



ELSEVIER

Performance Evaluation 46 (2001) 219–233

**PERFORMANCE
EVALUATION**
An International
Journal

www.elsevier.com/locate/peva

Dynamic on-line task scheduling on parallel processors

Cathy H. Xia^{a,*}, George Michailidis^{b,1}, Nicholas Bambos^{c,d,2}

^a IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA

^b Department of Statistics, University of Michigan, Ann Arbor, MI 48109, USA

^c Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA

^d Department of Management Science and Engineering, Stanford University, Stanford, CA 94305, USA

Abstract

The problem of scheduling on-line a stream of multi-tasked jobs on a set of parallel processors is considered. The goal is to minimize the average sojourn time experienced by each individual job in the system. The arriving jobs are comprised of parallel applications, each consisting of multiple independent tasks that are instantaneously assigned to processor queues upon its arrival to the system. The processors independently and concurrently service the tasks in their local queues which are first-come-first-served (FCFS). A job can depart from the system only when all its tasks have been executed and reassembled into the original single-job structure; until then, completed tasks are placed in a reassembly queue. All tasks have i.i.d. memoryless exponential service times and any task can be processed by any processor. Migration of tasks between queues — after their initial allocation — is not allowed. This model captures the main features of multiprocessor computer systems executing parallel programs.

The key scheduling issue is as follows. When some queue backlogs are small, an incoming job should spread its tasks to those lightly loaded queues in order to take advantage of the parallel processing gain and lower its processing delay. On the other hand, when all queues are fairly congested, the job should schedule all its tasks sequentially in a single queue to avoid excessive reassembly delay (due to backlog fluctuations) and lower its task synchronization delay. In this paper, the trade-off between these two objectives is quantified and it is shown that the optimal schedule's structure is based on backlog thresholds. Moreover, an off-line mechanism for determining these thresholds is provided, and further characteristics of the optimal scheduling policy under special cases is discussed. Finally, asymptotics and approximations for systems comprised of a large number of processors are considered. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Task schedule; Parallel processors; Backlog thresholds

1. Introduction

The problem of scheduling a set of distributed resources for parallel execution of tasks associated with a single job, arises in a wide range of distributed computer applications, manufacturing systems and

* Corresponding author. Tel.: +1-914-784-7844; fax: +1-914-784-7455.

E-mail address: cathyx@us.ibm.com (C.H. Xia).

¹ Research supported by the National Science Foundation.

² Research supported in part by the National Science Foundation.

communication network environments. The way in which the processors are allocated to the tasks of such parallel applications is fundamental to realizing the high performance of the corresponding systems, such as minimizing mean job response time and maximizing system throughput.

Specifically, we consider the problem of on-line scheduling the tasks of each incoming job on a set of distributed parallel processors. Each incoming job requires the completion of a number of tasks that can be executed separately by any of the servers, and a job can only leave the system when all of its tasks are completed. The objective is to determine the optimal schedule that minimizes the job average delay time in the system. The distinguishing features of the model described above are the so-called fork and join operations. A *fork* occurs whenever a job is split into multiple tasks which are then sent in parallel to their respective servers. A *join*, on the other hand, occurs when the finished tasks of a job may not leave the system until the rest of the tasks corresponding to the same job are also completed.

The behavior of this fork–join type of model has received considerable attention from the performance analysis perspective; for example in a series of papers [1–3,6,9,10], the throughput of the system under stationary ergodic inputs and under different set-ups has been studied. The work was primarily motivated by applications in the area of multiprocessor computer systems running parallel programs [3,4] but there are also applications in the context of flexible manufacturing systems [1].

In this work, we consider the problem from the optimal scheduling point of view. An early reference on this problem is [7], while a static version of the problem is examined in [4]. Intuitively, executing jobs in parallel across multiple processors has the benefits of speeding up job delay time at the price of incurring overheads associated with task synchronization. This trade-off between parallel and sequential executions leads to the fact that the optimal policy should have a threshold structure; that is, when the shortest queue length is beyond a certain threshold value, all incoming jobs should be scheduled in a sequential fashion (assigned to the same processor). In this paper, we show that not only there is a threshold beyond which no splitting is encouraged, but there are also layers of thresholds corresponding to different levels of splitting. In other words, the number of processors to split a job is monotone non-increasing in queue length. We provide a practical mechanism in constructing the optimal policy, and also address large-scale approximations to help estimate the sequential thresholds.

The remainder of this paper is organized as follows. Section 2 introduces the model under consideration and discuss some facts about the optimal policy. In Section 3, we present our mathematical analysis for the basic structure of the optimal schedule, discuss some practical scheduling strategies, and also provide further characterization of the optimal policy for special cases. The issue of how to calculate the completion time function and further obtain the optimal policy is addressed in Section 5, where we also provide asymptotics and approximations for systems comprised of a large number of processors. Our concluding remarks are drawn in Section 6.

2. Model and preliminaries

In this section, we present the parallel queuing system used in our study, and provide some technical preliminaries that are useful in constructing the optimal scheduling policy.

