Chapter 3
Optimal Scheduling of Parallel Jobs With Unknown Service Requirements

Benjamin Berg
Carnegie Mellon University, USA

Mor Harchol-Balter
Carnegie Mellon University, USA

ABSTRACT

Large data centers composed of many servers provide the opportunity to improve performance by parallelizing jobs. However, effectively exploiting parallelism is non-trivial. For each arriving job, one must decide the number of servers on which the job is run. The goal is to determine the optimal allocation of servers to jobs that minimizes the mean response time across jobs – the average time from when a job arrives until it completes. Parallelizing a job across multiple servers reduces the response time of that individual job. However, jobs receive diminishing returns from being allocated additional servers, so allocating too many servers to a single job leads to low system efficiency. The authors consider the case where the remaining sizes of jobs are unknown to the system at every moment in time. They prove that, if all jobs follow the same speedup function, the optimal policy is EQUI, which divides servers equally among jobs. When jobs follow different speedup functions, EQUI is no longer optimal and they provide an alternate policy, GREEDY, which performs within 1% of optimal in simulation.*

1. INTRODUCTION

The Parallelization Tradeoff

Modern data centers are composed of a large number of servers, affording programmers the opportunity to run jobs faster by parallelizing across many servers. To exploit this opportunity, jobs are often designed to run on any number of servers (Delimitrou & Kozyrakis, 2014). Running on additional serv-

DOI: 10.4018/978-1-7998-7156-9.ch003

ers may reduce a job's response time, the time from when the job arrives to the system until it is completed. However, effectively exploiting parallelism is non-trivial. Specifically, one must decide how many servers to allocate to each job in the system at every moment in time. We consider the setting where jobs arrive over time, and the system must choose each job's level of parallelization in order to minimize the mean response time across jobs. In choosing each job's server allocation, one must consider the following tradeoff. Parallelizing an individual job across multiple servers reduces the response time of that individual job. In practice, however, each job receives a diminishing marginal benefit from being allocated additional servers. Hence, allocating too many servers to a single job may decrease overall system efficiency. While a larger server allocation may decrease an individual job's response time, the net effect may be an increase in the overall mean response time across jobs. We therefore aim to design a system which balances this tradeoff, choosing each job's server allocation in order to minimize the mean response time across all jobs. It was shown in (Bienia et al., 2008) that many of the benefits and overheads of parallelization can be encapsulated in a job's speedup function, $s\left(k\right)$, which specifies a job's service rate on k servers. If we normalize $s\left(1\right)$ to be 1, we see that a job will complete s(k) times faster on k servers than on a single server. In general, it is conceivable that every job will have a different speedup function. However, there are also many workloads (Bienia et al., 2008) where all jobs have the same speedup function, such as when one runs many instances of the same program. It turns out that even allocating servers to jobs which follow a single speedup function is non-trivial. Hence, we will first focus on jobs following a single speedup function before turning our attention to multiple speedup functions. In addition to differing in their speedup functions, different jobs may represent different amounts of computation that must be performed. For example, if a data center is processing search queries, a simple query might require much less processing than a complex query. We refer to the amount of inherent work associated with a job as the job's size. It is common in data centers that job sizes are unknown to the system { users are not required to tell the system anything about the internals of their jobs when they submit them. Hence, we consider the case where the system does not know, at any moment in time, the remaining sizes of the jobs currently in the system.

The question of how to best allocate servers to jobs is commonly referred to as the choice between fine grained parallelism, where every job is parallelized across a large number of servers, and coarse grained parallelism, where the server allocation of each job is kept small. This same tradeoff arises in many parallel systems beyond data centers. For example, (Berg et al., 2017) considers the case of jobs running on a multicore chip. In this case, running on additional cores allows an individual job to complete more quickly, but leads to an inefficient use of resources which could increase the response times of subsequent jobs. Additionally, an operating system might need to choose how to partition memory (cache space) between multiple applications. Likewise, a computer architect may have to choose between fewer, wider bus lanes for memory access, or several narrower bus lanes. In all cases, one must balance the effect on an individual job's response time with the effect on overall mean response time. Throughout this chapter, we will use the terminology of parallelizing jobs across servers, however all of our remarks can be applied equally to any setting where limited resources must be shared amongst concurrently running processes.