. Majorization over deterministic quantities is extended to random variables in like manner by using an appropriate class of functions:

$$\begin{array}{lllll} C_1 & = & \{scx\} & = & \{f : \mathbb{R}^n \to \mathbb{R} \mid f \text{ Schur-convex } \}, \\ C_2 & = & \{cas\} & = & \{f : \mathbb{R}^n \to \mathbb{R} \mid f \text{ convex and symmetric } \}. \end{array}$$

These define respectively 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, $\boldsymbol{X} \prec_{scx} \boldsymbol{Y}$ iff

$$\int_{\mathbb{R}^n} \phi(\boldsymbol{x}) dF(\boldsymbol{x}) \leq \int_{\mathbb{R}^n} \phi(\boldsymbol{x}) dG(\boldsymbol{x}) \quad \forall \phi \in C_1$$

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

$$\int_{\mathbb{R}^n} \phi(\boldsymbol{x}) dF(\boldsymbol{x}) \le \int_{\mathbb{R}^n} \phi(\boldsymbol{x}) dG(\boldsymbol{x}) \quad \forall \phi \in C_2.$$

Note that $C_2 \subset C_1$ and thus, $\prec_{scx}$ is a stronger ordering than $\prec_{cas}$.

Stochastic orderings based on *likelihood ratio* play an especially important role in this paper. Consider non-negative integer valued r.v.'s X and Y with probability mass functions f and g.

Definition 2.3 (likelihood ratio) *X is defined to be smaller than Y in likelihood ratio, written as $X \le_{lr} Y$, iff*

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

Another important property for a probability distribution is known as *increasing likelihood ratio.*

Definition 2.4 (ILR) *The non-negative integer valued r.v. X is said to have* increasing likelihood ratio *(ILR) (and its probability mass function f is said to be ILR) iff*

$$c_1 + X \le_{lr} c_2 + X, \quad \text{whenever } 0 \le c_1 \le c_2.$$

Next we define another class of probability mass functions, those which have *increasing likelihood ratio under convolution.*

Definition 2.5 (ILRC) *Let f be a probability mass function defined on $\mathbb{N}$. f is said to have* increasing likelihood under convolution *(ILRC) iff $f^{(i)} \le_{lr} f^{(j)}$ whenever $i < j$.*

ILR distributions are known to be closed under convolution, even when the number of times convolution is applied is random (provided the distribution of this number is also ILR) [10].

Lemma 2.1 *Let f be an ILR probability mass function. Then*

- *f is ILRC.*
- *For any fixed integer $k > 0$, $f^{(k)}$ is ILR.*
- *Let N be an ILR positive integer-valued random variable. Then $f^{(N)}$ is ILR.*

Using these facts it is straightforward to prove the following.

Lemma 2.2 *Let f be an ILR probability mass function. Then*

- *If f is the branching distribution for a task, then for all generations q, N_q is ILR.*
- *For any subset $A \subseteq \mathbb{N}$, if $S_A = \sum_{i \in A} N_i$ has finite mean, then S_A is ILRC.*

Proof: The proof of the first claim is a simple induction on q that uses closure of the ILR property under random ILR mixtures; the proof of the second rewrites $S_A^{(i)}$ as $N_q + c_i$, where q is the least element of A, and $c_i \leq c_j$ almost surely whenever $i < j$. The result follows from Definition 2.4 and the fact that N_q is ILR. ∎

As we will see, the assumption of an ILR branching distribution often yields $\prec_{scx}$ orderings. The ILR condition is true of the discrete Uniform, Poisson, Geometric and Binomial distributions, showing that our results apply when the branching assumes some well-known distributions.

Next we show how these stochastic orderings may be used to develop stochastic majorizations between different static mappings.

3 Branching and Stochastic Majorization

In this section we establish conditions under which either $\prec_{cas}$ or $\prec_{scx}$ orderings can be established between "workload" vectors under different mappings. The notion of workload will be seen to be quite general. Throughout this section it is important to remember that the results relate to intrinsic properties of branching behavior, and do not depend on assumptions about execution behavior, e.g., synchronization.

Our results for the $\prec_{scx}$ ordering is based on the following theorem which is an application of Theorem 3.J.2 in [10]. The correspondence between our form and the original is pointed out in the Appendix.

