Towards Optimality in Parallel Job Scheduling

Benjamin Berg
Carnegie Mellon University
bsberg@cs.cmu.edu

Jan-Pieter Dorsman
University of Amsterdam
j.l.dorsman@uva.nl

Mor Harchol-Balter
Carnegie Mellon University
harchol@cs.cmu.edu

ABSTRACT

To keep pace with Moore's law, chip designers have focused on increasing the number of cores per chip. To effectively leverage these multi-core chips, one must decide how many cores to assign to each job. Given that jobs receive sublinear speedups from additional cores, there is a tradeoff: allocating more cores to an individual job reduces the job's runtime, but decreases the efficiency of the overall system. We ask how the system should assign cores to jobs so as to minimize the mean response time over a stream of incoming jobs.

To answer this question, we develop an analytical model of jobs running on a multi-core machine. We prove that EQUI, a policy which continuously divides cores evenly across jobs, is optimal when all jobs follow a single speedup curve and have exponentially distributed sizes. We also consider a class of "fixed-width" policies, which choose a single level of parallelization, k , to use for all jobs. We prove that, surprisingly, fixed-width policies which use the *optimal* fixed level of parallelization, k^* , become near-optimal as the number of cores becomes large. In the case where jobs may follow different speedup curves, finding a good scheduling policy is even more challenging. In particular, EQUI is no longer optimal, but a very simple policy, GREEDY*, performs well empirically.

For a full journal version of this paper see [2].

ACM Reference Format:

Benjamin Berg, Jan-Pieter Dorsman, and Mor Harchol-Balter. 2018. Towards Optimality in Parallel Job Scheduling. In *SIGMETRICS'18 Abstracts: ACM SIGMETRICS International Conference on Measurement & Modeling of Computer Systems Abstracts, June 18–22, 2018, Irvine, CA, USA*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3219617.3219666>

1 INTRODUCTION

The Parallelization Tradeoff. Modern multicore chips afford the opportunity to reduce job response time by parallelizing a job across several cores. To exploit this opportunity, modern parallel jobs are often designed to have the ability to run on any number of cores [6, 10]. However, effectively exploiting parallelism is non-trivial. Specifically, for each arriving job one must decide its *level of parallelization* – the number of cores on which the job is run. We consider the setting where jobs arrive over time and ask how to minimize the mean response time across jobs.

In choosing the optimal levels of parallelization, we consider the following *tradeoff*. Parallelizing an individual job across multiple cores reduces the response time of that individual job. In practice,

however, a job's speedup is sublinear and concave in its level of parallelization, leading to an inefficient use of resources and additional system load. Hence, while a higher level of parallelization decreases an individual job's response time, it may have a deleterious effect on overall mean response time across jobs. Thus, while traditionally the programmer chooses the level of parallelization for their jobs, we propose making this a system level decision with the goal of minimizing mean response time across *all* jobs.

Throughout this paper, we use the terminology of parallelizing *jobs across cores*. However, this tradeoff appears in many systems beyond multicore chips. Our remarks can be applied equally to parallelizing jobs across servers in a datacenter, or other settings where resources are shared amongst concurrently running processes.

Our Model

We assume that jobs arrive into a system composed of n cores according to a Poisson Process with rate Λ jobs/second where

$$\Lambda \equiv \lambda n$$

for some λ . The random variable X denotes the *size* of a job. X is the inherent work associated with a job. Any job can be run on any subset of cores. If a job of size X is parallelized across k cores, each of the k job pieces has a service requirement, X_k , where

$$X_k = \frac{X}{s(k)}$$

and $s(k) \leq k$ is called the *speedup factor*. The function $s : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ is assumed to be *sublinear*, *non-decreasing* and *concave*. We will not rely on the specifics of $s(k)$, but for numerical examples we will assume that $s(k)$ is specified by Amdahl's law [8].

In general, it is conceivable that every job will have a different speedup function. However, there are many workloads [3] where all jobs have the same speedup function, for example when one runs multiple instances of the same program. It turns out that even scheduling jobs with a single speedup function is non-trivial. Hence, for the majority of the paper we will focus on a single speedup function before turning our attention to multiple speedup functions.

