Speculation-aware Resource Allocation for Cluster Schedulers

Thesis by

Xiaoqi Ren

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

California Institute of Technology

Pasadena, California

2015

(Submitted September 28, 2014)

Acknowledgements

I am using this opportunity to express my deepest gratitude to my adviser, Prof. Adam Wierman. His excellent guidance, enthusiasm, patience and immense knowledge always inspires me and helps me overcome many obstacles during the past two years.

I am also grateful to my collaborators, Ganesh Ananthanarayanan, Minlan Yu and Michael Chien-Chun Hung. Their enormous help and constructive discussions are what made this thesis possible. The atmosphere and environment of the RSRG, Computer Science department and Caltech are amazing and helpful. It is really a pleasure to work and study here.

Last but not least, I would like to thank my family. I will be grateful forever for their unconditional love and support.

Abstract

Resource allocation and straggler mitigation (via “speculative” copies) are two key building blocks for analytics frameworks. Today, the two solutions are largely *decoupled* from each other, losing the opportunities of joint optimization. Resource allocation across jobs assumes that each job runs a fixed set of tasks, ignoring their need to dynamically run speculative copies for stragglers. Consequently, straggler mitigation solutions are unsure of how to provide for speculative tasks. In this thesis, we propose Hopper, a new *speculation-aware job scheduler*. Hopper dynamically allocates resources for effective speculation based on theoretically derived guidelines that provably minimize average job completion time. In addition, Hopper also provides a simple knob for trading off fairness and performance. Evaluations on a 200-node cluster with Hadoop and Spark prototypes, on production workloads from Facebook and Microsoft Bing, show job speedups of 50% – 70% compared to SRPT schedulers, fairness schedulers, and straggler mitigation algorithms.[1]

[1]The paper is undersubmission to 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’15).

Contents

Chapter 1

Introduction

In data analytics frameworks, application *jobs* are constructed as a DAG of *phases* with each phase running numerous parallel *tasks* (e.g., MapReduce [1], Dryad [2], Spark [3]). The tasks execute on *compute slots* over a large set of machines, and jobs are allocated slots for their tasks by the cluster-wide *job scheduler*.

Job scheduling is a much-studied topic, and there have been many algorithms suggested that seek to schedule the right jobs to run given the limited computing resources (slots) in order to minimize job completion time, e.g., [4, 5, 6, 7] or ensure fairness across jobs, e.g., [8, 9]. These algorithms take the requirements (number of slots) of the jobs when they arrive and allocate slots to them.

Requirements of jobs, however, *change* over time due to *straggler* tasks. Stragglers are those tasks that take significantly longer than expected and they occur commonly in large clusters [10, 11, 12, 1]. In fact, studies have reported nearly one-fifth of tasks straggling in a job [12] and running up to 8× slower [11], leading to significant performance degradation. The dominant solution for mitigating the impact of stragglers is *speculation*, i.e., speculatively running extra copies of tasks that either have already, or are likely to become, stragglers [13, 12, 1, 14], and then picking the earliest of the copies. While speculation policies differ in when to speculate tasks and how many copies to speculate, they all dynamically change the requirements of jobs.

Unfortunately, in all prior work, job scheduling and straggler mitigation have been *decoupled* from each other. Job schedulers assume that each job runs a fixed set of tasks and each task in the job requires a fixed amount of resources. However, given the existence of stragglers, each job may speculate new copies of tasks on the fly to speedup job completion. Straggler mitigation solutions, on the other hand, assume either a fixed amount of resources per job (which can be used for either speculation or scheduling new tasks), or a separate pool of resources that are reserved for speculative tasks.

While such decoupling was assumed for simplicity, it loses important analytical and systemic interactions, leading to significant inefficiencies. If the job scheduler performs best effort speculation by treating speculative tasks as normal ones, it often lacks the urgency required for speculative copies to be effective. However, if the scheduler budgets a fixed amount of resources for speculation, it risks reserving too little so as to not speculate enough tasks, or too much causing wastage of resources.

The main contribution of this thesis is the design of a new *speculation-aware job scheduler* that dynamically allocates resources to ensure effective speculation for each job. It does so to minimize the completion time of jobs. Our design is based on a theoretical model of schedulers built from first principles. Using this model, we derive structural (and simple) design guidelines for *provably optimal* speculation-aware job scheduling.

In particular, based on the theoretical model, we introduce a *virtual size* for every job that includes the "optimal speculation level" for the job. The optimal speculation level is based on the job's distribution of task durations and DAG characteristics. Building on virtual job sizes, we identify different resource allocation strategies based on the availability of slots in the cluster. When there are limited slots, we ensure the smaller jobs are allotted their virtual job sizes. When slots are sufficient, we allocate them proportionate to the virtual sizes of the jobs. Additionally, our theoretical model allows us to trade off performance for fairness using a simple knob.

Based on the design guidelines from our theoretical model, we develop Hopper, a scheduler for cluster frameworks. To our best knowledge, Hopper is the first scheduler that integrates straggler mitigation with job scheduling for data analytics clusters. Hopper uses virtual sizes and the guidelines discussed above to schedule jobs, thus speculating smartly. In addition, Hopper incorporates many practical features of jobs into its scheduling. It estimates the amount of *intermediate* data produced by the job and pipelines their transfer between the job's phases for better network utilization. It also carefully balances *data locality* requirements of tasks while staying faithful to the guidelines. Finally, Hopper is compatible with all existing straggler mitigation strategies (e.g., [13, 12, 14]).

To summarize, we make the following contributions:

1. We identify the importance of *jointly* designing job scheduling and straggler mitigation in clusters.
2. Using a theoretical model from first principles, we develop structural scheduling guidelines for *optimal* speculation-aware job scheduling.
3. We build a speculation-aware job scheduler, Hopper based on the theoretical guidelines and generalize it to consider practical system constraints in clusters.

To evaluate the performance of Hopper, we have developed prototypes inside the Hadoop [15] and Spark [3] computing frameworks. In doing so, our objective is to demonstrate Hopper's generic design and also expose it to many varied factors in scheduling based on task durations, straggler causes and DAG patterns.

We evaluate our prototypes on a 200 node private cluster using workloads derived from Facebook's and Microsoft Bing's production analytics clusters. Our results show that Hopper reduces average completion time of jobs by 50% compared to Shortest Remaining Processing Time (SRPT), which is by far one of the best approaches to minimize completion time. Compared to currently deployed fairness-based schedulers [8, 9], Hopper reduces average job completion time by 70%. It achieves these gains with limited fallout from unfairness: Hopper slows down fewer than 4% of jobs by $\leq 5\%$ compared to fairness-based schedulers.

Chapter 2

Challenges and Opportunities

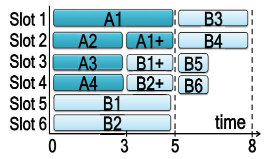
We now present the challenges and opportunities associated with joint decisions on job scheduling and straggler mitigation using simple examples and present some intuitions for joint scheduling decisions.

2.1 Background on Cluster Scheduling

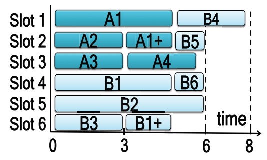
We begin by describing the commonly used and relevant algorithms for job scheduling and straggler mitigation.

Job Scheduling: While job scheduling in clusters is a classical problem, traditional approaches, typically, do not consider stragglers. They assume that each job has running time that is known *a priori*, and thus a job's remaining duration can be estimated from the remaining durations of its running tasks and unscheduled tasks. Perhaps the two most studied job scheduling polices are the following: one aiming to minimize average job duration and the other focusing on fairness.

Shortest Remaining Processing Time (SRPT): SRPT assigns slots to jobs in ascending order of their remaining duration (or equivalently, remaining number of tasks, for simplicity). The reason for the popularity of SRPT is that, in the case of a single server, it is provably optimal in a very strong sense: the number of jobs in the system is minimal at all times and the mean job completion time is also minimized [16]. Further, in the multiple

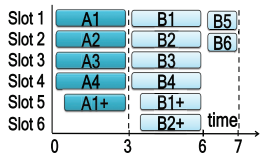


(a) SRPT: completion time for job A and B are 5 and 8.

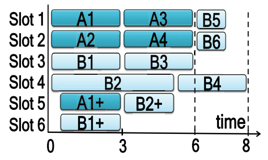


(b) Fair: completion time for jobs A and B are 6 and 8.

Figure 2.1: **Scheduling with "best effort" speculation for jobs A and B. The + suffix indicates a speculative copy.**



(a) SRPT: completion time for job A and B are 3 and 7.



(b) Fair: completion time for jobs A and B are 6 and 8.

Figure 2.2: **Scheduling with "budgeted speculation" (2 slots) for jobs A and B. The + suffix indicates speculation.**

server setting, although SRPT is no longer optimal in as strong a sense, it has the optimal achievable competitive ratio among online algorithms [4]. Of course, unfairness is a concern for SRPT given its prioritization of small jobs at the expense of large jobs.

Fair scheduling: Fairness is a crucial issue in many settings, and as a result, another popular job scheduler is based on fair allocations. Such schedulers *fairly* divide the resources in the cluster (slots, memory, network etc.) among the jobs. Without loss of generality, we focus on the so-called Fair Scheduler, which is commonly used in cluster frameworks today (e.g., [8]). The Fair scheduler allocates the available *compute slots* evenly among all the active jobs. Such strong fairness naturally comes with performance inefficiencies compared to SRPT.

Straggler Mitigation: Straggler mitigation strategies primarily rely on spawning *speculative* copies for stragglers, and pick the result from the earliest among them; the other copies are killed then. In particular, they assume that the job has been allocated its capacity of slots by the job scheduler (based on SRPT, fairness etc.) and then, given this

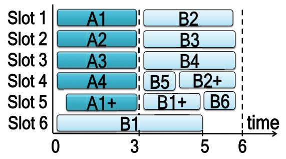
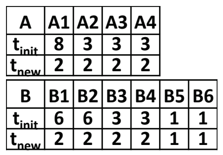


Figure 2.3: Hopper**: completion time for jobs A and B are** 3 **and** 6**, respectively.**



A	A1	A2	A3	A4
t_{init}	8	3	3	3
t_{new}	2	2	2	2

B	B1	B2	B3	B4	B5	B6
t_{init}	6	6	3	3	1	1
t_{new}	2	2	2	2	1	1

Table 2.1: t_{init} **and** t_{new} **are task durations of the original and speculative copies.** $t_{rem} = t_{init}$ **– elapsed time.**

capacity, decides when to schedule speculative copies of tasks. There are many varieties of speculation strategies, e.g., [13, 12, 14], which differ in their decisions of when to spawn speculative copies as well as how many speculative copies to spawn.