2.1. System model

Consider a distributed parallel processing system (see Fig. 1) comprised of K identical processors each one equipped with an infinity capacity queue. There is also a single arrival stream of jobs to the system,

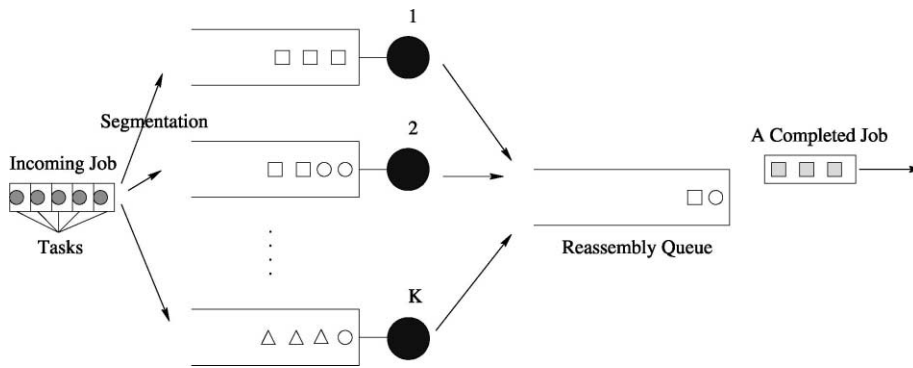


Fig. 1. A demonstration of the segmentation of an incoming job to its constituent tasks and distribution of those into the parallel queues for processing.

each consisting of multiple independent tasks that are instantaneously assigned to one of the K processor queues upon their arrival to the system. The processors independently and concurrently service the tasks in their local queues based on a first-come-first-serve (FCFS) discipline. A job can depart from the system only when all its tasks have been executed and reassembled into the original single-job structure. All tasks can be processed by any processor; each processor runs independently without idling until all jobs in its local queue have received service. Migration of tasks between queues — after their initial allocation — is not allowed. This model captures the main features of multiprocessor computer systems executing parallel programs as well as the splitting/reassembly dynamics of messages in IP packet switched networks [11]. We assume that running times of the tasks are i.i.d. exponentially distributed.³ A (on-line individual) scheduling policy is a strategy that allocates the tasks of each job upon its arrival to the processor queues. We say that a policy is *non-splitting* (or *sequential*) if the entire set of the tasks of a particular job are scheduled to run sequentially on the same processor. On the other hand, a *splitting* policy schedules the tasks of a job to multiple processor queues so that they can be run independently and potentially in parallel on different processors. The goal is to minimize the average sojourn time experienced by each individual job in the system. In this paper, we restrict attention to non-preemptive, non-idling and non-anticipative scheduling policies, where by non-anticipative we mean that the controller is only allowed to make use of the past history and the current state of the system.

2.2. Optimal scheduling

Let S be the random variable representing the individual task service time. At a given instant, we say that the system is in state $\mathbf{x} := (x_1, x_2, \dots, x_K)$, where x_k is the number of tasks in the queue of processor $k, k = 1, \dots, K$. Throughout the paper, we will use $\mathbf{x}^d := (x_1, \dots, x_d)$ to denote the vector composed by the first d components, and denote $|\mathbf{x}^d| = \sum_{i=1}^d x_i$ to be the corresponding vector sum for \mathbf{x}^d . Clearly, for the first d processors to finish serving the \mathbf{x}^d tasks in their corresponding queues, the expected completion time is

$$\eta(\mathbf{x}^d) = \mathbf{E}\{\max[Y_1(x_1), \dots, Y_d(x_d)]\}, \tag{1}$$

³ A similar approach can be applied if the running times are geometrically distributed in slotted time.

where $Y_i(x_i)$ represents the running time of processor i to finish the x_i tasks in queue, which is the sum of x_d independent copies of S . For convenience, when \mathbf{x}^d has all its components $x_i = x$, $i = 1, \dots, d$, we use $\eta_d(x)$ to denote the corresponding η -function.

Suppose that a job with N tasks arrives at time t , observing the system in state $\mathbf{x} = (x_1, \dots, x_K)$. Clearly, the N tasks of the incoming job can be split in various ways over an arbitrary number, say d ($1 \leq d \leq K$) of the K processors. For example, an incoming job to a system with $K = 3$ processors comprised of $N = 3$ jobs could be split among $d = 2$ processors — for example, one task assigned to the first and the remaining tasks to the second processor — or among $d = 3$ processors, with a single task assigned to every processor, and so on. If the job is segmented among d processors, then the resulting queue lengths of the corresponding d processors have a direct impact upon the job's completion time.

In order to assess the performance of the various possible splittings, we need to make comparisons among functions of real vectors. Given two real vectors $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and $\mathbf{y} = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$. Denote by $x_{[i]}$ (resp. $y_{[i]}$) the i th largest component of \mathbf{x} (resp. \mathbf{y}). We say that vector \mathbf{x} is *majorized* by \mathbf{y} , written as $\mathbf{x} <_w \mathbf{y}$, if $\sum_{i=1}^k x_{[i]} \leq \sum_{i=1}^k y_{[i]}$, for all $k = 1, \dots, n-1$, and $\sum_{i=1}^n x_{[i]} = \sum_{i=1}^n y_{[i]}$. Therefore, majorization provides us with a measure of how the components of two vectors with identical total values are distributed, and induces a *partial ordering* on \mathbb{R}^n [8]. Intuitively, it says that \mathbf{x} is distributing its components more equally than \mathbf{y} when $\mathbf{x} < \mathbf{y}$. In addition, we say that a function is Schur convex and it preserves the majorization ordering. That is, function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is Schur convex if $\mathbf{x} < \mathbf{y}$ implies $f(\mathbf{x}) \leq f(\mathbf{y})$.