As jobs arrive, each job is initially dispatched to some subset of the cores. At any moment in time, there may be several jobs (job pieces) running on a particular core. We assume each core schedules jobs according to the *Processor Sharing* (PS) scheduling discipline, which is common for cores in multicore machines.

All jobs in the system are scheduled according to a *scheduling policy*. A scheduling policy defines, at every moment in time, which particular cores each job runs on. Cores are assumed to be homogeneous, and thus any job is capable of running on any core.

The Problem. A job is considered complete when all pieces of the job have been completed. We define the random variable T to be the *response time* of a job – the time between when a job arrives and when it is completed. Our goal is to find and analyze scheduling

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGMETRICS'18 Abstracts, June 18–22, 2018, Irvine, CA, USA

© 2018 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5846-0/18/06.

<https://doi.org/10.1145/3219617.3219666>

policies that minimize the mean response time, $\mathbb{E}[T]$, across all jobs. Clearly, $\mathbb{E}[T]$ depends on the scheduling policy, the speedup function, s , the arrival rate of jobs into the system, Λ , the mean job size, $\mathbb{E}[X]$, and the number of cores, n . We define the average system load, ρ , to be

$$\rho \equiv \frac{\Lambda \mathbb{E}[X]}{n}.$$

We assume both ρ and $s(\cdot)$ are constant over time.

Prior Work. The problem of parallelization has been extensively considered by the SPAA/STOC/FOCS community, however it largely focuses on a *single* job. More recently, this community has considered the effect of parallelization on the mean response time of a stream of jobs, but only through the lens of competitive analysis which assumes the job sizes (service times), arrival times, and even speedup curves are adversarially chosen, e.g. [1, 4, 5].

Contribution 1: Optimal Parallel Scheduling. Our first contribution is an analysis of a scheduling policy first described in [4] called EQUI. We prove that EQUI achieves the minimal mean response time across jobs (see Section 2). However, EQUI requires jobs to be *malleable* – every job can change its level of parallelization while it runs. EQUI exploits this by continuously changing the level of parallelization of jobs as the number of jobs in the system changes.

Contribution 2: Practical Parallel Scheduling. In practice, jobs may not be malleable but are often *moldable* – a job can run on any number of cores, k , but it must run on the same k cores for its entire lifetime. In theory, a scheduling policy could choose a different number of cores on which to run each incoming job. We define a class of extremely simple scheduling policies which run every job on the *same fixed number* of cores. We refer to such policies as *fixed-width* scheduling policies. Section 3 presents one example of a fixed-width scheduling policy, *JSQ-Chunk*, and provides an approximate analysis of mean response time under this policy.

Contribution 3: Fixed-Width Scheduling Can Be Near-Optimal. Fixed-width scheduling policies seem heavily restricted in their ability to effectively exploit parallelism. We prove in Section 4, however, that certain fixed-width policies, like JSQ-Chunk, achieve near-optimal mean response time (equal to that of EQUI) as the number of cores, n , becomes large.

Contribution 4: Multiple Speedup Functions. The above contributions consider the case where all jobs follow the same speedup function $s(\cdot)$. We show that, when jobs are permitted to have different speedup functions, EQUI is no longer optimal. We also show how to numerically compute the optimal policy, OPT, when jobs follow multiple speedup functions (assuming exponentially distributed job sizes). Since finding OPT is computationally intensive, we introduce a simple class of policies called GREEDY, that performs well by maximizing the departure rate of jobs. We prove that the best policy in this class is GREEDY*, which both maximizes the overall departure rate and defers parallelizable work.

