Mitigating the effects of uncertainty in Join-the-Shortest-Queue resource allocation schemes through Prioritized Dispersal*

EXTENDED ABSTRACT

Yitzhak Birk and Noam Bloch
Electrical Engineering Department
Technion — Israel Inst. of Technology, Haifa 32000, Israel
birk@ee.technion.ac.il, Tel. (04) 829-4637

July 14, 1998

Abstract

Balancing the load among multiple resources is critical for their efficient utilization. It has been shown in various contexts that very good balancing is achieved by assigning each arriving task to the resource with the shortest queue among all queues or even among a randomly chosen small subset of the queues. There are, however, factors that limit the performance of such JSQ schemes in practical systems; these include untimely state information and the unknown service time of the queued tasks. In the absence of any queue-state information, a different approach, prioritized dispersal, can be effective. There, more resources than are needed are requested, but the extra requests are assigned a low priority; this represents selective exploitation of redundancy based on partial but current information. In this paper, we present a combined scheme, JSQ-PD, whereby a high-priority request is submitted to the (supposedly) shortest queue, and low-priority redundant requests are submitted to other queues. Analysis of the case of duplicate requests shows that the new hybrid scheme substantially outperforms JSQ in the case of untimely queue-length information or significant variance in service time, and is never inferior to JSQ. Possible applications of this approach are numerous, including parallel computers, operating systems and communication networks.

*Submitted to the Allerton conf., 1998

1. Introduction

An attractive way to increase the **potential** performance of a system is to use parallel architectures. By so doing, one can take advantage of the most cost-effective technology; moreover, fault-tolerance is facilitated. However, the potential performance can be fully realized only if the load is equally apportioned among the multiple resources. These may be stateless, such as processors and communication paths, so any resource can be used for any task; in other cases, such as storage devices and memory banks, the resources are stateful, in which case the choice among them is limited. The issue of load balancing has been studied extensively for a variety of applications.

Intuitively, the load is balanced by allocating tasks to unloaded resources or by transferring tasks from loaded resources to less loaded ones. In many cases, however, transferring tasks is costly or even impossible. The focus of this paper is on allocation schemes without reallocation.

1.1 "Join the shortest queue" (JSQ) policies

In its purest form, this scheme entails assigning an arriving task to the resource with the least number of tasks in its queue. In practice, the choice may be restricted by the fact that not every resource can carry out every task; examples include the choice of a cell site by a mobile communication unit [1] and mirrored disk arrays. Moreover, even if this restriction does not apply, the cost of discovering the queue length of all resources may be prohibitive, leading to a voluntary restriction on the choice. Mitzenmacher [2] and Vvedenskaya [3] showed that even such limited load balancing suffices for obtaining an exponential improvement in the decay of the tail of the queue-length probability density function when compared with the assignment of a task to a randomly-chosen server.

Performance can often be enhanced by permitting a task to be partitioned such that multiple resources can jointly serve it. Partitioning can also be viewed as an enabler for finer load balancing, since an unequal number of subtasks can be assigned to different resources if desired.

Yet another version of JSQ-based load balancing was presented in [4] and [5] for redundant disk arrays: at recording time, a block of data is partitioned into m subblocks, and r redundant ones are computed from those, such that any m of the $m+r$ suffice for reconstruction. These are then stored in $m+r$ different ("randomly" chosen) disk drives. When a block is requested, any m of the $m+r$ disks containing the subblocks may be accessed. Here, the choice is more restricted than in the original scheme. However, since stateful resources are involved, this permits an interesting trade-off between (storage) overhead and flexibility. [4] further addresses the cost of accessing "redundant" blocks instead of the original ones in the context of a video-on-demand server. It introduces *selective exploitation of redundancy*, which weighs the benefit of load balancing against the extra cost (processing and buffer space) when deciding which subblocks to read.

JSQ has been proposed for numerous applications, including multiprocessor systems [6][7][8], disk arrays [5][4], and communication networks [9] [10]. While it is a good technique, several factors may limit JSQ's effectiveness in practical situations:

- The queue-length information may be outdated due to physical limitations or the prohibitive overhead associated with constantly updating it (e.g., [11]).

- The service time for each queued task is often unknown. In this case, shortest queue does not necessarily imply least-loaded resource.
- A resource providing service may be stalled, awaiting information from other resources, so even the net service time for a task is not necessarily an accurate measure.

