Performance Evaluation 79 (2014) 306–327

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Value driven load balancing

Sherwin Doroudi [a,*], Esa Hyytiä [b], Mor Harchol-Balter [c]

[a] *Tepper School of Business, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, United States*
[b] *Department of Communications and Networking, Aalto University, 00076 Aalto, Finland*
[c] *School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, United States*

ARTICLE INFO

Article history:
Available online 11 July 2014

Keywords:
Task assignment
Server farms
Processor-Sharing
Heterogeneous values
Holding cost
C-MU rule

ABSTRACT

To date, the study of dispatching or load balancing in server farms has primarily focused on the minimization of response time. Server farms are typically modeled by a front-end router that employs a dispatching policy to route jobs to one of several servers, with each server scheduling all the jobs in its queue via Processor-Sharing. However, the common assumption has been that all jobs are equally important or valuable, in that they are equally sensitive to delay. Our work departs from this assumption: we model each arrival as having a randomly distributed value parameter, independent of the arrival's service requirement (job size). Given such value heterogeneity, the correct metric is no longer the minimization or response time, but rather, the minimization of value-weighted response time. In this context, we ask "what is a good dispatching policy to minimize the value-weighted response time metric?" We propose a number of new dispatching policies that are motivated by the goal of minimizing the value-weighted response time. Via a combination of exact analysis, asymptotic analysis, and simulation, we are able to deduce many unexpected results regarding dispatching.

1. Introduction

Server farms are commonplace today in web servers, data centers, and in compute clusters. Such architectures are inexpensive (compared to a single fast server) and afford flexibility and scalability in computational power. However, their efficiency relies on having a good algorithm for routing incoming jobs to servers.

A typical server farm consists of a front-end router, which receives all the incoming jobs and dispatches each job to one of a collection of servers which do the actual processing, as depicted in Fig. 1. The servers themselves are "off-the-shelf" commodity servers which typically schedule all jobs in their queue via Processor-Sharing (PS); this cannot easily be changed to some other scheduling policy. All the decision-making is done at the central dispatcher. The dispatcher (also called a load balancer) employs a *dispatching* policy (often called a *load balancing* policy or a *task assignment policy*), which specifies to which server an incoming request should be routed. Each incoming job is *immediately* dispatched by the dispatcher to one of the servers (this immediate dispatching is important because it allows the server to quickly set up a connection with the client, before the connection request is dropped). Typical dispatchers used include Cisco's Local Director [1], IBM's Network Dispatcher [2], F5's Big IP [3], Microsoft Sharepoint [4], etc. Since scheduling at the servers is not under our control, it is extremely important that the right dispatching policy is used.

* Corresponding author.
E-mail addresses: sdoroudi@andrew.cmu.edu (S. Doroudi), esa.hyytia@aalto.fi (E. Hyytiä), harchol@cs.cmu.edu (M. Harchol-Balter).

http://dx.doi.org/10.1016/j.peva.2014.07.019
0166-5316/© 2014 Elsevier B.V. All rights reserved.

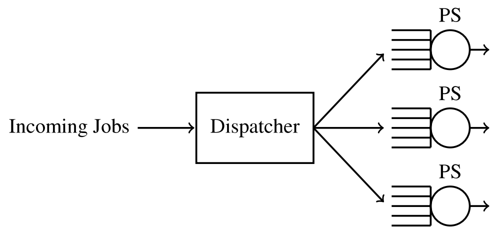

Fig. 1. Dispatching in server farms with Processor-Sharing (PS) servers.