For simplicity, we assume the following speculation strategy: if the remaining running time (t_{rem}) is longer than the time to run a new task (t_{new}), speculate a new copy. We also assume that the straggler can be detected after a new task runs for a small amount of time.

2.2 Strawman Approaches

SRPT and Fair scheduling do not explicitly take into account speculation policies. We now explore two natural strawman approaches for them to consider speculative copies. We use the strawman approaches to highlight the fundamental difficulties that motivate the need for dynamic joint decisions of scheduling and speculation.

Best-effort Speculation: Perhaps the most simple approach is to just treat speculative tasks the same as normal tasks. Thus, the job scheduler allocates resources for speculative tasks in a "best effort" manner, i.e., *whenever there is an open slot.* For example, for the SRPT policy that prioritizes the job with the shortest remaining time, a speculative copy for its task is scheduled when a new slot opens; crucially speculative copies are run *only* when new slots become available.

Concretely, in Figure 2.1a, the SRPT job scheduler has to wait till time 3 to find an open slot for the speculative copy of A1.[1] Clearly, this approach is problematic. For

[1]At time 3, when A2 finishes, the job scheduler allocates the slot to job A. This is because based on

example, in Figure 2.1a if the job scheduler had allocated resources to speculative tasks earlier, then job A's completion time would have been reduced, without slowing down job B (see Table 2.1 for the task durations).

The problem is also present with Fair scheduling. In Figure 2.1b, where both jobs are allotted 3 slots, job A can only speculate task A1 when task A2 finishes. If it could have speculated earlier, job A's completion time would have reduced, again, without slowing down job B.

Budgeted Speculation: An alternative approach is to have the job scheduler reserve a fixed "budget" of slots for speculative tasks. In this way, speculative tasks for stragglers do not have to wait and can be run much earlier, thus alleviating the issues discussed above. Budgeting the right size of the resource pool for speculation, however, is challenging because of varying straggler characteristics and fluctuating cluster utilizations. If the resource pool is too small, it may not be enough to immediately support all the tasks that need speculation. If the pool is too large, resource are left idle.

Concretely, in Figure 2.2a, two slots (Slot 5 and Slot 6) are reserved for speculative tasks. Slot 6 is vacant from time 0 to 3. If the job scheduler could have used the wasted resource to run a new task, say B1, then job B's completion time would have been reduced. The problem is also present with Fair Scheduling. In Figure 2.2b, if the job scheduler could have used Slot 6 to run B4 at time 3, then job B's completion time would have reduced.

Finally, note that reserving one instead of two slots will not solve the problem, since both SRPT and Fair scheduling need two reserved slots to run two speculative copies simultaneously at some time.

2.3 Speculation-aware Job Scheduling

The above strawman approaches highlight the difficulties involved in integrating speculation decisions with job scheduling. To that end, the main contribution of this thesis is a

SRPT, job A's remaining processing time is smaller than job B's. Job A speculates task A1 because A1's $t_{rem} = t_{init} - 3 = 5$ is larger than $t_{new} = 2$ (see Table 2.1).

speculation-aware job scheduler that allocates resources to dynamically sufficient capacity to jobs for effective speculation.

Figure 2.3 highlights the intuition behind and value of such scheduling. At time $0-3$, we allocate 1 extra slot to job A (for a total of 5 slots), thus allowing it to speculate task A1 promptly. After time 3, we can dynamically reallocate the slots to job B to optimize its speculation solution and reduce its completion time. As a consequence, the average completion time drops compared to both SRPT and Fair scheduling.

In other words, the performance of both jobs is equal to or better than their performance with fair scheduling despite the lack of fairness guarantees *and* the overall performance is improved relative to SRPT. Further, in this case it can easily be seen that the schedule provided by Hopper is *optimal* (with respect to completion time).

So, the goal of speculation-aware job scheduling distills to: *dynamically allocating slots for speculation based on the distribution of stragglers and cluster utilization, while not deviating too far from fair allocations.*

Chapter 3

Scheduling Guidelines

We formulate an analytical model to capture the interaction between straggler mitigation within jobs and resource allocation across jobs for optimal scheduling of job completion.

3.1 Model design

We focus on a system with S slots, each of which can have one task scheduled to it. Jobs arrive over time and the ith arrival is denoted by J_i has T_i tasks, each of which has an i.i.d. random task size τ. We denote the remaining number of tasks for the ith job at time t by $T_i(t)$. We characterize the service rate (i.e., throughput) of the ith job, $\mu_i(t)$, as a function of how many slots, S_i, it is allocated and the average number of speculative copies per task $k(t)$.

The key piece of our model is the characterization of the service rate of the ith job, $\mu_i(t)$, as a function how many slots, S_i, it is allocated and the average number of speculative copies per task at time t, $k(t)$. Note that $\mu_i(t)$ should be interpreted as the throughput of the ith job. We adopt the following approximation for $\mu_i(t)$, which has been used previous in the design of task level speculation policies by [14].

$$\min(S_i, T_i(t)k(t)) \times \left(\frac{E[\tau]}{k(t)E\left[\min(\tau_1, \ldots, \tau_{k(t)})\right]} \right) \tag{3.1}$$

To understand this approximate model, note that the first term approximates the completion rate of work and the second term approximates the "blow up factor," i.e., the ratio of the expected work completed without speculative copies to the amount of work done with speculative copies. To understand the first term, note that there are $T_i(t)k(t)$ tasks available to schedule at time t, including speculative copies. Given that the maximum capacity that can be allocated is S_i, we obtain the first term in (3.1). The second term computes the "blow up factor," which is the the expected amount of work done per task without speculation ($E[\tau]$) divided by the expected amount of work done per task with speculation ($k(t)E[\min(\tau_1, \tau_2, \ldots, \tau_{k(T_i(t))})]$, since $k(T_i(t))$ copies are created and then they are stopped when the first copy completes.

To specialize (3.1) further, we note that task completion times often show evidence of Pareto tails [14]. So, we focus on the case of Pareto(x_m,β) completion times. Given this form for the task size distributions, the optimal speculation policy has been shown in [14] to be as in (3.4).

$$k(t) = \begin{cases} \frac{2}{\beta}, & \frac{2}{\beta}T_i(t) \geq S_i \\ S_i/T_i(t), & S_i > \frac{2}{\beta}T_i(t); \end{cases} \tag{3.2}$$

Plugging the optimal speculation policy given in (3.4) into the model for $\mu(t)$ in (3.1) yields the following model for the service rate.

$$\mu(t) = \begin{cases} \frac{\beta^2}{4(\beta-1)}T_i(t), & \frac{2}{\beta}T_i(t) \geq S_i \\ \frac{\beta}{\beta-1}T_i(t) - \frac{1}{\beta-1}\frac{T_i(t)^2}{S_i}, & S_i > \frac{2}{\beta}T_i(t); \end{cases} \tag{3.3}$$

Importantly, the model of this service rate is general enough to provide insight on job level speculation regardless of the underlying task-level speculation policy.

While our analysis focuses on scheduling to maximize throughput (service rate), improving throughput usually corresponds to improvements in response time, especially in settings where systems are moderately or heavily loaded since improving throughput enlarges the capacity region for the system.

While our focus in this thesis is on scheduling to minimize completion times, the model described above is not well suited toward analytical results about that metric. Instead, our analysis focuses on scheduling to maximize throughput. Of course, improving throughput usually corresponds to improvements in response time, especially in settings where systems are moderately or heavily loaded since improving throughput enlarges the capacity region for the system.

It is natural to follow this approach when studying stragglers because replication pushes the system toward high loads and is fundamentally about trading off increased resource demands for improved performance. Importantly, our experimental results show that the design motivated by the analysis that follows does indeed result in considerable response time improvements.

3.2 Model Features

Our model incorporates both straggler mitigation policies per job (similar to [14]) as well as inter-job resource allocation to study the optimal job scheduler. Important features of jobs, like heterogeneous straggler behavior and DAGs of tasks, are included.

However, given the complexity of cluster scheduling, the model is necessarily simplistic in order to allow for analytic tractability. In particular, many important issues are ignored. For example, data locality is not considered. Additionally, it is assumed that the scheduler has perfect knowledge of the remaining work in jobs and that the allocation of slots to jobs can be adjusted dynamically at every point in time. Because of these simplifications, one should interpret the analytic results as providing guidelines for system design, which then need to be adjusted given practical factors that are excluded from the model. We discuss how these practical factors are handled in Hopper's system design in §5.

The optimal job scheduler is viewed as a dynamic resource allocation scheme, where each job is allocated (at each time) some fraction of the slots based on a combination of the remaining number of tasks in the job and some job-specific properties (e.g., the job's task size distribution and the job's DAGs of tasks). The key feature of Hopper is the careful determination of the interplay of these properties in order to ensure that the inefficiencies in the strawman solutions of §2.2 do not occur.

There are two key design components that the analytic results highlight: First, is the notion of a "virtual job size", which we use to quantify the impact that job-specific factors like stragglers, the DAG of tasks, etc., have on the optimal speculation level for a given job (§3.3). Second, is the impact of cluster utilization on the capacity allocation. It turns out that very different scheduling rules should be used depending on the number of available slots in the cluster and the virtual job sizes (§3.4). Finally, we present a simple mechanism to trade performance for fairness in §3.5.

We focus on homogeneous single-phased jobs in this section and handle heterogeneous DAGs of tasks in §4.

3.3 Virtual Job Sizes

A crucial aspect of speculation-aware job scheduling is an understanding of how much speculation is necessary for a given job. The idea of a "virtual job size" captures the fact that the "true" size of a job is really the job itself plus the speculative copies that will be spawned. It is this combined "virtual job size" that is crucial for determining how to divide capacity across jobs.[1]

A key observation is that the "optimal number of speculative copies", on average, for the tasks in a job is a function of the magnitude of the stragglers (i.e., the distribution of task durations) and the available compute slots (or cluster utilization). Thus, the expected "optimal level of speculation" can be derived analytically in terms of these factors.

To derive this optimal level of speculation, we assume that task durations follow a Pareto distribution, which is based on the production traces in Facebook and Microsoft Bing [14]. The Pareto tail parameter β represents the likelihood of stragglers. Roughly,

[1]Of course, straggler mitigation strategies typically spawn speculative copies for a task only after observing its performance for a short duration. We ignore this observation duration as it is relatively negligible to the task's duration.

when β is smaller, it means that if a task has already run for some time, there is higher likelihood of the task continuing to run longer. Typically, production traces suggest that $\beta < 2$, and so we make that assumption in our analysis.