Theorem 3.1 *Let f be an ILRC probability mass function, let $\mathbf{m} = (m_1, \ldots, m_K)$ be a vector of nonnegative integers, and for each $j = 1, \ldots, K$ let $X^{(m_j)}$ be a r.v. with distribution $f^{(m_j)}$. Suppose this set of r.v.s is independent, and let $\phi : \mathbb{R}^K \rightarrow \mathbb{R}$ be a Schur-convex function. Then*

$$\gamma(\mathbf{m}) = E\left[\phi(X^{(m_1)}, \ldots, X^{(m_K)})\right]$$

is a Schur-convex function of $\mathbf{m}$.

Using Theorem 3.1 we obtain our basic $\prec_{scx}$ ordering results.

Theorem 3.2 *Consider a set of n tasks, with common ILR branching distribution f, and let $\mathbf{m}$ and $\mathbf{m}'$ be two mapping vectors such that $\mathbf{m} \prec \mathbf{m}'$. Then*

- *For all generations q, $\boldsymbol{W}_q(\boldsymbol{m}) \prec_{scx} \boldsymbol{W}_q(\boldsymbol{m}')$.*
- *For any subset $A \subseteq \mathbb{N}$ such that S_A has finite mean,*

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

Proof. Lemma 2.2 shows that the distributions of N_q and S_A are each ILRC; the result follows from the definitions of $\boldsymbol{W}_q(\boldsymbol{m})$ and $\boldsymbol{W}_A(\boldsymbol{m})$, and Theorem 3.1. ∎

Observe that the statement of Theorem 3.1 applies more generally to the notion of a random "reward" associated with each initial WU. It states that if each initial WU earns a random ILRC reward, and if the reward to a processor is the sum of the rewards earned by its (independent) WU's, then a stochastic majorization on the rewards follows from a deterministic majorization of the initial WUs.

Our $\prec_{scx}$ results seem to require the assumption of ILR or ILRC branching distributions. However, by constraining our attention to symmetric convex functions we are able to obtain $\prec_{cas}$ orderings for completely general branching distributions. The details, which are numerous, are developed in the Appendix. The $\prec_{cas}$ counterpart to Theorem 3.2 is

Theorem 3.3 *Consider a set of n tasks, with common nonnegative branching distribution f, and let $\boldsymbol{m}$ and $\boldsymbol{m}'$ be two mapping vectors such that $\boldsymbol{m} \prec \boldsymbol{m}'$. Then*

- *For all generations q, $\boldsymbol{W}_q(\boldsymbol{m}) \prec_{cas} \boldsymbol{W}_q(\boldsymbol{m}')$.*
- *For any subset $A \subseteq \mathbb{N}$ such that S_A has finite mean,*

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

3.1 Heterogenous Constraints

The K-vector $\boldsymbol{m}_{opt} = (n/K, n/K, \ldots, n/K)$ is majorized by any other vector whose components are nonnegative and sum to n. Applied to the assignment problem, this shows that the obvious way to balance workload is indeed the best, even for complex stochastic tasks. Optimality is less clear, however, if the obvious assignment is prohibited by constraints. For each processor i let C_i be an upper bound on the number of WUs the processor may be given. Such constraints might arise, for instance, if the processors have different memory capacities. The obvious mapping is prohibited if any $C_i < n/K$. Majorization provides a way to identify the best assignment of complex stochastic tasks even in the face of such constraints.

Consider any feasible vector $\mathbf{y} = (y_1, \ldots, y_K)$, $y_i \le C_i$ for $i = 1, \ldots, K$. Suppose there exist i and j such that $y_j > y_i + 1$, and $y_i + 1 \le C_i$. Construct a new vector $\mathbf{x}$ from $\mathbf{y}$ by transfering one unit from y_j to y_i, i.e., $x_j = y_j - 1$, $x_i = y_i + 1$, $x_k = y_k$ for all $k \ne i, j$. It is shown in [10] (5.D) that $\mathbf{x} \prec \mathbf{y}$. This observation gives a rule by which we can iteratively improve a feasible solution, until no further improvement is possible. We say a vector $\mathbf{x}$ resulting from this processed is *balanced.*

Without loss of generality assume that $C_1 \le C_2 \le \cdots \le C_K$. It is apparent that $\mathbf{x}$ is balanced if and only if whenever $x_j > x_i + 1$, then $x_i = C_i$. A characterization of balanced vectors then is that there is some index j such that $x_i = C_i$ for $i = 1, \ldots, j$, and for all $l, m > j$ we have $|x_l - x_m| \le 1$. Furthermore, if $\mathbf{x}$ and $\mathbf{y}$ are both balanced, then this index j is the same for both of them. It follows then that $\mathbf{x} \prec \mathbf{y}$ *and* $\mathbf{y} \prec \mathbf{x}$, which shows the essential uniqueness of balanced vectors. Balanced vectors are thus optimal under heterogenous constraints.