It can be easily shown that the η -function defined in (1) is an increasing Schur convex function due to the increasing and Schur convex properties of the max function (see for example [8]).

Makowski and Nelson [7] proved the structural properties of the (individually) optimal scheduling policy for the queuing system defined above. Suppose upon arrival, a job finds the system in state $\mathbf{x} = (x_1, \dots, x_K)$. We assume, without loss of generality, that the processors are numbered so that $x_1 \leq x_2 \leq \dots \leq x_K$. The policy first determines the value of $d^* \geq 1$ that satisfies

$$d^* = \arg \min\{1 \leq d \leq K : \eta(x_1 + c_1^d, \dots, x_d + c_d^d)\}, \quad (2)$$

where c_i^d denotes the number of tasks that would be assigned to the i th queue if the arriving job's N tasks were scheduled in a manner that tends to equalize (in the stochastic majorization sense) the queue lengths of the first d processors, $c_i^d > 0$, $\forall i = 1, \dots, d$, and $\sum_{i=1}^d c_i^d = N$. The individually optimal scheduling policy then places c_i^d of the N tasks of the job arrival on the i th (shortest) processor queue, $1 \leq i \leq d^*$.

The logic for the individually optimal policy is clear. Since the η -function is Schur convex that preserves the majorization ordering, its value becomes smaller when its arguments become more "balanced". This property suggests that if the tasks of an incoming job are to be distributed over d processors, the best splitting scheme should result in the most "balanced" d queue length distribution. Therefore, all that remains is to figure out the right number d^* of such processors. Given the complexity of an exact mathematical analysis to identify d^* using (2), we next present a structural study on d^* based on the properties of the η -function.

3. Structure of the optimal policy

Throughout the rest of the paper, we consider an incoming job with $N (\geq 2)$ tasks, finding the system in state $\mathbf{x} = (x_1, \dots, x_K)$, where the queues are numbered so that $x_1 \leq x_2 \leq \dots \leq x_K$.

We say that $\mathbf{c}^d = (c_1, \dots, c_d)$ is a d -splitting of the N tasks over the first d (shortest) queues in state $\mathbf{x}^d = (x_1, \dots, x_d)$ if

$$c_1 + \dots + c_d = N, \quad c_1, \dots, c_d > 0. \tag{3}$$

Clearly, based on the Schur convexity of the η -function, we know that the best d -splitting, say \mathbf{c}_*^d , is one that satisfies

$$\mathbf{x}^d + \mathbf{c}_*^d < \mathbf{x}^d + \mathbf{c}^d \quad \text{for all } d\text{-splittings } \mathbf{c}^d.$$

If migration of tasks between queues — after their initial allocation — were allowed, clearly the most balanced d -vector corresponding to the total workload $|\mathbf{x}^d| + N$ is to divide them equally into the d buffers. Let $\bar{\mathbf{x}}^d(N)$ denote the resulting queue length vector, with its components sorted in a *non-increasing* order and equal either to $\lfloor (x_1 + \dots + x_d + N)/d \rfloor$ or to $\lceil (x_1 + \dots + x_d + N)/d \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ represent the floor and ceiling functions, respectively. We say that $\bar{\mathbf{x}}^d(N)$ corresponds to *the balanced d -spread* of N over \mathbf{x}^d , and denote the corresponding completion time function by

$$\phi_d(|\mathbf{x}^d| + N) := \eta(\bar{\mathbf{x}}^d(N)).$$

Furthermore, we say that $\bar{\mathbf{x}}^d(N)$ is *feasible* if

$$\left\lfloor \frac{x_1 + \dots + x_d + N}{d} \right\rfloor > x_d. \tag{4}$$

Clearly condition (4) implies that $\bar{x}_i^d(N) > x_i$ for all $i = 1, \dots, d$. In this case, $\bar{\mathbf{x}}^d(N)$ corresponds to a feasible d -splitting \mathbf{c}^d , with $c_i = \bar{x}_i^d(N) - x_i$ for $i = 1, \dots, d$, in which no migration is needed for the x_1, \dots, x_d tasks that are already in the respective queues.

We next examine more closely the characteristics of the balanced d -spreads. The following lemma reduces the number of the necessary comparisons in (2) to be made. It implies that if the *balanced d -spread* of the N tasks over \mathbf{x}^d is not feasible, then there is no need to consider any other k -splitting for $k \geq d$.