Given that task durations have Pareto (β) tails with $\beta < 2$, our analytic model shows that the optimal (average) speculation per task of a job is given by the following, where S_i is the number of slots allocated to job i and $T_i(t)$ is the remaining number of tasks of the job.

$$\begin{cases} \frac{2}{\beta}, & \frac{2}{\beta}T_i(t) \geq S_i \\ S_i/T_i(t), & S_i > \frac{2}{\beta}T_i(t); \end{cases} \tag{3.4}$$

Equation 3.4 can be interpreted as saying that the optimal (average) level of speculation for a job is $2/\beta$, which ensures that if stragglers are likely to be long (i.e., β is small), then more speculation is used. The first case in Equation 3.4, which corresponds to the early set of tasks, shows that the optimal level of speculation should not be sacrificed even when the system is capacity constrained (i.e., when not all tasks can be scheduled). However, the equation also highlights that during the last set of tasks of a job (second case in Equation 3.4), it should not leave slots unused. So, it should speculate aggressively to make use of the capacity available.

Given Equation 3.4, it is natural to think of $2/\beta$ as the optimal level of speculation that a job would like to maintain. And thus, we define the *virtual remaining size* of a job as its number of remaining tasks multiplied by the "optimal speculation level".

$$V_i(t) = \frac{2}{\beta}T_i(t) \tag{3.5}$$

A nice consequence of defining the virtual size of a job is the *decoupling* of the speculation decisions from the allocation of slots to jobs. Note that the virtual size of a job dynamically changes as its tasks finish.

3.4 Dynamic Resource Allocation

Given the virtual job sizes (i.e., how much capacity a job needs to perform optimal speculation), the next question is how to allocate resources across jobs. There are two distinct cases one must consider: (i) How should slots be allocated if there are not enough slots to give every job enough space to perform optimal speculation? (ii) How should slots be allocated if there are more than enough slots to give every job enough space to perform optimal speculation. In (i) the sum is more than the number of slots, while in (ii) the sum of the virtual sizes is less than the number of slots.

(i) Resource allocation when the system is capacity constrained: If there are not enough slots to give every job enough space to perform optimal speculation, then the key design challenge is to decide how much capacity to trim from the desired allocations of each job. There are many options for how to do this. For example, one could give the limited resources to a few jobs and allow them to maintain the optimal level of speculation, or one could give all jobs some sub-optimal amount of resources to avoid starving any of the jobs. Of course, there are also lots of strategies in between these extremes.

Our analytic results highlight that the job scheduler should give as many jobs as possible their optimal speculation level, i.e., their full virtual job size. Thus, the scheduler should start with the job with the smallest virtual job size $V_i(t)$ and work its way to larger jobs giving all the jobs the optimal level until capacity is exhausted.

Guideline 1. *If there are not enough slots for every job to maintain its optimal level of speculation, i.e., a number of slots equal to its virtual size, then slots should be dedicated to the smallest jobs and each job should be given a number of slots equal to its virtual size.*

This guideline is similar to the spirit of SRPT, however (unlike SRPT) it crucially pays attention to the optimal speculation level of jobs when allocating capacity. As the examples in §2 highlight, this leads to improved performance. Note that prioritizing small jobs may lead to unfairness for larger jobs. We discuss this issue in §3.5.

```
procedure Hopper(⟨Job⟩ $J$, int $S$, float $\beta$)
    totalVirtualTasks ← 0
    for each Job $j$ in $J$ do
        $j.\mathrm{V}_{\mathrm{rem}} = (2/\beta)\, j.\mathrm{T}_{\mathrm{rem}}$
                                        ▷ $j.\mathrm{T}_{\mathrm{rem}}$: remaining number of tasks
                                        ▷ $j.\mathrm{V}_{\mathrm{rem}}$: virtual remaining number of tasks
        totalVirtualTasks += $j.\mathrm{V}_{\mathrm{rem}}$
    SortAscending($J$, $\mathrm{V}_{\mathrm{rem}}$)
    if $S <$ totalVirtualTasks then
        for each Job $j$ in $J$ do
            $j.\mathrm{slots} \leftarrow \lfloor \min(S, j.\mathrm{V}_{\mathrm{rem}}) \rfloor$
            $S \leftarrow \max(S - j.\mathrm{slots}, 0)$
    else
        for each Job $j$ in $J$ do
            $j.\mathrm{slots} \leftarrow \lfloor (j.\mathrm{V}_{\mathrm{rem}}/\mathrm{totalVirtualTasks})\, S \rfloor$
```

Pseudocode 1: **Hopper (simple version) for jobs in set J with S slots in the cluster and shape parameter β.**

(ii) Resource allocation when the system is not capacity constrained: If there are more than enough slots to give every job enough space to perform optimal speculation, then the key design challenge becomes how to divide the extra capacity among the jobs present. There are many options for how to do this. For example, the scheduler could give all the extra slots to a few jobs in order to complete them very quickly, or the scheduler could split the slots evenly across jobs. Of course, there are many other options between these extremes.

Our analytic results highlight that the job scheduler should do a form of *proportional sharing* to determine the allocation of slots to jobs. Specifically, jobs should be allocated slots proportionally to their virtual job sizes, i.e., job i receives

$$\left(\frac{V_i(t)}{\sum_j V_j(t)}\right) S = \left(\frac{T_i(t)}{\sum_j T_j(t)}\right) S \text{ slots,} \tag{3.6}$$

where S is the number of slots available in the system. In the above we have assumed $V_i(t) = (2/\beta)T_i(t)$, as discussed above.

Guideline 2. *If there are enough slots to permit every job to maintain its optimal level of* *speculation, i.e., a number of slots equal to its virtual size, then the slots should be shared "proportional" to the virtual sizes of the jobs.*

Note that this guideline is different in spirit from SRPT – large jobs get allocated more slots than small jobs. The reason for this is that every job is guaranteed the optimal level of speculation already. Extra slots are more valuable for large jobs due to the fact that they are likely to incur more stragglers. Importantly, this prioritization of large jobs helps to reduce the unfairness large jobs experience due to Guideline 1.

Algorithm 1 (Hopper, single phased job)**.**
Let $J(t) = \{J_1, J_2, \ldots, J_n\}$ denote the jobs in the system at time t sorted in increasing order of remaining tasks, so $T_1(t) \leq \ldots \leq T_n(t)$.

1. *If $\frac{2}{\beta}\sum T_i(t) \geq S$, then assign $S_i = \frac{2}{\beta}T_i(t)$ to jobs in order from $i = 1$ to n until no slots remain and assign $S_i = 0$ for all remaining jobs.*
2. *If $\frac{2}{\beta}\sum T_i(t) < S$, the assign $S_i = \left(\frac{T_i(t)}{\sum T_j(t)}\right) S$ for all jobs $J_i \in J(t)$.*

Summary: Algorithm 1 (also see Pseudocode 1) combines the above two guidelines for homogeneous single phase jobs. And, we have the following theorem.

Theorem 1. *Algorithm 1 is throughput maximal for single-phased jobs, i.e., it maximizes $\sum \mu_i(t)$.*

Proof. We divide the problem into two cases based on the relationship of the total number of slots, S, and the sum of remaining number of tasks for all jobs, $\sum T_i(t)$.

Case 1: $S \leq \frac{2}{\beta}\sum T_i(t)$

If we assign slots more than its optimal speculation level to job J_i, the throughput for job J_i is,

$$\begin{aligned}
&\frac{\beta}{\beta-1}T_i(t) - \frac{1}{\beta-1}\frac{T_i(t)^2}{S_i} \\
=&\frac{1}{\beta-1}\frac{1}{S_i}\left(-T_i(t)^2 + \beta T_i(t)S_i - (\frac{\beta S_i}{2})^2\right) + \frac{1}{\beta-1}\frac{1}{S_i}(\frac{\beta S_i}{2})^2 \\
=&-\frac{1}{(\beta-1)S_i}(T_i(t) - \frac{\beta S_i}{2})^2 + \frac{\beta^2}{4(\beta-1)}S_i \\
\leq&\frac{\beta^2}{4(\beta-1)}S_i.
\end{aligned} \tag{3.7}$$

The above inequality implies that if any job that is assigned less than optimal speculation level slots, then, no job should get more than its optimal speculation level slots. In other words, when $\frac{2}{\beta}\sum T_i(t) \geq S$,optimal speculation scheduling should assign no more than $\frac{2}{\beta}T_i(t)$ to every job $J_i \in J(t)$. To minimize the total completion time, since $T_1(t) \leq T_2(t) \leq \ldots \leq T_n(t)$, from SRPT, we should always satisfy the need for small jobs, i.e., assign $\frac{2}{\beta}T_i(t)$ to jobs in order from $i = 1$ to n, until there is no slots remain.

Case 2: $S > \frac{2}{\beta}\sum T_i(t)$

When $\frac{2}{\beta}\sum T_i(t) \leq S$, denote the set of jobs which get $S_i \leq \frac{2}{\beta}T_i(t)$ by $J_1(t)$ and the set of jobs which get $S_i \geq \frac{2}{\beta}T_i(t)$ by $J_2(t)$. Then, the total throughput is,

$$\begin{aligned}
\sum_{J_1(t)}\mu_i(t) + \sum_{J_2(t)}\mu_i(t) &= \sum_{J_1(t)}\frac{\beta^2}{4(\beta-1)}S_i + \sum_{J_2(t)}\left(\frac{\beta}{\beta-1}T_i(t) - \frac{1}{\beta-1}\frac{T_i(t)^2}{S_i}\right) \\
&= \sum_{J_1(t)}\frac{\beta^2}{4(\beta-1)}S_i + \sum_{J_2(t)}\left(-\frac{1}{(\beta-1)S_i}(T_i(t) - \frac{\beta S_i}{2})^2 + \frac{\beta^2}{4(\beta-1)}S_i\right) \\
&= \frac{\beta^2}{4(\beta-1)}\sum_{J_1(t)+J_2(t)}S_i - \sum_{J_2(t)}\frac{1}{(\beta-1)S_i}(T_i(t) - \frac{\beta S_i}{2})^2 \\
&= \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)\sum\limits_{J_2(t)}S_i}\left(\sum_{J_2(t)}S_i\right)\left(\sum_{J_2(t)}\frac{1}{S_i}(\frac{\beta S_i}{2} - T_i(t))^2\right) \\
&\leq \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)\sum\limits_{J_2(t)}S_i}\left(\sum_{J_2(t)}(\frac{\beta}{2}S_i - T_i(t))\right)^2,
\end{aligned}$$

where the final line follows from the Cauchy-Schwartz inequality.