A simple $O(n)$ algorithm will construct a balanced assignment. Assume the processors are ordered by increasing constraint value, and initially set $x_i = 0$, $i = 1, 2, \ldots, K$. We loop repeatedly over indices 1 to K. Each pass through the loop we increment x_i once, provided $x_i < C_i$. This essentially assigns one unit to the i^{th} processor. We repeat the loop until all n units are assigned.

The main results of these section show that stochastic branching preserves stochastic majorization for additive reward systems. As we have seen, useful reward systems are derived from the generation sizes. The section to follow illustrates how these results can be fruitfully applied to various objective functions.

4 Objective Functions

We will now establish that a number of interesting objective functions are either Schur-convex or convex symmetric functions of some notion of workload. 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 can be established using different models of execution. For example, one easily envisions a computation where the tasks must synchronize globally after every generation, i.e., *GS synchronization.* This is typical of tasks associated with numerical computations. If the WUs each have unit execution time, then $\max_k\{(\boldsymbol{W}_q(\boldsymbol{m}))_k\}$ time is required under mapping $\boldsymbol{m}$ to execute all generation q WUs. N_q can be viewed as a random reward associated with an initial WU, thus Theorem 3.3 tells us that $\boldsymbol{W}_q(\boldsymbol{m}) \prec_{cas} \boldsymbol{W}_q(\boldsymbol{m}')$. The max operator is convex and symmetric, whence $E\left[\max_k\{(\boldsymbol{W}_q(\boldsymbol{m}))_k\}\right] \le E\left[\max_k\{(\boldsymbol{W}_q(\boldsymbol{m}'))_k\}\right]$.

This same result holds true if the WU execution times are random, and i.i.d. . Since the time between each synchronization is no larger under $\boldsymbol{m}$ than than under $\boldsymbol{m}'$, it follows that the overall finishing time is no larger.

Similar results are obtained under *TS synchronization*, where processors synchronize only at termination. The reward for an initial WU can be defined to be $S_{\boldsymbol{N}}$, the total size of the branching tree rooted in that WU. When the mean of the branching distribution is strictly less than one, then $E[S_{\boldsymbol{N}}] < \infty$. In this case, whenever $\boldsymbol{m} \prec \boldsymbol{m}'$, the expected maximum processor reward under $\boldsymbol{m}$ is no larger than under $\boldsymbol{m}'$. Even when the branching distribution's mean is greater than or equal to one (but is finite) we can always assert that the time to execute all generations up through q is no greater under $\boldsymbol{m}$ than it is under $\boldsymbol{m}'$, by defining the reward for an initial WU to be the sum of the sizes of generations through q. Any symmetric convex function of the processor rewards—such as the maximum processor reward—yields an $\prec_{cas}$ ordering.

Another metric of interest is the variation in the time to synchronize. The sample variance, defined below, is also symmetric and convex.

$$\begin{aligned} SampleVar(\boldsymbol{x}) &= \left(\sum_{k=1}^{n} (x_i - \bar{\boldsymbol{x}})^2 \right) \\ &= \sum_{k=1}^{n} x_i^2 - \bar{\boldsymbol{x}}^2 \end{aligned}$$

where $\bar{\boldsymbol{x}} = (\sum x_i)/n$. Thus,

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

for any generation q, and

$$SampleVar(\boldsymbol{W}_A(\boldsymbol{m})) \prec_{cas} SampleVar(\boldsymbol{W}_q(\boldsymbol{m}'))$$

for any $A \subseteq \boldsymbol{N}$ such that S_A has finite mean. When the branching distribution is ILR, a similar result holds true for the sample standard deviation (square root of variance) of time between synchronizations, because the standard deviation is Schur-convex ([10], pp. 71).