Lemma 3.1. *If*

$$\left\lfloor \frac{x_1 + \dots + x_d + N}{d} \right\rfloor \leq x_d, \tag{5}$$

then for all d -splittings \mathbf{c}^d ,

$$\phi_{d-1}(|\mathbf{x}^{d-1}| + N) \leq \phi_d(|\mathbf{x}^d| + N) \leq \eta(\mathbf{x}^d + \mathbf{c}^d). \tag{6}$$

Proof. Let $\bar{\mathbf{x}}^d(N)$ and $\bar{\mathbf{x}}^{d-1}(N)$ be the corresponding balanced spread of N over \mathbf{x}^d and \mathbf{x}^{d-1} , respectively, such that $|\bar{\mathbf{x}}^d(N)| = |\mathbf{x}^d| + N$, $|\bar{\mathbf{x}}^{d-1}(N)| = |\mathbf{x}^{d-1}| + N$. Since $\bar{\mathbf{x}}^d(N)$ is infeasible, we must have $\bar{x}_d^d(N) = \lfloor (x_1 + \dots + x_d + N)/d \rfloor \leq x_d$. It follows that

$$\sum_{i=1}^{d-1} \bar{x}_i^{d-1}(N) = |\mathbf{x}^{d-1}| + N \leq |\mathbf{x}^{d-1}| + x_d + N - \bar{x}_d^d(N) = |\bar{\mathbf{x}}^d(N)| - \bar{x}_d^d(N) = \sum_{i=1}^{d-1} \bar{x}_i^d(N).$$

We then have $(\bar{x}_1^{d-1}(N), \dots, \bar{x}_{(d-1)}^{d-1}(N)) \leq (\bar{x}_1^d(N), \dots, \bar{x}_{(d-1)}^d(N))$, and it follows that

$$\eta(\bar{x}_1^{d-1}(N), \dots, \bar{x}_{(d-1)}^{d-1}(N)) \leq \eta(\bar{x}_1^d(N), \dots, \bar{x}_{(d-1)}^d(N)) \leq \eta(\bar{x}_1^d(N), \dots, \bar{x}_d^d(N)) \leq \eta(x_1 + c_1, \dots, x_d + c_d),$$

where the first inequality is due to the increasing property of the η -function, and the last one is due to its Schur convexity. □

The next lemma further characterizes the balanced d -spread.

Lemma 3.2. *If the balanced d -spread $\bar{\mathbf{x}}^d(N)$ is feasible, then so is the balanced $(d - 1)$ -spread $\bar{\mathbf{x}}^{d-1}(N)$.*

Proof. Since $\bar{\mathbf{x}}^d(N)$ is feasible, using the same reasoning as in the proof of Lemma 3.1, it can be easily established that

$$\sum_{i=1}^{d-1} \bar{x}_i^{d-1}(N) > \sum_{i=1}^{d-1} \bar{x}_i^d(N).$$

Note that the components of both $\bar{\mathbf{x}}^d(N)$ and $\bar{\mathbf{x}}^{d-1}(N)$ are labeled in a *non-increasing* order, we must have

$$\bar{x}_{(d-1)}^{d-1}(N) \geq \bar{x}_{(d-1)}^d(N) > x_{d-1}.$$

Therefore, the balanced $(d - 1)$ -spread $\bar{\mathbf{x}}^{d-1}(N)$ is also feasible. □

Define next

$$k^* := \max \left\{ d : \left\lfloor \frac{x_1 + \dots + x_d + N}{d} \right\rfloor > x_d \right\}, \tag{7}$$

which determines the largest feasible balanced d -spread. Note that the balanced 1-spread is always feasible, so k^* is well defined. From Lemma 3.2 we know that for all $i \leq k^*$, the balanced i -spread is feasible. Lemma 3.1 implies that there is no need to consider the $(k^* + 1)$ -splitting or beyond it. Therefore (2) in Makowski’s optimal theorem can be refined as follows.

Proposition 3.3. *The optimal d^* in (2) is given by*

$$d^* = \arg \min \{ 1 \leq d \leq k^* : \phi_d(|\mathbf{x}^d| + N) \}. \tag{8}$$

Proof. The result follows directly from Lemmas 3.1 and 3.2. □

3.1. Implementation of the optimal policy

We discuss next some practical implementation issues for identifying the optimal d^* . According to Proposition 3.3, it suffices to compare the η -values of the first k^* balanced spreads, $\phi_d(|\mathbf{x}^d| + N)$, $d = 1, \dots, k^*$, in order to determine the optimal scheduling of the tasks of an incoming job. Due to the convexity of the η -function, the calculation for increasing d can be terminated as soon as one finds a value

Table 1
Expected completion times for different numbers of total tasks for a 3-processor system

N	$\phi_1(N)$	$\phi_2(N)$	$\phi_3(N)$
2	2.00	1.50	–
3	3.00	2.25	1.83
4	4.00	2.75	2.44
5	5.00	3.44	2.88
6	6.00	3.94	3.21
7	7.00	4.59	3.75
8	8.00	5.09	4.16
9	9.00	5.73	4.50
10	10.00	6.23	5.00
...

of d for which $\phi_d(|\mathbf{x}^d| + N) < \phi_{d+1}(|\mathbf{x}^{d+1}| + N)$ is satisfied. The practical implication of this finding is that the values of the expected completion times of the first k^* $\phi_d(N)$'s for different total task sizes N can be computed in advance and stored. The on-line schedule is then of linear complexity since it only involves at most k^* comparisons.