Next, since $\frac{\beta}{2}\sum\limits_{J_2(t)} S_i = \frac{\beta}{2}S - \frac{\beta}{2}\sum\limits_{J_1(t)} S_i \geq \frac{\beta}{2}S - \frac{\beta}{2}\left(\frac{2}{\beta}\sum\limits_{J_1(t)} T_i(t)\right) = \frac{\beta}{2}S - \sum\limits_{J_1(t)} T_i(t)$, we have

$$\begin{aligned}
\sum_{J_1(t)} \mu_i(t) + \sum_{J_2(t)} \mu_i(t) &\leq \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)\sum\limits_{J_2(t)} S_i}\left(\sum_{J_2(t)}(\frac{\beta}{2}S_i - T_i(t))\right)^2 \\
&= \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)\sum\limits_{J_2(t)} S_i}\left(\sum_{J_2(t)}\frac{\beta}{2}S_i - \sum_{J_2(t)} T_i(t)\right)^2 \\
&\leq \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)\sum\limits_{J_2(t)} S_i}\left(\frac{\beta}{2}S - \sum_{J_1(t)} T_i(t) - \sum_{J_2(t)} T_i(t)\right)^2 \\
&\leq \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)S}\left(\frac{\beta}{2}S - \sum_{J_1(t)} T_i(t) - \sum_{J_2(t)} T_i(t)\right)^2 \\
&\leq \frac{\beta^2}{4(\beta-1)}S - \frac{1}{(\beta-1)S}\left(\frac{\beta}{2}S - \sum_{J(t)} T_i(t)\right)^2
\end{aligned}$$

Equality is obtained when,

1. $\sum\limits_{J_2(t)} S_i = S$
2. $\sum\limits_{J_1(t)} T_i(t) + \sum\limits_{J_2(t)} T_i(t) = \sum\limits_{J(t)} T_i(t)$
3. For all $J_i \in J_1(t), S_i = \frac{2}{\beta}T_i(t)$
4. For all $J_i \in J_2(t)$, $\frac{T_i(t)}{S_i} =$ const., i.e., for all $J_i \in J_2(t)$, $S_i = \frac{T_i(t)}{\sum\limits_{J_2(t)} T_j(t)} \sum\limits_{J_2(t)} S_i$

That is, optimal scheduling satisfies $J_2(t) = J(t)$, and assigns $\frac{T_i(t)}{\sum T_j}S$ slots for any job $J_i \in J(t)$. It follows that if $\frac{2}{\beta}\sum T_i(t) < S$, the optimal scheduling should assign $S_i = \left(\frac{T_i(t)}{\sum T_j(t)}\right)S$ for all jobs $J_i \in J(t)$.

In summary, the optimal scheduling should:

1. If $\frac{2}{\beta}\sum T_i(t) \geq S$, then assign $S_i = \frac{2}{\beta}T_i(t)$ to jobs in order from $i = 1$ to n until no slots remain and assign $S_i = 0$ for all remaining jobs.
2. If $\frac{2}{\beta}\sum T_i(t) < S$, the assign $S_i = \left(\frac{T_i(t)}{\sum T_j(t)}\right) S$ for all jobs $J_i \in J(t)$.

□

3.5 Incorporating Fairness

Fairness is an important constraint on cluster scheduling and, intuitively, the guidelines we have described so far may create unfairness. We extend our guidelines to adapt the notion of fairness currently employed by cluster schedulers today, e.g., [8]: if there are $N(t)$ active jobs at time t, then each job is assigned $S/N(t)$ slots. While this is a natural notion of fairness, it leaves no flexibility for optimizing performance.

To allow some flexibility, while still tightly controlling the unfairness introduced, we define a notion of *approximate* fairness as follows. We say that a scheduler is ϵ-fair if it guarantees that every job receives at least $S/N(t) - \epsilon$ slots at *all* times t. The fairness knob ϵ can be set as a fraction of $S/N(t)$; $\epsilon \to 0$ indicates total fairness while $\epsilon \to 1$ indicate focus on performance.

In a nutshell, the scheduler should begin by using the guidelines we have already described. Then, if a job receives less than the its fair share, i.e., fewer than $S/N(t) - \epsilon$ slots, the job's capacity assignment is bumped up to $S/N(t) - \epsilon$. Next, the remaining slots are allocated to the remaining jobs according to Guidelines 1 and 2. Note that this is a form of projection from the original (unfair) allocation into the feasible set of allocations defined by the fairness constraints. Algorithm 2 describes it in detail.

Algorithm 2 (Fairness).
Let $J(t) = \{J_1, J_2, \ldots, J_n\}$ denote the jobs in the system at time t sorted in increasing order of remaining tasks, so $T_1(t) \leq \ldots \leq T_n(t)$. Define m_1 such that $i \leq m_1$ implies $\frac{2}{\beta}T_i(t) \leq \frac{S}{N} - \epsilon$.

1. *If* $S \leq \frac{2}{\beta} \sum\limits_{i=m_1+1}^{n} T_i(t) + m_1\left(\frac{S}{N} - \epsilon\right)$*, begin by assigning all jobs* $\frac{S}{N} - \epsilon$ *slots. Then assign an additional* $\frac{2}{\beta}T_i(t) - (\frac{S}{N} - \epsilon)$ *slots to jobs* J_i *from* $i = m_1 + 1$ *to* n *until no slots remain.*
2. *If* $S > \frac{2}{\beta} \sum\limits_{i=m_1+1}^{n} T_i(t) + m_1\left(\frac{S}{N} - \epsilon\right)$*, then define* m_2 *as the minimum value such that*

$$\frac{T_{m_2+1}(t)}{\sum\limits_{i=m_2+1}^{N} T_i(t)}(S - m_2(\frac{S}{N} - \epsilon)) \geq \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta}T_{m_2+1}(t)\}.$$

Then, assign $\frac{S}{N} - \epsilon$ *slots to jobs* J_i *with* $1 \leq i \leq m_2$*, and assign* $\frac{T_i(t)}{\sum\limits_{i=m_2+1}^{N} T_i(t)}(S - m_2(\frac{S}{N} - \epsilon))$ *slots to jobs* J_i *with* $m_2 + 1 \leq i \leq N$*.*

Theorem 2. *Algorithm 2 is throughput maximal among* ϵ*-fair allocations.*

Proof. Let $J_m(t) = \{J_1, J_2, \ldots, J_{m_1}\}$. Similarly, we divide the problem into two cases.

Case 1: $S \leq \frac{2}{\beta} \sum\limits_{i=m_1+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m_1$

Without the fairness constraint, when $S \leq \frac{2}{\beta}T_i(t)$, to maximize the throughput, the scheduler should assign exactly $\frac{2}{\beta}T_i(t)$ to jobs J_i for i from 1 to n until no remaining slot. With fairness constraint, for any job $J_i \in J_m(t)$, it will surely get $\frac{S}{N} - \epsilon \geq \frac{2}{\beta}T_i(t)$ slots. So when slots are not enough to share across jobs in $J(t) - J_m(t)$ to guarantee optimal speculation level for every job, jobs in $J_m(t)$ should not get more slots than $\frac{S}{N} - \epsilon$.

Thus, when $S \leq \frac{2}{\beta} \sum\limits_{i=m_1+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m_1$, optimal scheduling should assign every job $\frac{S}{N} - \epsilon$ slots at first step. Then, assign $\frac{2}{\beta}T_i(t) - (\frac{S}{N} - \epsilon)$ slots to job $J_i \in J(t)$ from $i = m + 1$ to N until slots remain.

Case 2: $S > \frac{2}{\beta} \sum\limits_{i=m_1+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m_1$

When $S \geq \frac{2}{\beta} \sum\limits_{i=m+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m$, all jobs should get at least $\max\{\frac{S}{N} - \epsilon, \frac{2}{\beta}T_i(t)\}$ slots. The first constraint is from fairness and the second constraint is from the optimality of $\frac{2}{\beta}T_i(t)$. Then, the throughput maximization problem is equivalent to the following optimization problem,

$$\begin{aligned}
&\text{maximize} && \sum_{i=1}^{N}\left(\frac{\beta}{\beta-1}T_i(t)-\frac{1}{\beta-1}\frac{T_i(t)^2}{S_i}\right)\\
&\text{subject to} && \sum_{i=1}^{N}S_i=S\\
& && S_i\geq\frac{2}{\beta}T_i(t)\\
& && S_i\geq\frac{S}{N}-\epsilon
\end{aligned}$$

Note that from the definition of m_1, $\frac{S}{N}-\epsilon\geq\frac{2}{\beta}T_i(t)$ for all $1\leq i\leq m_1$, and $\frac{S}{N}-\epsilon\leq\frac{2}{\beta}T_i(t)$ for all $m_1+1\leq i\leq N$. Thus, the optimization problem can be simplified as,

$$\begin{aligned}
&\text{minimize} && \sum_{i=1}^{N}\frac{T_i(t)^2}{S_i} && (3.8)\\
&\text{subject to} && \sum_{i=1}^{N}S_i=S\\
& && S_i\geq\frac{2}{\beta}T_i(t), i=m+1,\ldots,N\\
& && S_i\geq\frac{S}{N}-\epsilon, i=1,\ldots,m
\end{aligned}$$

The above is a convex optimization problem. The Lagrange dual function $L(S_1,\ldots,S_N,\lambda,v)$ is,

$$\sum_{i=1}^{N}\frac{T_i(t)^2}{S_i}+v(\sum_{i=1}^{N}S_i-S)+\sum_{i=1}^{m}\lambda_i(\frac{S}{N}-\epsilon-S_i)+\sum_{i=m+1}^{N}\lambda_i(\frac{2}{\beta}T_i(t)-S_i). \qquad (3.9)$$

From KKT condition, in optimal solution,

$$-\frac{T_i(t)^2}{S_i^2} + v - \lambda_i = 0, \tag{3.10}$$

which implies $S_i = \frac{T_i(t)}{\sqrt{v-\lambda_i}}$, and

$$\lambda_i \neq 0 \Leftrightarrow S_i = \max\{\frac{2}{\beta}T_i(t), \frac{S}{N} - \epsilon\}. \tag{3.11}$$

Substitute (3.11) into (3.10), we get if $S_i \neq \max\{\frac{2}{\beta}T_i(t), \frac{S}{N} - \epsilon\}$, then $S_i = \frac{T_i(t)}{\sqrt{v}}$, which indicates that for jobs J_i with $S_i \neq \max\{\frac{2}{\beta}T_i(t), \frac{S}{N} - \epsilon\}$, the slots assigned to J_i is on proportional to $T_i(t)$. Precisely, The slot assignment for each job falls into the following three cases,

1. $S_i = \frac{S}{N} - \epsilon$
2. $S_i = \frac{2}{\beta}T_i(t)$
3. $S_i \neq \frac{S}{N} - \epsilon$ and $S_i \neq \frac{2}{\beta}T_i(t)$

Let $J_i(t)$ denote the jobs falling in case i, for $i = 1, 2, 3$. Then, specifically, for job J_i in $J_3(t)$, $S_i = \frac{T_i(t)}{\sum\limits_{J_3(t)} T_i(t)} S_r$, where S_r is the remaining number of slots after assignment of $J_1(t)$ and $J_2(t)$.