4.2 Functions of Queue Length

When a WU completes its execution it generates its children and places them on the processor's work list. 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, defined as follows. Let $\boldsymbol{Q}(t)$ be the vector enumerating the number of WUs enqueued at each processor at time t, and let T be the computation's termination time. Then the total space-time product is $\sum_{k=1}^{K}\sum_{t=0}^{T}(\boldsymbol{Q}(t))_k$. This idea can be generalized—let $s(j)$ quantify the cost of holding j WUs in queue for one unit of time. Then the total space-time cost with respect to s is $\sum_{k=1}^{K}\sum_{t=0}^{T}s((\boldsymbol{Q}(t))_k)$. We will show that if s is increasing convex with $s(0) = 0$, and if $\boldsymbol{m} \prec \boldsymbol{m}'$, then under TS synchronization the expected space-time cost with respect to s is no worse under $\boldsymbol{m}$ than it is under $\boldsymbol{m}'$. This result is also demonstrated for GS synchronization when the branching distribution is ILR.

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): whenever a processor selects a WU for execution from its work list, it chooses one with least generation index. For simplicity, we also assume that the execution of a WU takes unit time.

The space-time function $s(k) = k$ gives rise to the usual space-time product, but other space-time cost 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 would be

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

where $\epsilon \ge 0$. The general assumptions that a space-time cost function be convex, increasing, and zero for empty queue lists seem to us quite natural.

Our treatment of space-time costs under TS synchronization hinges on the following observation: if processor k has exactly $(\boldsymbol{W}_q(\boldsymbol{m}))_k$ WU units in generation q, then under the SGF queueing discipline *at some point in time the processor's queue will have exactly* $(\boldsymbol{W}_q(\boldsymbol{m}))_k$ *WUs.* In particular, at the instant where the first WU of generation q is about to be executed, the queue consists entirely of generation q WUs, and contains all of them. We will show that the contribution to the expected space cost made by processor k while processing generation q WUs (under SGF scheduling) is an increasing convex function of $(\boldsymbol{W}_q(\boldsymbol{m}))_k$, and use this fact to find a majorization on the vector of expected contributions made by all processors while processing generation q WUs. This, in turn, will show that the total expected space-time cost under $\boldsymbol{m}$ is no worse than under $\boldsymbol{m}'$, when the expectations exist. This is a $\prec_{cas}$ result, applicable for any branching distribution.

Suppose $(\boldsymbol{W}_q(\boldsymbol{m}))_k = r$. The processing of the i^{th} WU in generation q $(i = 1, \ldots, r)$ produces a random number $X_{q,i}$ of WU units, who join the processor's queue. The queue length at the instant the i^{th} WU begins execution is $r - (i-1) + \sum_{j=1}^{i-1} X_{k,j}$, as there were r work units in queue at the point the first generation q WU was executed, $i - 1$ of them have been executed, and each one produced a random number of generation $q + 1$ WUs. Therefore, the conditional expected space-time cost suffered during the processing of this WU is

$$\phi(i,r) = E\left[s(r-(i-1)+\sum_{j=1}^{i-1} X_{q,j})\right]. \tag{2}$$

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). The expected space-time cost of processing all r members of generation q on processor k is

$$C_s(r) = \sum_{i=1}^{r} \phi(i,r).$$

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

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

The fact that $s(r)$ is increasing 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 $C_s(r)$.

The function $\mathcal{T}_s(r_1,\ldots,r_K) = \sum_{k=1}^{K} C_s(r_k)$ is symmetric and convex on $\mathbb{N}^K$, because whenever g is convex on $\mathbb{R}$ then $h(x) = \sum g(x_i)$ is convex on $\mathbb{R}^K$. Observe that $\mathcal{T}_s(\boldsymbol{W}_q(\boldsymbol{m}))$ is the random space-time cost with respect to s and generation q resulting from assignment vector $\boldsymbol{m}$. We have proven the following result.

Proposition 4.1 *Let s be an increasing convex function with $s(0) = 0$ and suppose the space cost of holding k WUs in one processor's queue for one time unit is $s(k)$. Define*

$$\mathcal{T}_s(\boldsymbol{W}_q(\boldsymbol{m})) = \sum_{k=1}^{K} C_s((\boldsymbol{W}_q(\boldsymbol{m}))_k)$$

to measure the space-time cost suffered while executing generation q, under the assignment given by $\boldsymbol{m}$. Then whenever $\boldsymbol{m} \prec \boldsymbol{m}'$,

- $E[\mathcal{T}_s(\boldsymbol{W}_q(\boldsymbol{m}))] \leq E[\mathcal{T}_s(\boldsymbol{W}_q(\boldsymbol{m}'))]$ *for $q = 0, 1, \ldots$.*
- *The expected total space-time cost using TS synchronization is no worse under $\boldsymbol{m}$ than under $\boldsymbol{m}'$:*

$$E[\sum_{q=0}^{\infty} \mathcal{T}_s(\boldsymbol{W}_q(\boldsymbol{m}))] \leq E[\sum_{q=0}^{\infty} \mathcal{T}_s(\boldsymbol{W}_q(\boldsymbol{m}'))] \quad \text{whenever the expectation exists.}$$