Prior work has studied dispatching policies with the goal of minimizing mean *response time*, $\mathbb{E}[T]$; a job's response time is the time from when the job arrives until it completes. Several papers have specifically studied the case where the servers schedule their jobs via PS (see [5–12]). Here, it has been show that the Join-the-Shortest-Queue (JSQ) policy performs very well, for general job size distributions. Even picking the shortest of a small subset of the queues, or simply trying to pick an idle queue if it exists, works very well. Interestingly, such simple policies like JSQ are superior even to policies like Least-Work-Left, which route a job to the server with the least remaining total work (sum of remaining sizes of all jobs at the queue), rather than simply looking at the number of jobs [13]. In addition, there have been many more papers studying dispatching policies where the servers schedule jobs in First-Come–First-Served (FCFS) order (see e.g., [9,14–25]). Here high job size variability can play a large role, and policies like Size-Interval-Task-Assignment (SITA) [14], which segregates jobs based on job size, or Least-Work-Left [26], which routes job to the queue with the least total remaining work (rather than the smallest number of jobs), are far superior to JSQ.

However, all of this prior work has assumed that jobs have equal importance (value), in that they are equally sensitive to delay. This is not at all the case. Some jobs might be background jobs, which are largely insensitive to delay, while others have a live user waiting for the result of the computation. There may be other jobs that are even more important in that *many* users depend on their results, or other jobs depend on their completion. We assume that every job has a value, V, independent of its size (service requirement). Given jobs with heterogeneous values, the right metric to minimize is not the mean response time, $\mathbb{E}[T]$, but rather the mean *value-weighed response time*, $\mathbb{E}[VT]$, where jobs of higher value (importance) are given lower response times.

The problem of minimizing $\mathbb{E}[VT]$, where V and T are independent, is also not new, although it has almost exclusively been considered in the case of server scheduling, *not in the case of dispatching* (see Prior Work section). Specifically, there is a large body of work in the operations research community where jobs have a *holding cost*, c, independent of the job size, and the goal is to minimizing $\mathbb{E}[c \cdot T]$ over all jobs. Here it is well-known that the $c\mu$ rule is optimal [27]. In the $c\mu$ rule, c refers to a job's holding cost and μ is the reciprocal of a job's size. The $c\mu$ rule always runs the job with the highest product c times μ; thus, jobs with high holding cost and/or small size are favored. However, there has been no $c\mu$-like dispatching policy proposed for server farms.

In this paper, we assume a server farm with a dispatcher and PS servers. Jobs arrive according to a Poisson process and are immediately dispatched to a server. The value, V, of an arrival is known, but its size, S, is not known. Furthermore, we assume that value and size are independent, so that knowing the job's value does not give us information about the job's size. We assume that we know the distribution job values. Furthermore, job sizes are exponentially-distributed with unit mean. By requiring that jobs are exponentially distributed, we are consistent with the assumption that there is no way to estimate a job's size; otherwise, we could use "age" information to update predictions on the remaining size of each job, and some of the policies of interest would become much more complex.[1] Nothing else is known about future arrivals. In making dispatching decisions, we assume that we know the *queue length* at each server (this is the number of jobs at the PS server) as well as the values of the jobs at each server. In this context, we ask:

"What is a good dispatching policy to minimize $\mathbb{E}[VT]$?"

Even in this simple setting, it is not at all obvious what makes a good dispatching policy. We consider several policies (see Section 4 for more detail):

- The **Random (RND)** dispatching policy ignores job values and queue lengths. Arrivals are dispatched randomly.
- The **Join-Shortest-Queue (JSQ)** dispatching policy ignores values and routes each job to the server with the fewest number of jobs. This policy is known to be optimal in the case where all values are equal [5].
- The **Value-Interval-Task-Assignment (VITA)** dispatching policy is reminiscent of the SITA policy, where this time jobs are segregated by *value*, with low-value jobs going to one server, medium value jobs going to the next server, higher-value jobs going to the next server, and so on. The goal of this policy is to isolate high value jobs from other jobs, so that the high value jobs can experience low delay. The distribution of V and system load ρ are used to determine the optimal threshold(s) for minimizing $\mathbb{E}[VT]$.

[1] We do in fact carry out a set of simulations assuming an alternative job size distribution, with policies that ignore "age" information. The qualitative results remain the same as those under exponentially distributed job sizes; see Section 5.