The only remaining question is, given a job J_i, in optimal scheduling, which set, $J_1(t)$, $J_2(t)$ or $J_3(t)$, it belongs to. From the following three claims, we prove $J_1(t) \subset J_{m_1}(t), J_2(t) = \emptyset$, and $J_3(t) = J(t) - J_1(t)$.

1. Claim: In optimal solution, $S_1 \leq S_2 \leq \ldots \leq S_n$.

   Proof. For any $i < j$, if $S_i \geq S_j$, we can always let $S_i' = S_j$ and $S_j' = S_i$ and obtain an smaller result in (3.8). □

2. Claim: there exists a number m_2, $1 \leq m_2 \leq m_1$ such that in optimal scheduling, $J_1(t) = \{J_1, J_2, \ldots, J_{m_2}\}$

   Proof. From objective function $\sum_{i=1}^{N} \frac{T_i(t)^2}{S_i}$, and claim in (a),

   If $S_i = \frac{S}{N} - \epsilon$, then$\forall j \leq i, S_j = \frac{S}{N} - \epsilon$, and if $S_i \neq \frac{S}{N} - \epsilon$, which implies $S_i > \frac{S}{N} - \epsilon$, then $\forall j \geq i, S_j > \frac{S}{N} - \epsilon$.

   Suppose the last job in $J(t)$ with $\frac{S}{N} - \epsilon$ slots is J_{m_2}. Obviously, $1 \leq m_2 \leq m_1$. Then, $J_1(t) = \{J_1, J_2, \ldots, J_{m_2}\}$. □

3. Claim : $J_2(t) = \emptyset$

   Proof. Since $S > \frac{2}{\beta} \sum_{i=m+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m_1$, $J_3(t) \neq \emptyset$.

   Denote the total slots assigned to $J_2(t)$ and $J_3(t)$ by S_2 and S_3, respectively. It is easy to verify that $\frac{T_i(t)}{\sum_{J_2(t)+J_3(t)} T_i(t)}(S_2 + S_3) \geq \frac{2}{\beta} T_i(t)$, and $\frac{T_i(t)}{\sum_{J_2(t)+J_3(t)} T_i(t)}(S_2 + S_3) \geq \frac{S}{N} - \epsilon$ (second equality holds since $\frac{\beta}{2} T_i(t) \geq \frac{S}{N} - \epsilon, \forall J_i \in J_2(t) + J_3(t)$). Thus, if $J_2(t) \neq \emptyset$, we can always combine $J_2(t)$ and $J_3(t)$, and do load balancing in the new set. From Theorem 1, the latter method obtains a better throughput. □

From the above three claims, in optimal scheduling, $J_1(t) = \{J_1, \ldots, J_{m_2}\}$, and $J_3(t) = J(t) - J_1(t)$. The only question to ask is, what m_2 is in optimal scheduling. We find m_2 by studying the optimal total throughput.

The total throughput is,

$$
\begin{aligned}
&\sum_{i=1}^{N} \left(\frac{\beta}{\beta - 1} T_i(t) - \frac{1}{\beta - 1} \frac{T_i(t)^2}{S_i} \right) \\
=&\sum_{i=1}^{N} \frac{\beta}{\beta - 1} T_i(t) - \frac{1}{\beta - 1} \sum_{i=1}^{m_2} \frac{T_i(t)^2}{\frac{S}{N} - \epsilon} - \frac{1}{\beta - 1} \sum_{i=m_2+1}^{N} \frac{T_i(t)^2}{S_i} \\
=&\sum_{i=1}^{N} \frac{\beta}{\beta - 1} T_i(t) - \frac{1}{\beta - 1} \sum_{i=1}^{m_2} \frac{T_i(t)^2}{\frac{S}{N} - \epsilon} - \frac{1}{\beta - 1} \left(\sum_{i=m_2+1}^{N} T_i(t) \right)^2 \frac{1}{S - m_2(\frac{S}{N} - \epsilon)}
\end{aligned}
$$

It is easy to verify, as m_2 increases, the total throughput decreases. Thus, the optimal scheduling should find the minimal m_2 while satisfies the following conditions,

1. $1 \leq m_2 \leq m_1$

2. $\frac{T_i(t)}{\sum_{i=n_1+1}^{N} T_i(t)}(S - n_1(\frac{S}{N} - \epsilon)) \geq \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta}T_i(t)\}$, for all $i \geq n_1 + 1$

Note, since $T_1(t) \leq T_2(t) \leq \ldots \leq T_N(t)$, condition 2 can be simplified as,

$$\frac{T_{m_2+1}}{\sum\limits_{i=m_2+1}^{N} T_i(t)}(S - m_2(\frac{S}{N} - \epsilon)) \geq \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta}T_{m_2+1}\}.$$

And m_2 always exists, since m_1 itself satisfies the above two conditions.

In summary, the optimal scheduling should:

1. If $S \leq \frac{2}{\beta} \sum_{i=m_1+1}^{n} T_i(t) + m_1\left(\frac{S}{N} - \epsilon\right)$, begin by assigning all jobs $\frac{S}{N} - \epsilon$ slots. Then assign an additional $\frac{2}{\beta}T_i(t) - (\frac{S}{N} - \epsilon)$ slots to jobs J_i from $i = m_1 + 1$ to n until no slots remain.
2. If $S > \frac{2}{\beta} \sum_{i=m_1+1}^{n} T_i(t) + m_1\left(\frac{S}{N} - \epsilon\right)$, then define m_2 as the minimum value such that

$$\frac{T_{m_2+1}(t)}{\sum\limits_{i=m_2+1}^{N} T_i(t)}(S - m_2(\frac{S}{N} - \epsilon)) \geq \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta}T_{m_2+1}(t)\}.$$

Then, assign $\frac{S}{N} - \epsilon$ slots to jobs J_i with $1 \leq i \leq m_2$, and assign $\frac{T_i(t)}{\sum_{i=m_2+1}^{N} T_i(t)}(S - m_2(\frac{S}{N} - \epsilon))$ slots to jobs J_i with $m_2 + 1 \leq i \leq N$.

$\square$

The main message from the analysis is that ϵ-fairness can be maintained without major changes to the structure of the algorithm.

Our experimental results (§6.3) highlight that ϵ-fairness achieves significant gains with little downside. In fact, even at moderate values of ϵ, nearly all jobs finish faster then they would have under fair scheduling.

This fact, though initially surprising, is actually similar to the conclusions that have been derived about fairness of policies that prioritize small jobs in other contexts. For example, in single server scheduling it has been shown that SRPT scheduling, which is seemingly unfair to large job sizes, actually can improve the response time of every job size (when job sizes are heavy-tailed) compared to fair scheduling policies [17, 18, 19].

Chapter 4

Heterogeneous Job DAGs

The design guidelines we have discussed so far are based on homogeneous single-phased jobs. In this section, we extend them to handle more complex real-world DAGs of jobs (§4.1) with heterogeneous distributions (β) of task durations (§4.2). Our generic model ensures that the guidelines require only minor adjustments.

4.1 DAG of Tasks

We now extend our analysis of single-phased jobs to multi-phased DAGs of tasks that have varied communication patterns (e.g., many-to-one or all-to-all). We consider multiple phases that are not separated by strict barriers but are rather *pipelined.* Downstream tasks do not wait for *all* the upstream tasks to finish but read the upstream outputs as the tasks finish.

Reading the outputs relies on the network and pipelining the reads is beneficial because the upstream tasks are typically bottlenecked on other *non-overlapping* resources (CPU, memory). While pipelining improves utilization of the different resources in the cluster, it adds a challenging dimension to the scheduling problem.

The scheduler's goal is to balance the gains due to overlapping network utilization while still favoring upstream phases with smaller number of tasks. We capture this using a simple weighting factor, α per job, set to be the ratio of remaining work in network transfer in the downstream phase to the work in the upstream phase. We approximate the former using the amount of data remaining to be read and the latter with the number of remaining upstream tasks. We defer the exact details of estimating α to §5.2 but it suffices to understand that it is favors jobs with higher remaining communication and lower remaining tasks in the running phase.

Given the weighting factor α, our analytic results highlight that the structural form of Guidelines 1 and 2 do not change. However, the following adjustments are required.

First, in Guideline 1, the prioritization of jobs based on $T_i(t)$ should be replaced by a prioritization of jobs based on $\max\{T_i(t), T'_i(t)\}$, where $T_i(t)$ is the remaining number of tasks in the current phase and $T'_i(t)$ is the remaining work in communication in the downstream phase. This adjustment is motivated by the work of [5], which proves that, so-called, MaxSRPT is 2-speed optimal for completion times.[1] However, the model in [5] does not include stragglers, and so we need to supplement MaxSRPT using Guidelines 1 and 2 in order to incorporate speculation.

To accomplish this, the second change we make is to redefine the virtual size of a job to include α. In particular, we now define the virtual size of a job as

$$V_i(t) = \frac{2}{\beta} T_i(t) \sqrt{\alpha_i}.$$

This change to the virtual size impacts both Guideline 1 and 2. Importantly, it means that Guideline 2 suggests sharing capacity as follows: job i receives

$$\left(\frac{V_i(t)}{\sum V_j(t)} \right) S = \left(\frac{T_i(t)\sqrt{\alpha_i}}{\sum T_j(t)\sqrt{\alpha_j}} \right) S \text{ slots.} \tag{4.1}$$

We used a weighting factor α_i to understand how to adjust the optimal speculation level of jobs depending on the relative sizes of job i in the current phase and the following phase. For example, by setting $\alpha_i = T'_i/T_i$, it captures number of tasks created in the next

[1]2-speed optimal means that MaxSRPT guarantees response times better than the optimal in the original system, if it is given twice the service capacity. Note that [5] also shows that it is impossible to be constant-competitive without being granted extra service capacity.

phase per task completed in the current phase, which is appropriate when adjacent phases in the DAG can be pipelined.