1.2 Prioritized dispersal [12]

In multipath communication networks, the state of the resources (network links) is never fully known. Maxemchuk [13] proposed *dispersity routing*, whereby a message is partitioned into several (m) submessages, which are then sent to the destination over the different paths in a store-and-forward network. Dispersal assists in balancing the load among paths. Also, the different submessages can be transmitted and propagated concurrently, thereby reducing latency. However, all submessages must be received before the message can be reconstructed, so the delay along the slowest path determines the delay of the message.

To overcome the "weakest link" sensitivity of dispersity routing, Maxemchuk also suggested *redundant dispersity routing*: a number (r) of additional, "redundant" submessages are derived from the original ones and are also transmitted, and any sufficiently large subset of submessages suffices for the reconstruction of the original message. Later, Rabin [14] suggested the *Information Dispersal Algorithm (IDA)*, which implements redundant dispersal, and analyzed it in the context of interconnection networks such as a hypercube.

Redundant dispersal can tolerate some loaded paths, but increases the overall load, reduces capacity and may thus even increase delay and loss probability. This problem can be mitigated by using the recently-proposed *prioritized dispersal* scheme [12], whereby "redundant" submessages receive lower priority than the "original" ones and do not interfere with them; moreover, the use of non-FCFS queuing policies for the redundant submessages leads to the timely arrival of at least a fraction of them even under heavy load.

Casting prioritized dispersal in generic terms, it entails partitioning a task into m subtasks along with r redundant subtasks, and assigning the tasks to a randomly-chosen set of $m+r$ resources. While no knowledge pertaining to the load on the resources is available at assignment time, the assignment of low priority to the redundant subtasks acts at each resource to exploit it when available while deferring to "original" subtasks when it is loaded. This helps balance the load among the resources with a bias toward fair treatment of the different tasks. Clearly, this scheme is imprecise, but the partial information that does get used (implicitly) is current, and analysis shows it to offer substantial advantages.

JSQ and prioritized dispersal represent different approaches to helping balance the load. The main contribution of this paper is JSQ-PD, a family of schemes that combine the two approaches. Using a simple member of the family, we show that this hybrid approach may offer substantial benefits.

The remainder of the paper is organized as follows. In section 2., we describe the JSQ-PD family of schemes. In section 3., we describe and analyze a specific scheme. Section 4. presents a numerical comparison among schemes, and section 5. offers concluding remarks.

2. The JSQ-PD family of schemes

Join the Shortest Queue and Prioritized Dispersal (PD) pursue complementary approaches for load balancing. JSQ exploits state information, but must do so at an early stage (task arrival time). PD, in contrast,

employs an initially random decision; however, the additional low-priority tasks that are assigned to resources permit it to take advantage of those when they are available (without causing any harm). This "decision" whether or not a low-priority task receives service, albeit distributed, imprecise and implicit, is postponed and thus benefits from more current state information at the resources.

Motivated by this observation, we propose to combine the two schemes: the initial decision is based on the available state information, but redundant, low-priority tasks are assigned to additional, presumably busier, servers. The simplest form is task replication, but the same idea can be applied in conjunction with any of the aforementioned JSQ or dispersal schemes.

We next describe a specific JSQ-PD scheme, present a queuing model and use it to analyze the scheme's performance in terms of delay distribution.

3. Analysis of a simple JSQ-PD scheme

3.1 The (1,1)JSQ-PD Scheme

For each arriving task, two resources are picked at random (from among the available resources) and considered. One copy of the task is assigned a high priority and is submitted to the resource whose queue is believed to be shorter, and a second copy is assigned a low priority and submitted to the resource whose queue is believed to be longer. In the case of equality or lack of queue-state information, the assignment is random. Service is provided by a resource based on priority. H-P tasks are served FCFS, and L-P ones are served LCFS.

3.2 Queuing model

We model the set of parallel resources as a system of N parallel queues, one per resource. The two copies of a task are assigned to two randomly chosen queues. N is assumed to be large and the queues are assumed to be i.i.d., so it suffices to analyze a single queue.

The order of service in the queues is FCFS for the high priority submessages and LCFS for the low-priority ones. A preemptive-resume priority discipline is used to favor H-P tasks, and is also applied among low priority tasks. The latter is mainly for simplicity of analysis.