The analysis of space-time costs under GS synchronization requires more work, and the assumption of an ILR branching distribution. Suppose that $(\boldsymbol{W}_q(\boldsymbol{m}))_k = r_k$, for $k = 1, \ldots, K$. The space-time cost to processor k during the interval of time when generation q WUs are executed has two components. We have already seen the first: $C(r_k)$—the cost accumulated over the period of length r_k while generation q WUs are executed. The second component is the space-time cost suffered waiting for the most heavily loaded processor to finish. If processor k generates x generation $q+1$ WUs, then the space-time cost it suffers waiting at the barrier is $(\max_i\{r_i\} - r_k)s(x)$. Recalling the definition of ϕ (equation (2)) we may write the expected total space-time cost of processing generation q WUs under GS synchronization (conditioned on $(\boldsymbol{W}_q(\boldsymbol{m}))_k = r_k$, for $k = 1, \ldots, K$) as

$$\mathcal{G}(r_1, \ldots, r_K) = \sum_{k=1}^{K} \left(\sum_{i=1}^{r_k} \phi(i, r_k) + (\max_j\{r_j\} - r_k)\phi(r_k + 1, r_k) \right),$$

Observe that $\phi(r_k + 1, r_k)$ is $E[s(X^{(r_k)})]$, where X is the branching random variable . $\mathcal{G}$ is Schur-convex on $\mathbb{N}^K$, a fact we show using the following characterization of Schur-convex functions on $\mathbb{N}^K$ (3.A.2.b in [10]).

> A function α on $\mathbb{N}^K$ is Schur-convex if and only if α is symmetric and
>
> $$\alpha(r_1, t - r_1, r_3, \ldots, r_K) \quad \text{is increasing in } r_1 \geq t/2$$
>
> for each fixed $t, r_3, \ldots, r_K$.

Fix $r_3, \ldots, r_K$, and consider $r_1 > r_2$. If the difference $\mathcal{G}(r_1 + 1, r_2 - 1, \ldots, r_K) - \mathcal{G}(r_1, r_2, \ldots, r_K)$ is always nonnegative, then the condition above tells us that $\mathcal{G}$ is Schur-convex. We need to examine two cases, $\max_j\{r_j\} = r_1$, and the alternative. Assuming the former, straightforward algebra shows that the difference is bounded from below by

$$\sum_{i=1}^{r_1} [\phi(i, r_1 + 1) - \phi(i, r_1)] - \sum_{i=1}^{r_2 - 1} [\phi(i, r_2) - \phi(i, r_2 - 1)] +$$
$$(\phi(r_1 + 1, r_1 + 1) - \phi(r_2, r_2)) - (r_1 - r_2)(\phi(r_2 + 1, r_2) - \phi(r_2, r_2 - 1)).$$

Both of the two summations above are positive, because $\phi(i, r)$ increases in r. Since $\phi(i, r)$ is convex in r and $r_1 > r_2$, it also follows that $\phi(i, r_1 + 1) - \phi(i, r_1) \geq \phi(i, r_2) - \phi(i, r_2 - 1)$ for every i. Thus the positive summation above dominates the negative summation, and the desired inequality will hold if

$$(\phi(r_1 + 1, r_1 + 1) - \phi(r_2, r_2)) - (r_1 - r_2)(\phi(r_2 + 1, r_2) - \phi(r_2, r_2 - 1)) \geq 0.$$

Since $\phi(r, r)$ is a convex function of r, we have

$$\phi(r_1 + 1, r_1 + 1) - \phi(r_2, r_2) \quad = \quad \sum_{i=1}^{r_1 + 1 - r_2} (\phi(r_2 + i, r_2 + i) - \phi(r_2 + i - 1, r_2 + i - 1))$$

$$
\begin{aligned}
&\geq \sum_{i=1}^{r_1+1-r_2} (\phi(r_2+1, r_2+1) - \phi(r_2, r_2)) \\
&= (r_1 - r_2)(\phi(r_2+1, r_2+1) - \phi(r_2, r_2)).
\end{aligned}
$$

From this inequality we see that the desired bound will hold if $(\phi(r_2+1, r_2+1) - \phi(r_2, r_2)) \geq (\phi(r_2+1, r_2) - \phi(r_2, r_2-1))$. The convexity of s implies that

$$
\begin{aligned}
\phi(r_2+1, r_2+1) - \phi(r_2, r_2) &= E[s(1 + X^{(r_2)}) - s(1 + X^{(r_2-1)})] \\
&\geq E[s(X(r_2) - s(X(r_2 - 1))] \\
&= \phi(r_2+1, r_2) - \phi(r_2, r_2-1),
\end{aligned}
$$

as needed.