To illustrate this last point, let us consider a 3-processor system with task times $S \sim \exp(1)$. First we generate the off-line table which records the expected completion time of the balanced d -spreads for different numbers of total tasks N , as shown in Table 1. The first column indicates the total number of tasks N to be allocated, while the second, third and fourth columns record, respectively, the corresponding expected completion times of the balanced 1-, 2-, and 3-spreads, namely $\phi_1(N)$, $\phi_2(N)$ and $\phi_3(N)$. A method to calculate the value of the η (and therefore the ϕ) function is given in Section 5.

Consider for example a 4-task job arriving to a 3-processor system in state $\mathbf{x} = (1, 2, 5)$. The optimal policy first determines that $k^* = 2$; subsequently, it looks up the table for the following entries: $\phi_1(1 + 4)$ and $\phi_2(1 + 2 + 4)$. Since $\phi_1(1 + 4) = 5 > \phi_2(1 + 2 + 4) = 4.59$, the optimal $d^* = 2$ and the policy is simply to balance spread the 4 tasks over the first two queues — by allocating 2 tasks to the first processor and the remaining 2 to the second processor.

4. Further characterization of the optimal policy

We see in the last section that the balanced spreads takes the most advantage of parallelism. However, this benefit from parallelism is obtained at the price of incurring overheads associated with task synchronization. In this section, we show that as more and more tasks get queued, the synchronization will become more and more expensive and the benefit of parallelism would eventually diminish. For special cases, we establish further structural characteristics of the optimal policy.

4.1. A threshold for parallelism

We assume without loss of generality that $S \sim \exp(1)$. For the general case with $S \sim \exp(\mu)$, the necessary scaling factor can be easily computed. The value of $\eta(x, x)$ can then be computed directly

from its probability density function or recursively using the random walk techniques which will be discussed in the next section.

Lemma 4.1.

$$\eta(x, x) = x + x \binom{2x}{x} \left(\frac{1}{2}\right)^{2x}, \quad (9)$$

which for large values x can be approximated by

$$\eta(x, x) \approx x + \frac{1}{\sqrt{\pi}} \sqrt{x}. \quad (10)$$

Proof. See the next section on how to obtain (9). An application of Stirling's formula to (9) then establishes (10). \square

Note that if the task time is deterministic with $S \equiv 1$ a.s., we would then have $\eta(x, x) = x$. Therefore, the second part in (9) is due to the variability of the task times S , and its effect on the completion time will become larger and larger as more and more tasks await processing, which can be observed from (10). Then Corollary 4.2 follows.

Corollary 4.2. *For any $N > 0$, there exists a finite threshold x^* such that $\eta_d(x) > x + N$ for all $x \geq x^*$, and $d \geq 2$.*

Corollary 4.2 says that due to the large scale characteristics of the η -function, when the number of jobs in each queue becomes very large, the reassembly from multiple processors becomes expensive due to the large variation in completion time of each queue; therefore, assigning all the tasks of an incoming job to the same processor is preferable since it incurs a smaller cost (expected completion time). Hence, there exists a finite threshold x^* for the shortest queue length, above which sequential processing of the tasks on a single processor outperforms parallel processing. The next proposition summarizes this threshold structure of the optimal policy.

Proposition 4.3. *The optimal $d^* \equiv 1$ whenever the shortest queue length x_1 exceeds some constant threshold x^* . In this case, the optimal policy corresponds to sequentially schedule all N tasks on the first processor.*

4.2. A layered threshold structure

We next consider the special case where $N = K$ and the system is in the completely balanced state $\mathbf{x} = (x, x, \dots, x)$. In this case, it is easy to see that $k^* = K$ and the balanced spread is feasible for all d . Let $d^*(x)$ be the optimal number of processors that should be used to balance spread the K tasks. We would like to look at the structural properties of $d^*(x)$ as a function of the queue length x .

Corollary 4.2 states that there is a finite x^* such that $d^*(x) = 1$ for all $x \geq x^*$. In this section, we show that not only there is a threshold for parallelism beyond which the sequential scheduling is optimal, there also exist layers of thresholds, with each layer corresponding to different number of processors to be used for parallel processing.

With a simple application of the Jensen’s inequality, one can easily show that

$$\eta(x + 1 + \lfloor \frac{1}{2}N \rfloor, x + 1 + \lceil \frac{1}{2}N \rceil) - \eta(x + \lfloor \frac{1}{2}N \rfloor, x + \lceil \frac{1}{2}N \rceil) \geq x + 1 + N - (x + N).$$

Therefore, whenever $d^*(x) = 1$, we have $d^*(y) = 1$ for all $y \geq x$.

This non-increasing property of $d^*(x)$ in x is also true for $N = K = 3$ as established in the next lemma.

Lemma 4.4. *Consider $N = K = 3$. For all $y \geq x$, $\eta(x + 1, x + 2) \leq \eta(x + 1, x + 1, x + 1)$ must imply $\eta(y + 1, y + 2) \leq \eta(y + 1, y + 1, y + 1)$. Therefore, whenever $d^*(x) \leq 2$, we also have $d^*(y) \leq 2$.*

Proof. It suffices to show that for $y = x + 1$,

$$\eta(y + 1, y + 1, y + 1) - \eta(x + 1, x + 1, x + 1) \geq \eta(y + 1, y + 2) - \eta(x + 1, x + 2). \tag{11}$$

The necessary calculations for establishing (11) are given in the next section. □

A heuristic justification for (11) is that peeling-off one task from each of the 3 processors should take more time than from 2 processors as more variability is involved. Therefore, when it becomes better to split N over 2 processors rather than over 3 for a system in state (x, x, x) , it is also better splitting over 2 than over 3 for any state (y, y, y) with $y \geq x$. This indicates that the higher x is, the fewer splittings it encourages.

