The Case for Phase-Aware Scheduling of Parallelizable Jobs

Benjamin Berg, Justin Whitehouse, Benjamin Moseley, Weina Wang, Mor Harchol-Balter
Carnegie Mellon University

1. INTRODUCTION

Parallelizable workloads are ubiquitous and appear across a diverse array of modern computer systems. Data centers, supercomputers, machine learning clusters, distributed computing frameworks, and databases all process jobs designed to be parallelized across many servers or cores. Unlike the jobs in more classical models, such as the M/G/k queue, that each run on a single server, parallelizable jobs are capable of running on multiple servers simultaneously. When a job is parallelized across additional servers or cores, the job receives a speedup and can be completed more quickly.

When scheduling parallelizable jobs, a *scheduling policy* must decide *how to best allocate servers or cores among the jobs in the system at every moment in time.* This paper describes and analyzes scheduling policies for systems that process an online *stream* of incoming parallelizable jobs. Given a set of K servers, we will derive scheduling policies that minimize the *mean response time* across jobs — the average time from when a job arrives to the system until it is completed.

The difficulty in scheduling parallelizable jobs arises largely from the fact that a job's parallelizability is not constant over time. Across a wide variety of systems, jobs typically consist of multiple *phases*, each of which has its own scalability characteristics.

For example, in databases, a single query often alternates between highly parallelizable phases and non-parallelizable phases. Specifically, modern databases translate queries into a pipeline composed of multiple phases corresponding to different database operations [1]. A phase that corresponds to a sequential table scan is *elastic*, capable of perfectly parallelizing and completing k times faster when run on k cores. On the other hand, a phase corresponding to a table join is *inelastic*, receiving a severely limited speedup from additional cores. Figure 1 shows that this phenomenon holds for a variety of queries from the Star Schema Benchmark [4].

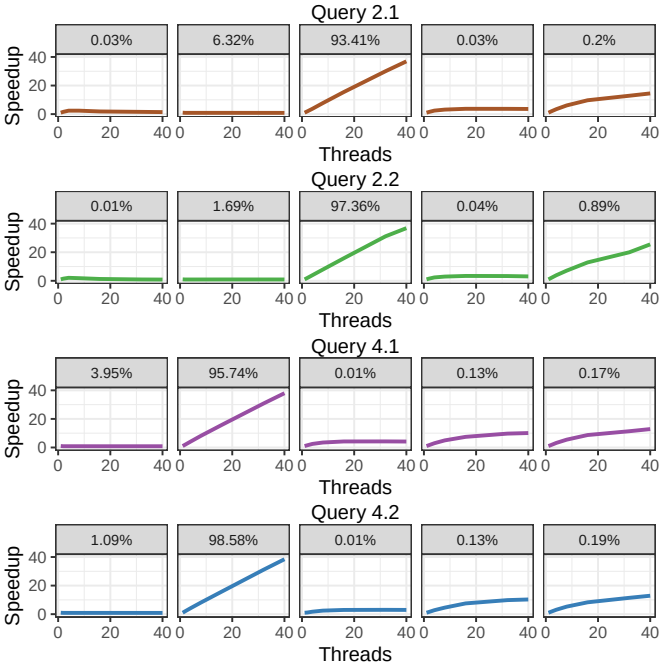

Figure 1: Speedup functions for each phase of four queries from the Star Schema Benchmark. Queries were executed using the NoisePage database [1]. Phases are either elastic (highly parallelizable) or inelastic (highly sequential). The percentages denote the fraction of time spent in each phase when the query was run on a single core. Despite the queries spending most of their time in elastic phases, the overall speedup function of each query is highly sublinear due to Amdahl's law.

Optimal Phase-Aware Scheduling

This paper focuses on the analysis of the Inelastic-First (`IF`) policy first described in [3]. `IF` gives strict priority to jobs in inelastic phases, allocating one server to each inelastic phase and then allocating any remaining servers to the earliest arriving elastic phase. We compare `IF` to several policies from the literature. We consider the Elastic-First (`EF`) policy, also proposed in [3], which gives strict priority to jobs in an elastic phase. We also compare `IF` to a phase-unaware policy called `EQUI` [2], which divides servers evenly among all jobs in the system. Finally, we compare `IF` to a scheduling policy commonly used in databases [1] called Phase-Aware First-Come-First-Served (`PA-FCFS`). `PA-FCFS` allocates servers to jobs in first-come-first-served order, but only allocates a single server to each job in an inelastic phase. If `PA-FCFS` has any servers remaining when it reaches a job in an elastic phase, this phase receives all remaining servers.

Contributions of This Paper

- We first present a novel model of parallelizable jobs composed of elastic and inelastic phases where the scheduler

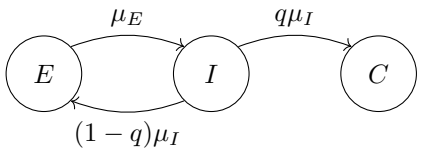

Figure 2: The Markov chain governing the evolution of a multi-phase job when running on a single server. E refers to the elastic phase, I refers to the inelastic phase, and C is the completion state.

knows, at all times, what phase a job is in. Specifically, we assume that the ordering and sizes of phases for each job is governed by a Markov chain as shown in Figure 2.