2 EQUI

EQUI [4] is a generalization of Processor Sharing to systems with multiple cores. Under EQUI, whenever there are m jobs in the system, each job runs on n/m of the cores. We define $\mu = 1/\mathbb{E}[X]$

to be the rate at which jobs complete when run on a single core. In analyzing EQUI, we assume that we can linearly interpolate along the speedup curve to allow for non-integer levels of parallelization. Hence, when the number of jobs in the system, i , is less than n , jobs complete with rate $i\mu s(n/i)$. When $i \geq n$, jobs complete with rate $n\mu$. Our analysis of EQUI assumes exponentially distributed job sizes, which allows us to represent EQUI via the continuous time Markov chain in Figure 1.

Figure 1: Markov chain representing the total number of jobs under EQUI.

Theorem 1 proves that EQUI is an optimal scheduling policy with respect to mean response time.

Theorem 1. Within our model, assuming jobs are malleable,

$$\mathbb{E}[T]^{\text{EQUI}} \leq \mathbb{E}[T]^P$$

for any scheduling policy P .

PROOF. See [2] Theorem 2.3 for proof. $\square$

3 A PRACTICAL POLICY: JSQ-CHUNK

While EQUI is optimal, it requires jobs to be malleable, which is sometimes impractical. We now focus on a fixed-width policy which requires only that jobs are *moldable* . The JSQ-Chunk policy first determines a fixed level of parallelization, k , to use for all incoming jobs, where k divides n . Each arriving job is dispatched to the k cores with the fewest jobs, similar to the familiar *Join-Shortest-Queue* (JSQ) scheduling policy. JSQ-Chunk additionally partitions the n cores into $c = \frac{n}{k}$ chunks. Each arriving job is assigned to the chunk with the fewest jobs and run across all k cores of the chunk.

Observation 1. Our parallel system with JSQ-Chunk scheduling, n cores, $c = n/k$ chunks, total arrival rate Λ , mean job size $\mathbb{E}[X]$, and level of parallelization k has the same mean response time as a traditional JSQ queueing system with, c queues, a total arrival rate of Λ , and mean job size $\mathbb{E}[X_k]$.

Observation 1 allows us to apply the approximation from [9] for JSQ to derive an approximation for mean response time under JSQ-Chunk. While the approximation from [9] applies to FCFS queues, we can apply it to PS queues by invoking results from [7].

Using the closed-form approximation for mean response time under JSQ-Chunk, we can find the *optimal level of parallelization*, k^* , which minimizes this approximation. Figure 2 shows both mean response time and k^* for JSQ-Chunk as a function of the system load, ρ , where k^* is found numerically. This figure illustrates the aforementioned parallelization tradeoff – the optimal level of parallelization decreases as system load increases, since the system becomes less tolerant of the load added by parallelization.

4 NEAR-OPTIMALITY OF JSQ-CHUNK

As expected from Theorem 1, Figure 2 shows that the mean response time under EQUI forms a lower bound on the mean response time under JSQ-Chunk. However, we see that EQUI is only *slightly* better

Figure 2: Mean response times under JSQ-Chunk (both analysis and simulation) as compared with analysis of the EQUI model (the thick line) when $n = 64$. We assume a speedup curve of Amdahl's law with a parameter of $f = 0.5$ [8] and exponentially distributed job sizes with mean $\mathbb{E}[X] = 1$. We see that JSQ-Chunk with the optimal k^* is not far from EQUI.

than JSQ-Chunk. This is surprising given that JSQ-Chunk is a fixed-width policy. In fact, this gap decreases with higher n (not shown). This leads us to believe that, as n increases, EQUI is effectively choosing an *average* level of parallelization which is equal to k^* . Theorem 2 below formalizes this intuition.

Theorem 2. (Rough Statement) For any fixed $k^* > 1$, as n becomes large, the mean response time under JSQ-Chunk becomes close to the mean response time under EQUI.

PROOF. See [2] Theorem 4.2 for full statement and proof. $\square$

5 MULTIPLE SPEEDUP FUNCTIONS

Thus far, we have assumed that jobs are homogeneous with respect to speedup. In practice, however, it may be the case that there are two or more *classes* of jobs, each with a unique speedup function reflecting the amount of sequential work, number of IO operations, and communication overhead that its jobs will experience when run across multiple cores.