The argument for the case when $r_1 \neq \max_j\{r_j\}$ is almost exactly the same, and so is omitted.

The Schur-convexity of $\mathcal{G}$ gives us a stochastic majorization for GS synchronization.

Proposition 4.2 *Let s be an increasing convex function with $s(0) = 0$, suppose the space cost of holding k WUs in one processor's queue for one time unit is $s(k)$, and suppose the branching distribution is ILR. Define*

$$
\mathcal{G}(r_1, \ldots, r_K) = \sum_{k=1}^{K} \left(\sum_{i=1}^{r_k} \phi(i, r_k) + (\max_j\{r_j\} - r_k)\phi(r_k+1, r_k) \right),
$$

to measure the space-time cost with respect to s of executing some generation q under GS synchronization, where the each processor i has r_i generation q WUs. Then $\mathcal{G}$ is Schur-convex on $\mathbb{N}^K$, so that whenever $\boldsymbol{m} \prec \boldsymbol{m}'$,

- $E[\mathcal{G}(\boldsymbol{W}_q(\boldsymbol{m}))] \leq E[\mathcal{G}(\boldsymbol{W}_q(\boldsymbol{m}))]$ *for $q = 0, 1, \ldots$.*
- *The expected total space-time cost using GS synchronization is no worse under $\boldsymbol{m}$ than under $\boldsymbol{m}'$:*

$$
E[\sum_{q=0}^{\infty} \mathcal{G}(\boldsymbol{W}_q(\boldsymbol{m}))] \leq E[\sum_{q=0}^{\infty} \mathcal{G}(\boldsymbol{W}_q(\boldsymbol{m}))] \quad \textit{whenever the expectation exists.}
$$

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 a processor's time-to-failure distribution has an increasing hazard rate function, the 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 $\boldsymbol{m}$. The result is proven for TS synchronization.

Suppose that processor i's time to failure is the random variable Z_i, with an monotone hazard rate function $\lambda(u)$. It is well known that

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

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

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

$$\mathcal{R}(t_1, \ldots, t_K) = \prod_{i=1}^{K} \Pr\{Z_i > t_i\} \tag{3}$$

is Schur-concave, or equivalently, that $-\mathcal{R}(t_1, \ldots, t_K)$ is Schur-convex. When $\lambda(u)$ decreases then $\mathcal{R}(t_1, \ldots, t_K)$ is Schur-convex.

If processor i is assigned m_i WUs initially, it ends up processing $S_N^{(m_i)}$ WUs total. This is also processor i's processing time under the assumptions of SGF scheduling, TS synchronization, and unit execution cost per WU. Given $S_N^{(m_i)} = t_k$ for $i = 1, \ldots, K$, equation (3) gives the probability that every processor executes all WUs without processor failure. The unconditional probability is obtained by taking the expectation with respect to the joint distribution of $\boldsymbol{S_N}(\boldsymbol{m})$:

$$\Pr\{\text{every processor executes all its WUs before failing}\} \quad = \quad E[\mathcal{R}(\boldsymbol{S_N}(\boldsymbol{m}))].$$

Lemma 2.2 asserts that S_N is ILRC if the branching distribution is ILR. It follows from Theorem 3.1 that when $\lambda(u)$ is increasing, $E[\mathcal{R}(S_N(\boldsymbol{m}))]$ is a Schur-concave function of $\boldsymbol{m}$. This proves the following proposition.

Proposition 4.3 *Suppose the hazard rate function $\lambda(u)$ for the time to processor failure is increasing, and suppose the branching distribution is ILR. Let $\gamma(\boldsymbol{m})$ be the probability that every processor executes all its WUs without processor failure. Then under TS synchronization and SGF scheduling, whenever $\boldsymbol{m} \prec \boldsymbol{m}'$ we have $\gamma(\boldsymbol{m}) \geq \gamma(\boldsymbol{m}')$ The inequality is reversed if $\lambda(u)$ is decreasing.*

5 Assignment of Processor Pools

Our last application of stochastic majorization concerns a problem where a large number P of processors are to be partitioned among a smaller number T of complex tasks. Parallel processing can be applied to the tasks to accelerate execution time. We assume that a task requires that all of its generation i WUs to be executed before any of its generation $i+1$ WUs are, but that all generation i WUs may be processed in parallel. As before, the overall system may use either TS or GS synchronization.