Mathematically, one can show that if we seek to maximize the α-weighted throughput, i.e. $\sum_i \alpha_i \mu_i(t)$, then the optimal speculation level changes from $2/\beta T_i(T)$ to $2/\beta T_i(t)\sqrt{\alpha_i/\alpha_{min}}$, where α_{min} is the smallest α_j among the jobs that are currently running. This leads to the following algorithm for the case of DAGs of tasks.

Algorithm 3 (DAGs of tasks)**.**
Let $J(t) = \{J_1, J_2, \ldots, J_n\}$ denote the jobs in the system at time t sorted in ascending order of $\max\{T_i(t), T_i'(t)\}$. If J_i and J_j have the same $\max\{T_i(t), T_i'(t)\}$ then the job with larger weight is listed first. Let $\alpha_{\min}^{(k)}$ denote the minimum weight of weights for first k jobs in $J(t)$, so $\alpha_{\min}^{(k)} = \min\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$. And let $J_{k_{\min}}$ denote the job has the minimum weight in first k jobs, so the weight of $J_{k_{\min}}$ is $\alpha_{\min}^{(k)}$. Let $V_i(t)$ denote the virtual size for job $J_i \in J(t)$, so $V_i(t) = \frac{2}{\beta}T_i(t)\sqrt{\alpha_i}$.

1. *If $S \le \frac{V_1(t)}{\sqrt{\alpha_{\min}^{(2)}}}$,assign $S_1 = S$ and $S_i = 0$ for $i > 1$.*
2. *If $\exists\ k < n$ such that $\sum_{i=1}^{k} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}} < S \le \sum_{i=1}^{k+1} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}}$,assign $S_i = \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}}$ for i in order of $\{1, 2, \ldots, k_{\min}-1, k_{\min}+1, \ldots, k, k+1, k_{\min}\}$ until no remain slots, and $S_i = 0$ for $i > k+1$.*
3. *If $\exists\ k < n-1$ such that $\sum_{i=1}^{k+1} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}} < S \le \sum_{i=1}^{k+1} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+2)}}}$, then assign $S_i = \frac{V_i(t)}{\sum_{i=1}^{k+1} V_i(t)}S$ for $i = 1, \ldots, k+1$, and $S_i = 0$ for $i > k+1$.*
4. *If $\sum_{i=1}^{n} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(n)}}} < S$, then assign $S_i = \frac{V_i(t)}{\sum_{i=1}^{n} V_i(t)}S$ for $i = 1, 2, \ldots, n$.*

Algorithm 3 presents the details of the allocation and we evaluate the gains from this generalization in §6.4. Interestingly, the optimality of a square-root weighting factor has been observed in other heterogeneous cluster scheduling problems as well, e.g., load balancing across servers with heterogeneous speeds [20].

4.2 Heterogeneous Stragglers

In the previous sections, we have assumed that all jobs have the same task size distributions, i.e., have the same straggler behavior. This may not always be the case and, more generally, different classes of jobs may have different straggler behaviors. This could be specific to the jobs' computation and input locations, or wider (but time-varying) cluster characteristics like resource contentions due to utilization and hotspots [21].

Heterogeneous straggler behaviors can have a significant impact on scheduling. In particular, if one class of jobs is likely to have stragglers more frequently, then speculation will be more valuable within those jobs. Thus, the job scheduler may want to leave more capacity for such jobs, but this extra capacity comes at the expense of other jobs, and so it is not clear how much extra capacity should be allocated.

Our analytic results highlight that the structural forms of Guidelines 1 and 2 do not change in this setting; however, the virtual sizes of the jobs are adjusted depending on the job-class β_i and the specific form of the proportional sharing should be adjusted as follows. Specifically, suppose that there are two classes that have different Pareto(β_i) task size distributions. Then, our analytic results suggest that class I should be allocated

$$\left(\frac{\frac{\sum_I T_i(t)}{\sqrt{\beta_I - 1}}}{\frac{\sum_I T_i(t)}{\sqrt{\beta_I - 1}} + \frac{\sum_{II} T_i(t)}{\sqrt{\beta_{II} - 1}}}\right) S \text{ slots,}$$

and the allocation among jobs within the class should then happen according to the the proportional sharing in Equation 4.1. Algorithm 4 gives the details of the design and we evaluate the gains from this generalization in §6.4. Interestingly, the form mimics the proportional sharing in Guideline 2, and the weighting of β is again by its square root. Note that the importance of $(\beta - 1)$ is natural since when $\beta < 2$ the mean is infinite.

Algorithm 4 (Heterogenous stragglers)**.**

Let $I(t)$ and $II(t)$ denote the set of type 1 and type 2 jobs present at time t, respectively.

1. *If* $S < \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t)$*, then assign all* S *slots to type 1 jobs.*

2. *If* $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) < S \le \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t)+\frac{2}{\beta_2}\sum_{II(t)} T_i(t)$*, then assign* $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t)$ *slots to type 1 jobs and the rest to type 2 jobs.*

3. *If* $S \ge \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t)+\frac{2}{\beta_2}\sum_{II(t)} T_i(t)$*, then assign* $\frac{\frac{\sum_{I(t)} T_i(t)}{\sqrt{\beta_1-1}}}{\frac{\sum_{I(t)} T_i(t)}{\sqrt{\beta_1-1}}+\frac{\sum_{II(t)} T_i(t)}{\sqrt{\beta_2-1}}}S$ *slots to type 1 jobs and the rest to type 2 jobs.*

Given this allocation of capacity to type 1 and 2 jobs, use Algorithm **??** *to assign capacity within each type.*

Theorem 3. *Algorithm 4 is throughput maximal.*

Proof. Note, if we know the optimal scheduling algorithm assigns S_1 slots to type 1 jobs and S_2 slots to type 2 jobs, then we know how to allocate slots across jobs within the same type as indicated in Algorithm **??**. The remaining question is how to allocate slots across different types.

Similar to proof for Theorem 1, we divide the problem into three cases.

Case 1: $S \le \frac{2}{\beta_1}\sum_{I(t)} T_i(t)$

Note that $\beta_1 < \beta_2$ gives $\frac{\beta_1^2}{4(\beta_1-1)} > \frac{\beta_2^2}{4(\beta_2-1)}$, as $f(x) = \frac{x^2}{x-1}$ is a decreasing function for $x \in (1,2)$. When $S \le \frac{2}{\beta_1}\sum_{I(t)} T_i(t)$, from (3.7), we know, we should assign all slots to type 1 jobs.

Case 2: $\frac{2}{\beta_1}\sum_{I(t)} T_i(t) \le S \le \frac{2}{\beta_1}\sum_{I(t)} T_i(t) + \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$

When $\frac{2}{\beta_1}\sum_{I(t)} T_i(t) \le S \le \frac{2}{\beta_1}\sum_{I(t)} T_i(t)+\frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, denote the number of slots we assign to type 1 jobs by S_1 and the number of slots we assign to type 2 jobs by S_2. From (3.7), unless type 1 jobs get optimal speculation level slots, no slot should be assigned to type 2 jobs. Thus, the slots assigned to type 1 and type 2 jobs should satisfy $S_1 \ge \frac{2}{\beta_1}\sum_{I(t)} T_i(t)$

and $S_2 \leq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$. The total throughput is,

$$\begin{aligned}
&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) - \frac{1}{(\beta_1 - 1)S_1}(\sum_{I(t)} T_i(t))^2 + \frac{\beta_2^2}{4(\beta_2 - 1)}S_2 \\
=&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) - \frac{1}{(\beta_1 - 1)S_1}(\sum_{I(t)} T_i(t))^2 + \frac{\beta_2^2}{4(\beta_2 - 1)}(S - S_1) \\
=&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2 - 1)}S - \frac{1}{(\beta_1 - 1)}(\sum_{I(t)} T_i(t))^2 \frac{1}{S_1} - \frac{\beta_2^2}{4(\beta_2 - 1)}S_1 \qquad (4.2) \\
\leq&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2 - 1)}S - \frac{\beta_2}{\sqrt{(\beta_1 - 1)(\beta_2 - 1)}}\sum_{I(t)} T_i(t),
\end{aligned}$$

where the last line follows from $a^2 + b^2 \geq 2ab$.

Equality is achieved when $\frac{\beta_2^2}{4(\beta_2-1)}S_1 = \frac{1}{(\beta_1-1)S_1}(\sum\limits_{I(t)} T_i(t))^2$, i.e., $S_1 = \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t)$. And (4.2) increases for $S_1 \leq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t)$ and decreases afterwards. Also note that, if $S_1 = \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t)$, then $S_1 \geq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t)$. Thus, when $\frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t) \leq S \leq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, the optimal scheduling should:

1. when $S < \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t)$, assign all S slots to type 1 jobs.

2. when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t) < S \leq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, assign $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t)$ slots to type 1 jobs and the rest to type 2 jobs.

Case 3: $S \geq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$

When $S \geq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, similarly, from (3.7), in optimal scheduling, the number of slots assigned to type 1 jobs should be no less than the optimal speculation scheduling level, so $S_1 \geq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t)$. Depending on how many slots we have, S_2 can be either less than or more than optimal speculation level. We discuss the two cases separately in the following.

1. If $S_2 \leq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, which implies $S - S_1 \leq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, as we already proved, $S_1 = \max\{\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t), S - \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)\}$, and $S_2 = S - S_1$. Specifically,
   (a) when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t) \geq S - \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, if $S_1 \geq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t)$ and $S_2 \leq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$ in optimal scheduling, then $S_1 = \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t)$ and $S_2 = S - S_1$.
   (b) when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t) < S - \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, if $S_1 \geq \frac{2}{\beta_1} \sum\limits_{I(t)} T_i(t)$ and $S_2 \leq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$ in optimal scheduling , then $S_1 = S - \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$ and $S_2 = S - S_1$.

2. If $S_2 \geq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, which implies $S - S_1 \geq \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, total throughput is,

$$\begin{aligned}
&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) - \frac{1}{(\beta_1-1)S_1}(\sum_{I(t)} T_i(t))^2 + \frac{\beta_2}{\beta_2 - 1}\sum_{II(t)} T_i(t) - \frac{1}{(\beta_2-1)S_2}(\sum_{II(t)} T_i(t))^2 \\
=&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2 - 1}\sum_{II(t)} T_i(t) - \frac{1}{(\beta_1-1)S_1}(\sum_{I(t)} T_i(t))^2 - \frac{1}{(\beta_2-1)S_2}(\sum_{II(t)} T_i(t))^2 \\
\leq&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2 - 1}\sum_{II(t)} T_i(t) - \frac{1}{S_1+S_2}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2 \\
=&\frac{\beta_1}{\beta_1 - 1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2 - 1}\sum_{II(t)} T_i(t) - \frac{1}{S}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2,
\end{aligned} \tag{4.3}$$

where the inequality follows from Cauchy-Schwartz inequality.