The arrival process of H-P tasks to any single queue is Poisson with rate λ; task service time is distributed exponentially and independently from task to task, with mean $\frac{1}{\mu}$; the resulting load is $\rho \equiv \frac{\lambda}{\mu}$. The same holds for the L-P tasks. We also assume that the H-P and H-P arrival processes to any given queue are independent.

We assume that the task-assignment algorithm does not know whether the state information that it possesses is reliable. We express this questionable reliability as a probability: for any given arriving task, there is probability P_{lb} that the information is reliable, resulting in a correct JSQ decision; with probability $P_{nb} = 1 - P_{lb}$, the information is unreliable and the resulting assignment is effectively random.

We now proceed to analyze a single priority queue with FCFS service for H-P customers and LCFS service for L-P customers. In view of the symmetry and independence among queues, this suffices. It should, however, be noted that the queue states as seen by arriving tasks can become coupled through the

assignment policy. With pure JSQ, for example, it is known that the queue joined by the L-P task is at least as long as that joined by the corresponding H-P task. We begin the analysis by computing occupancy probabilities, and then use those to compute the distribution of delay.

3.3 Occupancy probabilities for the $(1,1)$JSQ-PD scheme

Mitzenmacher [2] and Vvedenskaya [3] considered a system in which tasks are assigned to the resource with the shortest queue among d resources chosen randomly from among $N >> d$ resources. They do not assume that queues are independent (but the results obtained are asymptotically the same as those obtained for independent queues). They analyzed such a system with Poisson arrivals of tasks and exponentially distributed service times, and showed that the decay of the tail of the occupancy probability function is doubly exponential. This is an exponential improvement over the non-balanced case, wherein each task is assigned to a randomly chosen server. In [2], the analysis was extended to the case wherein reliable queue-length information exists and is used with probability P_{lb}; with probability $P_{nb} = 1 - P_{lb}$, such information is unavailable and the assignment is made randomly. (We use "nb" and "lb" to denote no balancing (random assignment) and load balancing (JSQ), respectively.)

Our analysis of $(1,1)$JSQ-PD uses the fact that L-P tasks do not interfere with the H-P ones. Consequently, the occupancy probabilities for H-P tasks is equal to the occupancy probabilities calculated by [2] and [3] for a single priority level.

Let L_i be the probability for at least i high-priority tasks in a randomly chosen queue. By [2],

$$L_i = \rho L_{i-1}\left(P_{nb} + (1-P_{nb})L_{i-1}\right); \quad L_0 = 1, \qquad 0 \le p_{nb} \le 1.$$

For pure JSQ (information always available), this becomes

$$L_i = \rho^{2^i-1}, \qquad p_{nb} = 0.$$

P_i, the probability for i high-priority tasks in a randomly chosen queue is simply

$$P_i = L_i - L_{i+1}.$$

With an LCFS–preemptive-resume queueing discipline for the L-P tasks, the delay of a tagged L-P task depends only on the number of H-P tasks that it sees upon arrival and on the number of H-P and L-P tasks that arrive (subsequently) before its service ends. The number of L-P tasks found in the queue at the tagged L-P task's arrival time does not affect its delay because they are not served until the tagged task's service ends.

Let $P^H(k|lb)$ and $P^H(k|nb)$ denote the probability that an arriving H-P task joins a queue with k H-P customers given that it chose the shortest among the two queues or a queue at random, respectively. Also, let $P^L(j|lb,k)$ and $P^L(j|nb,k)$ denote the probability that an arriving L-P task joins a queue with j H-P tasks given that its H-P partner, having chosen the shorter among two queues or one of them at random, respectively, joined a queue with k H-P customers.