Numerical experiments show that such phenomena are true in general for higher dimensions, that is, if

$$\phi_{d-1}((d - 1)x + K) \leq \phi_d(dx + K), \tag{12}$$

then for all $y \geq x$,

$$\phi_{d-1}((d - 1)y + K) \leq \phi_d(dy + K). \tag{13}$$

As an immediate consequence, we conclude the following corollary.

Corollary 4.5. *$d^*(x)$ is monotonically non-increasing in x . Equivalently, this indicates that there are layers of thresholds $x_K^* < x_{K-1}^* < \dots < x_1^*$ such that*

$$d^*(x) = d \quad \text{for } x_d^* \leq x < x_{d-1}^*,$$

where $d = 1, \dots, K$ with $x_K^* = 0$ and $x_0^* = \infty$.

The following example illustrates the property stated above. Consider a $K = 3$ processor system with exponentially mean 1 service times, and suppose that a 3-task job arrives to the system in state (x, x, x) . Numerical calculations show that the optimal policy is to equally spread the 3 tasks to $d^*(x)$ processors listed as in Table 2. Clearly, $d^*(x)$ is non-increasing in x , and the layers of thresholds are given by $x_3^* = 0$, $x_2^* = 3$, $x_1^* = 6$, $x_0^* = \infty$.

4.3. Estimation of the optimal thresholds

For a 2-server system, using the fact that $\phi_2(x_1 + x_2 + N) \geq \eta(x_1 + \lfloor \frac{1}{2}N \rfloor, x_1 + \lceil \frac{1}{2}N \rceil) \geq \eta(x_1, x_1) + \lfloor \frac{1}{2}N \rfloor$, the threshold x^* (for the shortest queue x_1) upon which no-splitting is encouraged can be

Table 2
Optimal thresholds for a 3-processor example

x	0	1	2	3	4	5	6	7	8	9	10	...
$d^*(x)$	3	3	3	2	2	2	1	1	1	1	1	...

determined by

$$\eta(x, x) > x + \frac{1}{2}(N + 1). \quad (14)$$

Based on (10), we then have an estimation of the optimal threshold x^* for large N ,

$$x^* \approx \frac{1}{4}\pi(N + 1)^2. \quad (15)$$

As the dimension d increases, the calculation of the η -values becomes more complicated. The following approximation comes handy in such cases. Recall that for large x , an application of the central limit theorem establishes that $(Y_i(x) - x)/\sigma\sqrt{x}$, $i = 1, \dots, d$, can be approximated by a standard normal variate, where $\sigma^2 < \infty$ denotes the variance of a single task processing time. Thus, we can approximate $(\eta_d(x) - x)/\sigma\sqrt{x}$ by the expectation of the maximum of d independent standard normal variates. Using a result in [5], we get that $\mathbf{E} \max_{1 \leq i \leq d} Z_i \approx \sqrt{2 \log d}$ for large d , where $Z_i \sim N(0, 1)$'s are i.i.d. random variables. Hence,

$$\eta_d(x) \approx x + \sigma\sqrt{2 \log d}\sqrt{x}$$

for large d . Note the generality of the above approximation, since it is independent of the task time distributions and their parameters.

This approximation can then be used to estimate the *optimal sequential threshold* x^* (above whose value all tasks of an incoming job should be scheduled to the shortest queue without splitting):

$$x^* \approx \frac{N^2}{2\sigma^2 \log 2}.$$

It can be seen that x^* grows roughly as N^2 for large N , and is inversely proportional to the variance of the task times. The latter suggests that as the variability of the task processing times decreases, parallel scheduling of incoming jobs becomes more preferable.

5. Calculating the η -function

In determining the optimal schedule for an incoming job, we have to compare the η -values of the feasible balanced spreads. Here we assume w.l.o.g. that task times are $\sim \exp(1)$. The following lemma can be obtained easily.

Lemma 5.1.

$$\eta_d(1) = \sum_{k=1}^d \frac{1}{k}. \quad (16)$$

Proof of Lemma 4.1. From (19), we know that

$$\eta(x + 1, x + 1) - \eta(x, x) = 1 + \frac{1}{2} \binom{2x}{x} \left(\frac{1}{2}\right)^{2x}.$$