Equality is achieved when $S_1 = \frac{\frac{\sum_{I(t)} T_i(t)}{\sqrt{\beta_1-1}}}{\frac{\sum_{I(t)} T_i(t)}{\sqrt{\beta_1-1}} + \frac{\sum_{II(t)} T_i(t)}{\sqrt{\beta_2-1}}} S$

Combining above two results, when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum\limits_{I(t)} T_i(t) < S - \frac{2}{\beta_2} \sum\limits_{II(t)} T_i(t)$, the optimal assignment from first case is a boundary point for second case. Obviously, the optimal assignment from case 2 is the global optimal assignment. But, when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t) \geq S - \frac{2}{\beta_2}\sum\limits_{II(t)} T_i(t)$, it still remain unclear which assignment is optimal. Thus, in next step, we compare the maximum total throughput in two cases under that setting.

(a) In case 1, from (4.2), throughput $\mu_s = \frac{\beta_1}{\beta_1-1}\sum\limits_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2-1)}S - \frac{\beta_2}{\sqrt{(\beta_1-1)(\beta_2-1)}}\sum\limits_{I(t)} T_i(t)$

(b) In case 2, from (4.3), throughput $\mu_l = \frac{\beta_1}{\beta_1-1}\sum\limits_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2-1}\sum\limits_{II(t)} T_i(t) - \frac{1}{S}\left(\sqrt{\frac{1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\right.$

$$
\begin{aligned}
\mu_s - \mu_l =& \frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2-1)}S - \frac{\beta_2}{\sqrt{(\beta_1-1)(\beta_2-1)}}\sum_{I(t)} T_i(t) \\
&- \left(\frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) - \frac{1}{S}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2\right) \\
=& \frac{\beta_2^2}{4(\beta_2-1)}S + \frac{1}{S}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2 \\
&- \frac{\beta_2}{\sqrt{(\beta_1-1)(\beta_2-1)}}\sum_{I(t)} T_i(t) - \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) \\
\geq& \frac{\beta_2}{\sqrt{\beta_2-1}}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right) \\
&- \frac{\beta_2}{\sqrt{(\beta_1-1)(\beta_2-1)}}\sum_{I(t)} T_i(t) - \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) \\
=& 0
\end{aligned}
$$

The above result implies when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t) \geq S - \frac{2}{\beta_2}\sum\limits_{II(t)} T_i(t)$, the optimal assignment from case 1 is the global optimal assignment.

In summary, the optimal scheduling should:

1. If $S < \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t)$, then assign all S slots to type 1 jobs.

2. If $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t) < S \leq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t) + \frac{2}{\beta_2}\sum\limits_{II(t)} T_i(t)$, then assign $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t)$ slots to type 1 jobs and the rest to type 2 jobs.

3. If $S \geq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum\limits_{I(t)} T_i(t) + \frac{2}{\beta_2}\sum\limits_{II(t)} T_i(t)$, then assign $\frac{\frac{\sum_{I(t)} T_i(t)}{\sqrt{\beta_1-1}}}{\frac{\sum_{I(t)} T_i(t)}{\sqrt{\beta_1-1}} + \frac{\sum_{II(t)} T_i(t)}{\sqrt{\beta_2-1}}} S$ slots to type 1 jobs and the rest to type 2 jobs.

$\square$

Chapter 5

Hopper: Design and Implementation

In this section, we build our system, Hopper, based on the theoretical guidelines. Hopper is implemented inside Hadoop [15] and Spark [3] compute frameworks.

5.1 Data Locality

Implicit in our model's goal of allocating slots to jobs is the assumption that all the slots are equivalent. In practice, however, tasks have preferences towards machines with specific characteristics [22]. The dominant instance of such preferences is *data locality*, i.e., execute tasks on the same machine as their input. With the trend towards in-memory storage [3, 23], reading data from local memory is appreciably faster than remote reads over the network. As these tasks are predominantly IO-intensive, data locality is crucial.

As per our guidelines, however, tasks of the next best job to schedule may not have memory local slots available [21]. Our analysis shows that memory locality drops from 98% of tasks with currently deployed techniques to as low as 54% if scheduled purely based on our guidelines without regard to memory locality. Not only are the tasks not achieving memory locality slowed down, the ensuing increase in network traffic also slows down the data transfers of intermediate phases.

We devise a simple relaxation approach to balance adherence to our guidelines and locality. In the ordering of jobs, instead of allotting slots to the job with the smallest virtual size, we allow for picking any of the smallest k% of jobs whose tasks can run with memory locality on the available slots. Among these smallest $k\%$ jobs, we pick the one which can achieve memory locality for the maximum number of tasks. Further, once we pick a job, we schedule all its tasks (so that a few unscheduled tasks do not delay it) before resuming to the scheduling order as per our guidelines. In practice, a small value of k suffices ($\leq 5\%$) due to high churn in task completions and slot availabilities (evaluated in §6.4.1).

5.2 Estimating Intermediate Data Sizes

Recall from §4.1 that our scheduling guidelines recommend scaling every job's allocation by $\sqrt{\alpha}$ in the case of DAGs. The factor α is the ratio of the amounts of work remaining in the downstream phase over the amounts of works remaining in the upstream phase of the job's DAG. The purpose of the scaling is to ensure pipelining of the reading of upstream tasks' outputs over the network.

The key to calculating α is estimating the size of the *intermediate* output produced by tasks. Unlike the job's input size, intermediate data sizes are not known upfront. We predict intermediate data sizes based on similar jobs in the past. Clusters typically have many recurring jobs that execute periodically as newer data streams in, and produce intermediate data of similar sizes.

For multi-waved jobs [23, 24], Hopper can do better. It uses the ratio of intermediate to input data of the completed tasks as a predictor the future (incomplete) tasks. Data from Facebook's and Microsoft Bing's clusters (described in §6.1) shows that while the ratio of input to output data size of tasks vary from 0.05 all the way to 18, the ratios *within* tasks of a phase have a coefficient-of-variation of only 0.07 and 0.24 at median and 90th percentile, thus lending themselves to effective learning. Hopper calculates α as the ratio of the data remaining to be read (by downstream tasks) over the data remaining to be produced (by upstream tasks).

Hopper's approach for pipelining phases easily composes to DAGs of arbitrary depth since it deals with only two phases at time, i.e., the currently running phase and the downstream phase reading the output from the running phase. Further, the usage of α is generically applicable to all communication patterns of intermediate data (e.g., many-to-one, all-to-all) as it only considers the amount of data outputted and read (shown in §6.2.1)

5.3 System Implementation

We implement Hopper inside two frameworks: Hadoop YARN (version 2.3) and Spark (version 0.7.3). Hadoop jobs read data from HDFS [25] while Spark jobs read from in-memory RDDs. Consequently, Spark tasks finish faster than Hadoop tasks for the same input size.

Briefly, these frameworks implement two level scheduling where a central *resource manager* assigns slots to the different *job managers*. When a job is submitted to the resource manager, a job manager is started on one of the machines, that then executes the job's DAG of tasks. The job manager negotiates with the resource manager for resources for its tasks.

We built Hopper as a scheduling plug-in module to the resource manager. This makes the frameworks use our design to allocate slots to the job managers. We also piggybacked on the communication protocol between the job manager and resource manager to communicate the intermediate data produced and read by the phases of the job to vary α accordingly; locality and other preferences are already communicated between them.

Chapter 6

Evaluation

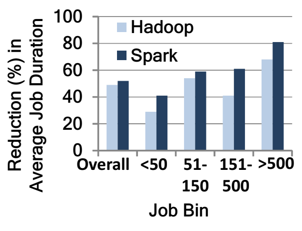
We evaluate our prototype of Hopper on a 200 node cluster using production workloads from Facebook and Microsoft Bing. We present Hopper's overall gains in §6.2, fairness results in §6.3, and design implications in §6.4.

1. Hopper improves the average job duration by 50% compared to SRPT scheduling and 70% compared to fair schedulers.
2. Hopper's balancing of fairness and performance ensures that only 4% of jobs slow down and by $\leq 5\%$.

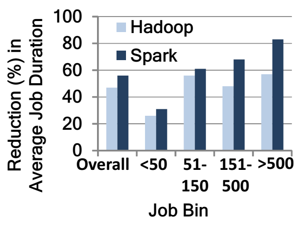
6.1 Setup

Workload: Our evaluation is based on traces from Facebook's production Hadoop [15] cluster ($3,500$ machines) and Microsoft Bing's Dryad cluster ($\mathcal{O}(1000)$ machines) from Oct-Dec 2012. The traces capture over a million jobs (experimental & production). The tasks had diverse resource demands of CPU, memory and IO, varying by a factor of $24\times$. To create our workload, we retain the inter-arrival times of jobs, their input sizes and number of tasks, resource demands as well as job scripts.

Cluster Deployment: We deploy our Hadoop and Spark prototypes on a 200-node private cluster and evaluate them using the workload described above. Each machine had 16 cores, 34GB of memory, 1Gbps network and 4 disks. The machines were connected using a



(a) Facebook



(b) Bing

Figure 6.1: Hopper's **gains (Facebook and Bing workloads).**

network with no over-subscription. Each experiment is repeated five times and we report the median.

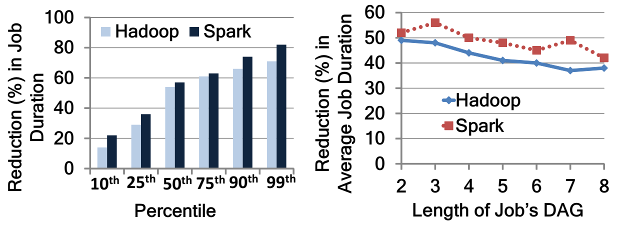
Baseline: We contrast Hopper with state-of-the-art scheduling algorithms and straggler mitigation schemes. We use scheduling baselines of SRPT and fair scheduling. Fair scheduling is commonly used in clusters today, e.g., [8, 9], but is inefficient with respect to completion time. In contrast, SRPT provides quite a competitive baseline for completion time (at the expense of unfairness). We combine each of them with the LATE [13], Mantri [12] and GRASS [14] speculation algorithms.