- We prove that `IF`, which *defers parallelizable work* by giving strict priority to jobs which are in an inelastic phase, is optimal under our model.
- We perform an extensive simulation-based performance evaluation to illustrate that `IF` outperforms a range of scheduling policies. Even in settings that violate the assumptions of our model, `IF` can perform nearly 30% better than `PA-FCFS` and a factor of 3 better than `EQUI`.
- Lastly, we perform a case study on scheduling in databases where queries consist of elastic and inelastic phases. In this setting, the scheduler sometimes has additional information about each query beyond just the query's current phase. We show how to generalize `IF` to leverage this additional information and improve upon state-of-the-art database scheduling by roughly 50% in simulation. This case study is omitted from this abstract for brevity.

2. OVERVIEW OF RESULTS

In this section, we provide an overview of the theoretical and empirical results in our paper.

2.1 Theoretical Results

At a high level, our main theoretical result states that `IF` is optimal with respect to mean response time. More specifically, we show that the number of jobs completed by any point in time under `IF` stochastically dominates the number of jobs completed by the same time under any other algorithm.

Theorem 1. *Consider a K server system serving multi-phase jobs. The policy* `IF` *stochastically maximizes the number of jobs completed by any point in time. Specifically, for a policy A, let $C_A(t)$ denote the number of jobs completed by time t and let $N_A(t)$ denote the number of jobs in the system at t. Then under any arbitrary arrival time process, $C_{\text{IF}}(t) \geq_{st} C_A(t)$ for all times $t \geq 0$. Consequently, $N_{\text{IF}}(t) \leq_{st} N_A(t)$ for all times $t \geq 0$.*

Theorem 1 provides far-reaching results about job response time. In particular, if the arrival time process is a renewal process[1], we can show that `IF` minimizes the steady-state mean response time. We formalize this idea in the following immediate corollary of Theorem 1.

[1]By a renewal process, we mean the inter-arrival times $t_n - t_{n-1}$ are i.i.d., and that the initial phases of jobs p_n are i.i.d. as well.

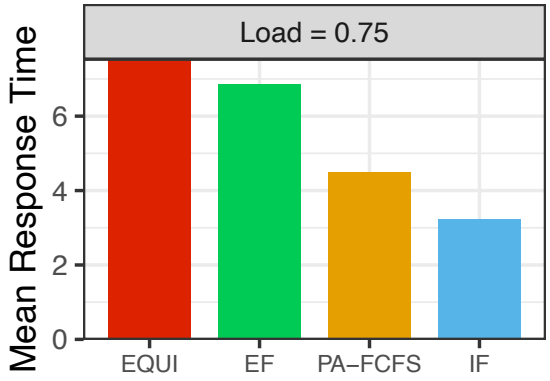

Figure 3: The mean response time of `EQUI`, `EF`, `PA-FCFS`, and `IF` processing a workload consisting of a mixture of 5 queries from the Star Schema Benchmark. We assume Poisson arrivals. `IF` improves upon the next best policy, the `PA-FCFS` policy used in the NoisePage database, by up to 30%.

Corollary 2. *Suppose the same system setup as in Theorem 1. For any arbitrary policy A, let T_A be the steady-state job response time when it exists. If the arrival time process is a renewal process, then $\mathbb{E}[T_{\text{IF}}] \leq \mathbb{E}[T_A]$.*

Theorem 1 and its corollary show that `IF` succeeds by both *deferring parallelizable work* and working on jobs with smaller expected remaining sizes. Specifically, while elastic phases can be completed more quickly by parallelizing across all servers, there are benefits to keeping elastic phases in the system. These elastic phases are flexible and can ensure that all K servers remain utilized. It is also possible to allocate some servers to inelastic phases without significantly increasing the runtime of an elastic phase. Furthermore, jobs in inelastic phases have smaller expected remaining sizes. Hence, deferring parallelizable work also results in favoring shorter jobs. For these reasons, the optimal policy, `IF`, defers as much parallelizable work as possible without over-allocating to inelastic phases.

2.2 Overview of Empirical Results

We performed an extensive simulation-based performance evaluation of `IF` and several other policies described in the literature. Figure 3 shows a comparison of these policies in simulation when processing a workload composed of the database queries shown in Figure 1. Each of the competitor policies suffers from its failure to defer parallelizable work.

3. REFERENCES

[1] NoisePage - The Self-Driving Database Management System. `https://noise.page`.

[2] B. Berg, J.P. Dorsman, and M. Harchol-Balter. Towards optimality in parallel scheduling. *ACM POMACS*, 1(2), 2018.

[3] Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang, and Justin Whitehouse. Optimal resource allocation for elastic and inelastic jobs. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 75–87, 2020.

[4] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. *The Star Schema Benchmark and Augmented Fact Table Indexing*, pages 237—-252. 2009.