Let $g(X,m)$ give the time required by m processors to execute X WUs. We assume that $g(X,m)$ is convex in m, e.g., $g(X,m) = X/m$, and that $g(0,m) = 0$.

Suppose there are K initial WUs. We may describe our assignment of n processors to these WUs with vector $\boldsymbol{m}$, whose i^{th} component gives the number of processors assigned to the i^{th} WU. Also let $N_{q,i}$ denote the random number of WUs associated with generation q of task i. Under GS synchronization, the time required to complete the q^{th} generation is

$$\gamma_q(\boldsymbol{m}) = \max\{g(N_{q,1},m_1), g(N_{q,2},m_2), \ldots, g(N_{q,K},m_K)\}.$$

Under our assumptions, $E[\gamma_q(\boldsymbol{m})]$ is a symmetric convex function of $\boldsymbol{m}$ (B.4 Proposition in [10]), showing that $E[\gamma_q(\boldsymbol{m})] \leq E[\gamma_q(\boldsymbol{m}')]$ whenever $\boldsymbol{m} \prec \boldsymbol{m}'$. It follows immediately that the overall expected finishing time under GS synchronization is no worse under $\boldsymbol{m}$ than under $\boldsymbol{m}'$.

Under TS synchronization the finishing time is

$$\rho(\boldsymbol{m}) = \max\{\sum_{q=0}^{\infty} g(N_{q,1},m_1), \ldots, \sum_{q=0}^{\infty} g(N_{q,K},m_k)\}.$$

A sum of convex functions remains convex, whence $E[\rho(\boldsymbol{m})]$ is symmetric and convex in $\boldsymbol{m}$. When $\boldsymbol{m} \prec \boldsymbol{m}'$ we are assured that the expected finishing time under TS synchronization is no worse using $\boldsymbol{m}$ than it is with $\boldsymbol{m}'$.

6 Conclusions

This paper explores 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 develop 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. We also show how the theory applies to the processor partitioning problem. The utility of the theory lies in the generality of the objective functions that can be considered, and in the fact that optimal solutions can be identified even when constraints are placed on potential assignments.

A Appendix

In this appendix we prove some claims made earlier in the paper.

The ILRC condition upon which our $\prec_{scx}$ results depend involves the notion of *totally positive functions*. Chapter 18 of [10] is the source for the following definition.

Definition A.1 (Totally Positive Function) *Let A and B be subsets of the real line. A function $\alpha : A \times B \to \mathbb{R}$ is said to be* totally positive of order k, *denoted TP_k, if for all m, $1 \leq m \leq k$ and all $x_1 < x_2 < \ldots < x_m$, $y_1 < y_2 < \ldots < y_m$ ($x_i \in A, y_j \in B$)*

$$\det \begin{bmatrix} \alpha(x_1, y_1) & \cdots & \alpha(x_1, y_m) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \alpha(x_m, y_1) & \cdots & \alpha(x_m, y_m) \end{bmatrix} \geq 0.$$

We will use the following result (18.A.4.a in [10]).

Lemma A.1 *If K is TP_m and L is TP_n, and σ is a σ-finite measure, then the convolution*

$$M(x, y) = \int K(x, z) L(z, y) d\sigma(z)$$

is $TP_{\min\{m,n\}}$.

The relationship between total positivity and ILRC distributions is direct. Given any integer-valued nonnegative probability mass function f we may define the function $\alpha_f : \mathbb{N} \times \mathbb{N} \to [0, 1]$:

$$\alpha_f(i, x) = f^{(i)}(x).$$

α_f is TP_2 iff

$$f^{(i)}(n) f^{(j)}(m) \geq f^{(i)}(m) f^{(j)}(n)$$

for all $i < j$, $m < n$. But this is equivalent to saying that $f^{(i)} \leq_{lr} f^{(j)}$, i.e., that f is ILRC.

The reason for our interest in ILRC distributions f is that their convolution functions α_f satisfy three criteria required by Theorem 3.J.2 of [10]

- $\alpha_f(x, y) = 0$ whenever $y < 0$;
- α_f is totally positive of order 2;
- $\alpha_f(x + z, y) = \int \alpha_f(x, u) \alpha(z, y - u) dv(u)$, for some measure v on $\mathbb{N}$.