Using the fact that

$$\frac{1}{2} \binom{2x}{x} \left(\frac{1}{2}\right)^{2x} = (x + 1) \binom{2(x + 1)}{x + 1} \left(\frac{1}{2}\right)^{2(x+1)} - x \binom{2x}{x} \left(\frac{1}{2}\right)^{2x},$$

we can then obtain (9) easily. □

Similarly, $\eta(x_1, \dots, x_d)$ is related to the extinction time of the random walk in \mathbb{Z}_+^d . Note that

$$\eta(\mathbf{x}^d) = \frac{1}{d} + \sum_{i=1}^d \frac{1}{d} \eta(\mathbf{x}^d - e_i), \tag{20}$$

where $\mathbf{x}^d = (x_1, \dots, x_d)$ and e_i is a d -dimensional unit vector with its i th entry being 1 and the remaining entries being 0. Let $T(\mathbf{x}^d)$ be the expected extinction time of the random walk that starts from \mathbf{x}^d and every time moves one-step downward in one of the d directions with probability $1/d$ until it is absorbed by the origin 0^d . We then have that $\eta(\mathbf{x}^d) = (1/d)T(\mathbf{x}^d)$. By reasoning analogously, we have

$$\eta(\mathbf{x}^d + \mathbf{1}^d) - \eta(\mathbf{x}^d) = p_d^d \eta_d(1) + p_{d-1}^d \eta_{d-1}(1) + \dots + p_1^d \eta_1(1), \tag{21}$$

where $\mathbf{1}^d$ is a d -dimensional vector with all entries being 1, p_k^d denotes the probability that, when hitting the $\mathbf{1}^d$ -frame starting from $\mathbf{x}^d + \mathbf{1}^d$, the random walk first hits points that have exactly k entries being equal to 1 on the frame.

The following result will be useful in proving Lemma 4.4.

Lemma 5.2. *Let $\delta(x_1, x_2) = \eta(x_1 + 1, x_2 + 1) - \eta(x_1, x_2)$; then δ is decreasing in x_2 for all $x_2 \geq x_1 - 1$, and symmetrically, decreasing in x_1 for all $x_1 \geq x_2 - 1$.*

Proof. It can be easily checked that p_2 given by (18) has the monotonicity property stated above. □

In general, let $\Delta_d(x) = \eta_d(x + 1) - \eta_d(x)$; then Δ_d is a decreasing function in x . This follows from the fact that the larger the x is, the smaller the probabilities p_k^d will be.

We are now ready to prove Lemma 4.4.

Proof of Lemma 4.4. It suffices to show that

$$\Delta_3(x) \geq \Delta_2(x), \tag{22}$$

from which (11) can then be established using the fact that $\Delta_2(x) \geq \eta(x + 1, x + 2) - \eta(x, x + 1)$ based on Lemma 5.2.

We know from (19) that

$$\Delta_2(x) = 1 + \frac{1}{2} p_2^2, \tag{23}$$

where p_2^2 is given by (18).

Based on the random walk related to $\eta_3(x)$ in \mathbb{Z}^3 , we know that

$$\Delta_3(x) = p_3^3 \eta_3(1) + p_2^3 \eta_2(1) + p_1^3 \eta_1(1) = 1 + \frac{1}{2}(p_2^3 + p_3^3) + \frac{1}{3}p_3^3, \tag{24}$$

where

$$p_3^3 = \binom{3x}{x, x} \left(\frac{1}{3}\right)^{3x}.$$

Note that p_2^3 is simply the probability that the random walk in \mathbb{Z}^3 , when starting from (x, x, x) , first hits one of the three edges $E = X^- \cup Y^- \cup Z^-$, where X^- denotes the negative portion of the X -axis, similarly for Y^- and Z^- . Since the walk is symmetric, $p_2^3 = 3P_{Z^-}(x)$. Note that when first hitting the axis Z , the last step of the random walk must be either in X or Y direction, therefore,

$$P_{Z^-}(x) = P_Z(x) - \frac{2}{3}\beta_x, \tag{25}$$

with

$$\beta_x = \sum_{k=0}^x \binom{k+2x-1}{x, x-1} \left(\frac{1}{3}\right)^{k+2x-1}.$$

Let

$$\gamma_x = \sum_{k=0}^x \binom{k+2x}{x, x} \left(\frac{1}{3}\right)^{k+2x}.$$

Use

$$\binom{k+2x}{2x} = \binom{k+2x-1}{2x} + \binom{k+2x-1}{2x-1},$$

and work out the algebra, we can obtain that

$$\beta_x = \gamma_x + \frac{1}{2}p_3^3. \tag{26}$$

We observe that $P_Z(x)$ can be computed from the projection of the random walk onto the X – Y plane, where the resulting projected walk decreases in X or Y direction or does not move, each with probability $\frac{1}{3}$, and $P_Z(x)$ then corresponds to the probability, say $p(x, x)$, of first hitting the origin $(0, 0)$ from (x, x) . Note that $p(x, x) = \frac{1}{3}p(x, x) + \frac{1}{3}p(x-1, x) + \frac{1}{3}p(x, x-1)$, which implies that $p(x, x) = \frac{1}{2}p(x-1, x) + \frac{1}{2}p(x, x-1)$. It follows that $p(x, x) = p_2^2$. We then have ⁴

⁴ The inequality in (27) should be intuitive. It can also be established as follows. It suffices to show that

$$\sum_{k=0}^x \binom{k+2x}{k} \left(\frac{1}{3}\right)^{k+2x} \leq \left(\frac{1}{2}\right)^{2x}.$$