Observation 2. In the case of multiple speedup functions, EQUI is no longer optimal. EQUI's optimality stems from the fact that it maximizes the rate of departures in every state when jobs follow a single speedup function. When jobs are permitted to have different speedup functions, maximizing the rate of departures will require allocating more cores to the more parallelizable jobs and fewer cores to the less parallelizable jobs.

Since EQUI fails to maximize the rate of departures when there are multiple speedup functions, we propose the GREEDY class of policies – the class of policies which maximize the total rate of departures in every state. GREEDY is truly a *class* of policies since, in a given state, there may be multiple allocations which maximize the total rate of departures. For the case when there are two classes of jobs, we define GREEDY*, the policy in GREEDY with the highest rate of departures of the less parallelizable class of jobs in every state. We prove the follow theorem regarding GREEDY*.

Theorem 3. For any GREEDY policy, P ,

$$\mathbb{E}[T]^{\text{GREEDY}^*} \leq \mathbb{E}[T]^P.$$

PROOF. See [2] Theorem 5.1 for proof. $\square$

GREEDY* is the GREEDY policy which always opts to defer parallelizable work when possible. This allows parallelizable work to be done in future states with fewer jobs in the system and thus more cores allocated per job. By deferring parallelizable work, the total rate of departures will be higher in these future states.

Unfortunately, this same phenomenon shows why GREEDY* is *not* the optimal policy. The optimal policy may be willing to achieve a lower total rate of departures in the current state in order to defer parallelizable work for future states. Since the optimal policy (OPT) must consider both current and future states, we derive the optimal policy by formulating the problem as a Markov Decision Process (MDP) and using value iteration. Although GREEDY* is not optimal, we find that it performs within 1% of OPT in a wide range of cases.

6 CONCLUSION

This paper presents the first stochastic analysis of the optimal scheduling policy, EQUI, when parallelizable jobs follow a single speedup function. We find that a fixed-width policy with the optimal level of parallelization, k^* can achieve near-optimal mean response time, close to that of EQUI. When jobs follow different speedup functions, EQUI is no longer optimal. We thus propose another policy, GREEDY* which performs well empirically.

Acknowledgements

The first and third authors were supported by: NSF-CMMI-1538204, NSF-XPS-1629444. Parts of second author's work was done while visiting CMU and while affiliated with Leiden University. This author's research was partly funded by NWO Gravitation project NETWORKS, grant 024.002.003.

REFERENCES

- [1] K. Agrawal, J. Li, K. Lu, and B. Moseley. 2016. Scheduling Parallelizable Jobs Online to Minimize the Maximum Flow Time. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '16)*. ACM, New York, NY, USA, 195–205. <https://doi.org/10.1145/2935764.2935782>
- [2] B. Berg, J.P. Dorsman, and M. Harchol-Balter. 2018. Towards Optimality in Parallel Scheduling. *ACM POMACS* 1, 2 (2018).
- [3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*. ACM, New York, NY, USA, 72–81. <https://doi.org/10.1145/1454115.1454128>
- [4] J. Edmonds. 1999. Scheduling in the dark. *Theoretical Computer Science* 235 (1999), 109–141.
- [5] J. Edmonds and K. Pruhs. 2009. Scalably scheduling processes with arbitrary speedup curves. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*. ACM, New York, NY, USA, 685–692.
- [6] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. 1997. Theory and Practice in Parallel Job Scheduling. In *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '97)*. Springer-Verlag, London, UK, 1–34. <http://dl.acm.org/citation.cfm?id=646378.689517>
- [7] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt. 2007. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation* 64 (2007), 1062–1081.
- [8] J. McCool, M. Robison, and A. Reinders. 2012. *Structured Parallel Programming: Patterns for Efficient Computation*. Elsevier.
- [9] R. D. Nelson and T. K. Philips. 1993. An Approximation for the Mean Response Time for Shortest Queue Routing with General Interarrival and Service Times. *Performance Evaluation* 17 (1993), 123–139.
- [10] S. Srinivasan, S. Krishnamoorthy, and P. Sadayappan. 2003. A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '03)*. 92–99.