Theorem 3.J.2's conclusion is that if $\boldsymbol{m} = (m_1, \ldots, m_K) \in \mathbb{N}^K$, μ is counting measure, and $\phi : \mathbb{R}_+^K \rightarrow$ is Schur-convex, then

$$\gamma(\boldsymbol{m}) = \int_{(y_1,\ldots,y_K)} \prod_{i=1}^{K} \alpha_f(m_i, y_i)\phi(\boldsymbol{y}) \prod \mu(y_i) \tag{4}$$

is Schur-convex on $\mathbb{N}^K$. Theorem 3.1 is a restatement of this result, where $\forall u, dv(u) = 1$; because $\alpha_f(m_i, y_i)$ is a probability, we recognize that $\gamma((m))$ expresses the expected value of $\phi(\boldsymbol{y})$.

$\prec_{cas}$ Results

We next consider the $\prec_{cas}$ ordering. In this case, we are able to obtain the analogue of Theorem 3.2, save that the $\prec_{cas}$ result holds for completely general branching distributions. We first must introduce a little more terminology, and develop an intermediate result.

A random vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ is said to have ***exchangeable components*** if the joint distribution of $X_1, \ldots, X_n$ is invariant under permutations of its components. Our basic $\prec_{cas}$ results rest on the following observation.

Lemma A.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, Y and $\boldsymbol{Z}$ are independent r.v.'s and define $\boldsymbol{U} = (X, Y)$ and $\boldsymbol{V} = (X + Y, 0)$. Then*

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

Proof. Let $\phi : \mathbb{R}_+^2 \rightarrow \mathbb{R}$ be a convex symmetric function. Define the function $\psi : \mathbb{R}_+^2 \rightarrow \mathbb{R}$ 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.

Now $\boldsymbol{U} \prec \boldsymbol{V}$ a.s. from which it follows

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

■

The result extends easily to $\mathbb{R}_+^n$.

Lemma A.3 *Let X,Y be any nonnegative random variables, let $\boldsymbol{Z} = (Z_1, Z_2, \cdots, Z_n) \in \mathbb{R}^n_+$ be a random vector with independent components such that Z_1 and Z_2 have the same distribution. Assume that X, Y, and the components of $\boldsymbol{Z}$ are mutually independent and define $\boldsymbol{U} = (X, Y, 0, \cdots, 0)$ and $\boldsymbol{V} = (X + Y, 0, \cdots, 0)$. Then*

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

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

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

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

We are now prepared to prove Theorem 3.3. Let $\boldsymbol{m}'$ be any mapping vector where there are processors i, j such that $m'_i > m'_j$. Without loss of generality we may take $i = 1$ and $j = 2$, and let $\boldsymbol{m}''$ be the mapping vector obtained from $\boldsymbol{m}'$ by moving one WU from processor 1 to processor 2. We will apply lemma A.2. Interpret Z_1, Z_2 as m'_2-fold convolutions of initial WU rewards, X as the convolution of $m'_1 - m'_2 - 1$ initial WU rewards, Y as a single initial WU reward, and each Z_j for $j > 2$ as the convolution of m'_j initial WU workloads. The application of lemma A.3 yields $\boldsymbol{R}(\boldsymbol{m}'') \prec_{cas} \boldsymbol{R}(\boldsymbol{m}')$.

The incremental movement of a task from a heavily loaded processor to a more lightly loaded processor corresponds to the more general notion of a "transfer" [10]. It is known that whenever $\boldsymbol{x} \prec \boldsymbol{y}$, then $\boldsymbol{x}$ can be constructed from $\boldsymbol{y}$ with a finite number of transfers, where each transformed vector is always dominated under $\prec$ by its predecessor. Consequently if $\boldsymbol{m}'$ is a mapping vector with $\boldsymbol{m} \prec \boldsymbol{m}'$, then one demonstrates that $\boldsymbol{W}(\boldsymbol{m}) \prec_{cas} \boldsymbol{W}(\boldsymbol{m}')$ through a repeated application of Lemma A.3 to the sequence of transfers that transmute $\boldsymbol{m}'$ into $\boldsymbol{m}$. This proves the result.

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 Balancing 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] R.R.Weber, "On the Optimal Assignment of Customers to Parallel Servers", *J. Applied Probability*, Vol. 15, 1978, pp. 406-413.

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