Obviously, when the queues are chosen at random,

$$P^H(k|nb) = P_k, \quad k \ge 0 \tag{1}$$

and

$$P^L(j|nb,k) = P_j, \quad j \geq 0. \tag{2}$$

When the H-P task joins the shortest queue,

$$P^H(k|lb) = 2P_k \cdot L_{k+1} + P_k^2 = 2P_k \left(1 - \sum_{j=0}^{k} P_j\right) + P_k^2, \quad k \geq 0 \tag{3}$$

and

$$P^L(j|lb,k) = \frac{P_j}{L_k} = \frac{P_j}{1 - \sum_{j=0}^{k-1} P_j}, \quad j \geq k \tag{4}$$

3.4 Delay Distribution

The delay incurred by a task is the minimum of the delays incurred by its copies. In this section we derive $D(t)$, the probability that this delay is smaller than t. Let $D^H(t|k,nb)$ and $D^H(t|k,lb)$ denote the probability that the delay of an H-P task is smaller than t given that it joins a queue with k H-P tasks based on JSQ or random assignment, respectively. Also, let $D^L(t|k,nb)$, $D^L(t|k,lb)$ denote the probability that the delay of an L-P task is smaller than t given that its H-P partner joins a queue with k H-P tasks based on JSQ or random assignment, respectively. Then,

$$D(t) = P_{nb} \sum_{k=0}^{\infty} P^H(k|nb) \cdot D(t|k,nb) + P_{lb} \sum_{k=0}^{\infty} P^H(k|lb) \cdot D(t|k,lb), \text{ where}$$
$$D(t|k,nb) = 1 - (1 - D^H(t|k,nb)) \cdot (1 - D^L(t|k,nb)) \text{ and}$$
$$D(t|k,lb) = 1 - (1 - D^H(t|k,lb)) \cdot (1 - D^L(t|k,lb)).$$

The delay of an H-P task that joins a queue with k H-P tasks consists of $k+1$ exponentially distributed service times, so

$$D^H(t|k,lb) = D^H(t|k,nb) = \int_0^t \frac{\mu(\mu x)^k \exp^{-\mu x}}{k!} dx.$$

If the H-P task joins a queue with k H-P tasks based on JSQ, its partner L-P task joins a queue with at least k H-P tasks. In the case of a random selection, the arriving L-P task sees the a priori queue length distribution. Consequently,

$$D^L(t|k,nb) = \sum_{i=0}^{\infty} D_i^L(t) P_i \text{ and}$$

$$D^L(t|k,lb) = \sum_{i=k}^{\infty} D_i^L(t) \cdot P^L(i|lb,k).$$

Substituting for $P^L(i|lb,k)$ from (4) yields

$$D^L(t|k,lb) = \sum_{i=k}^{\infty} D_i^L(t) \frac{P_i}{1 - \sum_{j=0}^{k-1} P_j},$$

where $D_i^L(t)$ is the probability that an arriving L-P task that finds i H-P tasks in the queue incurs a delay less than or equal to t.

Derivation of $D_i^L(t)$

To derive the sojourn time (delay) distribution of a low-priority subtask that finds i high-priority tasks in the queue when it arrives, we first define a stochastic process $U(t)$ – the remaining work (at time t) that should be done before the termination of the service of the tagged low priority subtask that arrived at time t_0. The sojourn time of the tagged subtask is the time needed for U to reach zero. $U(t_0^-)$ consists of the residual service time of the H-P subtask in service plus the service time of the high priority subtasks waiting in the queue. $U(t_0)$ includes, in addition, the tagged subtask's service time. If high- or low-priority subtasks arrive before U reaches zero, their service times are added to U. One may notice that, since the system is work conserving, $U(t)$ is independent of the order in which service is granted to those that are served prior to the completion of the tagged L-P subtask's service.

For convenience, we assume the following order of service: first serve the H-P subtask in service, the high priority subtasks that were waiting in the queue at t_0 and the tagged L-P subtask (all together are considered as "the first service", with Laplace transformed time distribution $B_f^*(s)$). Then serve the (high and low priority) subtasks that arrived during the first service and afterward until U reaches zero.

This procedure is the same as the one used to derive the distribution of busy period duration in a queue in which there is exceptional first service ($U(t_0) = B_f^*(s)$ in our case). For such systems, the Laplace transform of the busy period is

$$D_{i_l}^*(s) = B_f^*(s + 2\lambda - 2\lambda Y_0^*(s)), \tag{5}$$

where $Y_0^*(s)$ is the Laplace transform of the busy period in an ordinary M/G/1 queue (with non-exceptional first service) and arrival rate 2λ. It can be calculated as

$$Y_0^*(s) = B^*(s + 2\lambda - 2\lambda Y_0^*(s)) \tag{6}$$