Let

$$l_x = \sum_{k=0}^x \binom{k+2x}{k} \left(\frac{1}{3}\right)^k,$$

it can be checked that

$$l_x \left(\frac{2}{3}\right)^2 = l_{x-1} - \frac{1}{3x} \binom{3x-2}{x-1} \left(\frac{1}{3}\right)^x.$$

Therefore, $l_x \leq \left(\frac{3}{2}\right)^2 \leq l_{x-1} \leq \dots \leq \left(\frac{3}{2}\right)^{2x}$.

$$P_Z(x) = p_2^2 \geq \gamma_x = \sum_{k=0}^x \binom{k+2x}{x, x} \left(\frac{1}{3}\right)^{k+2x}. \quad (27)$$

Using (25)–(27), we then have

$$p_2^3 + p_3^3 = 3P_Z(x) - 2\gamma_x \geq p_2^2. \quad (28)$$

Relationship (22) then follows immediately from (23), (24) and (28). \square

6. Concluding remarks

In this paper, we examined the problem of scheduling the tasks of a job on a parallel processor system and identified a low complexity on-line optimal policy. The main findings are the following:

1. In general, the optimal schedule can be determined by comparing at most K η -values of the most balanced spread of the tasks of the incoming job, and it has a structure separated by layers of thresholds. More specifically, upon the arrival of a job with N tasks, if the smallest queue length of the system is beyond the first threshold value, then all the N tasks should be scheduled sequentially to the smallest queue using a single processor. Otherwise, choose the right layer to decide the optimal number of processors to be used for parallel processing.
2. In implementing the optimal policy, one has only to compute the table of expected completion times *off-line*. Then, the optimal policy boils down to identifying and comparing the appropriate entries in the table.
3. A random walk approach and an approximation method are used to evaluate the completion time function η , which also gives a qualitative way of identifying the optimal sequential threshold. Our results are in accordance with our intuition that as a general rule, less variability favors parallelism; moreover, it is shown that the threshold grows as N^2 .

Our results suggest that parallelism is not always efficient unless the task times are nearly deterministic. If the system is already heavily loaded, the variation of the actual task processing times along with the synchronization overhead associated with parallel processing will favor sequential processing. From the view point of individual jobs they simply need to choose the optimal number of shortest queues to split its tasks, where the appropriate splitting would result in the most balanced queue lengths.

References

- [1] F. Baccelli, W.A. Massey, D. Towsley, Acyclic fork–join queuing networks, *J. ACM* 36 (1989) 615–642.
- [2] F. Baccelli, A.M. Makowski, A. Schwartz, The fork–join queue and related systems with synchronization constraints: stochastic ordering and computable bounds, *Adv. Appl. Probab.* 21 (1989) 629–660.
- [3] F. Baccelli, Z. Liu, On the execution of parallel programs on multiprocessor systems — a queueing theoretic approach, *J. ACM* 37 (1990) 373–414.
- [4] C.S. Chang, R. Nelson, D.D. Yao, Scheduling parallel processors: structural properties and optimal policies, *Math. Comput. Model.* 23 (1994) 93–114.
- [5] J. Galambos, *The Asymptotic Theory of Extreme Order Statistics* 2nd edition, Krieger, Melbourne, FL, 1987.
- [6] P. Konstantopoulos, J. Walrand, Stationarity and stability of fork–join networks, *J. Appl. Probab.* 26 (3) (1989) 604–614.

- [7] A.M. Makowski, R.D. Nelson, Optimal scheduling for a distributed parallel processing model, IBM Research Report RC-17449, 1992.
- [8] A.W. Marshall, I. Olkin, Inequalities: Theory of Majorization and Its Applications, Academic Press, New York, 1979.
- [9] R. Nelson, A.N. Tantawi, Approximate analysis of fork/join synchronization in parallel queues, IEEE Trans. Comput. 37 (1988) 739–743.
- [10] R. Nelson, D. Towsley, A.N. Tantawi, Performance analysis of parallel processing systems, IEEE Trans. Software Eng. 14 (1988) 532–540.
- [11] L.L. Peterson, B.S. Davie, Computer Networks: A Systems Approach, Morgan Kaufmann, Los Altos, CA, 1996.



Cathy H. Xia received her Ph.D. in Operations Research from Stanford University in 1998. She then joined IBM T.J. Watson Research Center, where she is currently a Research Staff Member in the System Analysis and Optimization group. Her research interests are in the areas of control and performance analysis of computer and communication networks, queuing theory, statistical modeling, stochastic scheduling and optimization of Internet service and applications.



George Michailidis received his Ph.D. in Mathematics from UCLA in 1996. He was a post-doctoral fellow in the Department of Operations Research at Stanford University from 1996 to 1998. He joined the Department of Statistics at The University of Michigan in 1998, where he is currently an Assistant Professor. His research interests are in the areas of stochastic network modeling and performance evaluation, queuing analysis and congestion control and statistical modeling and analysis of Internet traffic.



Nicholas Bambos received his Ph.D. in Electrical Engineering and Computer Sciences (EECS) from U.C. Berkeley in 1989. He is currently an Associate Professor at Stanford University, having a joint appointment in the Department of Management Science and Engineering and the Department of Electrical Engineering. His current research interests are in the areas of high-performance network architectures and bandwidth management, computer network modeling and performance evaluation, queuing analysis and congestion control, and Internet service management.