6.2 Hopper's Improvements

We first compare Hopper with SRPT. Unless specified, both Hopper and the baseline of SRPT executes the GRASS speculation algorithm per job, i.e. GRASS+Hopper vs. GRASS+SRPT; recent results have shown GRASS beats its competitors [14]. However, we also evaluate Hopper's compatibility with other speculation algorithms (LATE, Mantri) in §6.2.2. In our experiments, we set the fairness allowance ϵ to be 10% and locality parameter k as 3% unless otherwise stated.

Overall Gains: Figure 6.1 plots Hopper's gains in both Hadoop and Spark compared to SRPT. Jobs, overall, speedup by $\sim$ 50% in both prototypes (and workloads), which is significant given our aggressive baselines.

We observe two trends in the results. (a) First, gains for small jobs are less compared to the large jobs. This is only expected given that our baseline of SRPT already favors



(a) Distribution (b) DAG

Figure 6.2: **(a) Hopper's gains at various percentiles, and (b) gains as the length of the job's DAG varies.**

the small jobs. Nonetheless, Hopper's smart allocation of speculative slots offers $29\% - 40\%$ improvement. Gains for large jobs, in contrast, are over 80%. This not only shows that there is sufficient room for the large jobs despite favoring small jobs (due to the power law in distribution of job sizes [23, 11]) but also that the value of deciding between speculative tasks and unscheduled tasks of other jobs increases with the number of tasks in the job. With trends of smaller tasks and hence, larger number of tasks per job [24], Hopper's allocation will become important. (*b*) Second, gains for Spark are consistently higher (albeit, only modestly). Spark's small task durations makes it more sensitive to stragglers and thus it spawns many more speculative copies. This makes Hopper's scheduling more crucial.

Given the similarity in results (and for brevity), we only present the Facebook workload's results from now.

Distribution of Gains: Figure 6.2a plots the distribution of gains across jobs. While the median gains are just higher than the average, there is a $> 70\%$ gains at higher percentiles. The encouraging aspect is that gains even at the 10^{th} percentile are 14% and 22% in our Hadoop and Spark prototypes, respectively, which shows Hopper's ability to improve even the worse case performance.

6.2.1 DAG of Tasks

Hopper's gains hold steady for jobs with varying DAG lengths. We achieve different DAG lengths by modifying the input script of the job to contain the required number of phases, and enforce pipelining of data transfers of downstream phases with upstream tasks [26]. The communication patterns in the DAGs are varied (e.g., all-to-all, many-to-one etc.) and thus the results also serve to underscore Hopper's generality. As Figure 6.2b shows, name's gains continue to hold with the job's DAG length.

Recall from §4.1 that we use a factor α for pipelined downstream communication. To appropriately capture the network-intensiveness of the downstream phase, we use a dampening value for α.[1] For Spark jobs with fast in-memory map phases, intermediate data communication is the bottleneck, and a dampening value of 0.8 works best. Hadoop jobs spend most of their time in the map phase [23], and we use a dampening value of 0.3.

6.2.2 Speculation Algorithm

We now experimentally evaluate Hopper's performance with the different speculation mechanisms that are proposed and deployed. LATE [13] is deployed in Facebook's clusters, Mantri [12] is in operation in Microsoft Bing, and GRASS [14] is a recently reported straggler mitigation system that was demonstrated to perform near-optimal speculation. Our experiments in this section still use SRPT as the baseline but pair with the different straggler mitigation algorithms (e.g., LATE+SRPT vs. LATE+Hopper, and so forth). Figure 6.3 plots the results. We show results for Hadoop only due to space restrictions. The results for Spark are similar.

While the earlier results were achieved in conjunction with GRASS, a remarkable point is the similarity in gains even with LATE and Mantri. This indicates that as long as the straggler mitigation algorithms are aggressive in asking for speculative copies, Hopper will appropriately allocate as per the optimal speculation level. Overall, it emphasizes the aspect that resource allocation across jobs (with speculation) has a higher performance value than straggler mitigation within jobs.

[1]The dampening value is in $[0, 1]$. High values indicate a shuffle-intensive job, low values indicate a map-intensive job. We set the dampening value based on earlier similar jobs.

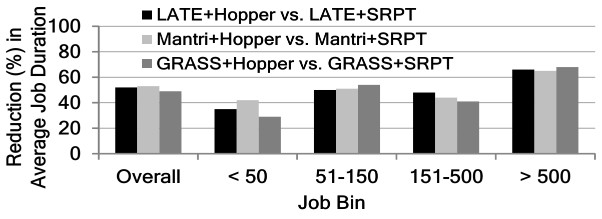
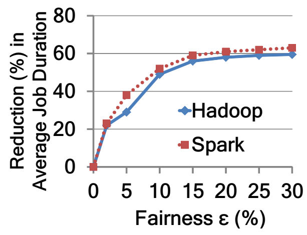
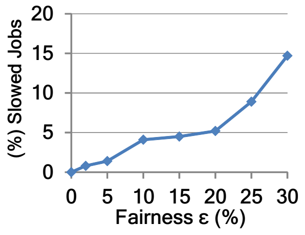


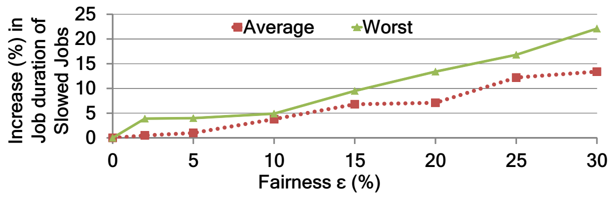
Figure 6.3: Hopper's **results are independent of the straggler mitigation strategy.**



(a) Sensitivity



(b) (%) of Jobs Slowed



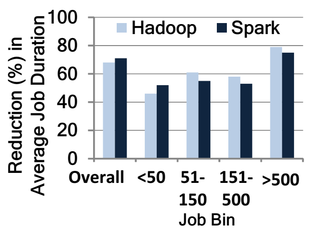
(c) Magnitude (%) of Slowdown

Figure 6.4: ϵ **Fairness. Figure (a) shows sensitivity of gains to ϵ. Figure (b) shows the fraction of jobs that slowed down compared to a fair allocation, and (c) shows the magnitude of their slowdowns (average and worst).**

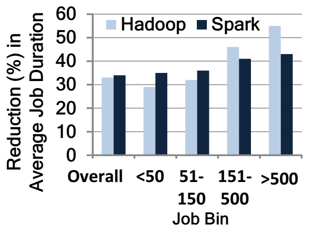
6.3 Fairness

As we had described in §3.5, the fairness knob of ϵ decides the leeway for Hopper to trade-off fairness for performance. Thus far, we had set ϵ to be 10% of the perfectly fair share of a job (ratio of total slots to jobs), now we analyze its sensitivity to Hopper's gains.

Figure 6.4a plots the increase in gains as we increase ϵ from 0 to 30%. The gains quickly rise for small values of ϵ and plateau beyond $\epsilon = 15\%$ with both our Hadoop as well as Spark prototypes; both curves follow each other. Hence, we conservatively set ϵ to be 10%. An important consideration in setting ϵ is the amount of *slowdown* of jobs compared to a



(a) Baseline: Fair Scheduler



(b) Baseline: DRF

Figure 6.5: Hopper's **gains with fair allocators as baseline.**

perfectly fair allocation ($\epsilon = 0$), i.e., all the jobs are guaranteed their fair share at all times.

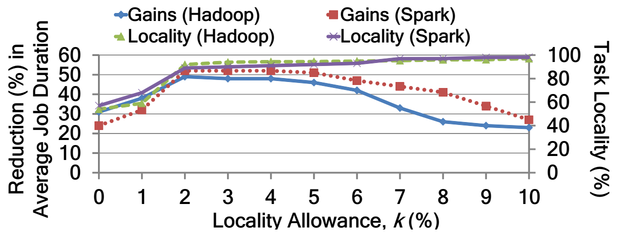
Figure 6.4b measures the number of jobs that slowed down, and for the slowed jobs, Figure 6.4c plots their average and worst case slowdowns. Note that fewer than 4% of jobs slow down with Hopper compared to a fair allocation at $\epsilon = 10\%$. The corresponding numbers for the Bing workload are 3.8% of jobs slowing down. In fact, both the average and worst case slowdowns are limited at $\epsilon = 10\%$, thus demonstrating that Hopper's focus on performance does *not* unduly slow down jobs.

Comparison to fairness algorithms: While we used SRPT scheduling as our baseline due to its focus on job completion, clusters commonly deploy fairness based job schedulers. We briefly look at Hopper's gains with the slot-based Fair Scheduler [8] and Dominant Resource Fairness (DRF) scheduler [9] as baselines.[2] Job speedups are over 70%, or in other words a $> 20\%$ raise from earlier (Figure 6.5). While gains for small jobs improve, gains for large jobs stay relatively unchanged. We believe that these gains compared to currently deployed fair schedulers coupled with very few jobs slowing down motivates clusters to safely deploy Hopper.

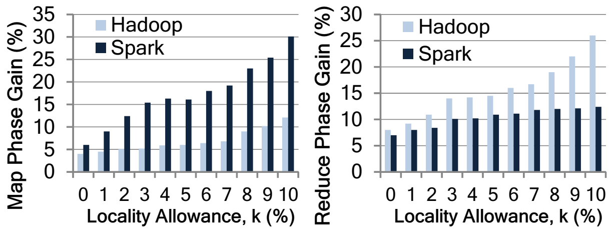
6.4 Evaluating Hopper's Design Decisions

We evaluate the design implications of Hopper: (*i*) data-local scheduling (§6.4.1), (*ii*) heterogeneous stragglers (§6.4.2), and (*iii*) scheduler scalability (§6.4.3).

[2]We implement multi-resource scheduling in Hopper by calculating a equal-weighted normalized value



(a) Sensitivity of gains and % local tasks to k.



(b) Map Phase (c) Reduce Reduce

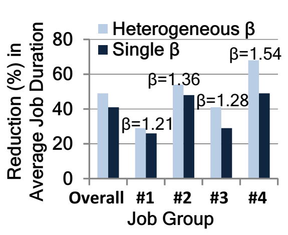
Figure 6.6: **Locality Allowance (k), see §5.1.**

6.4.1 Locality

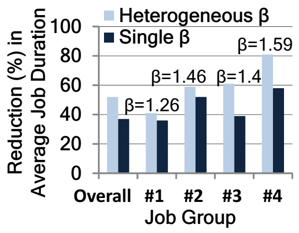
Achieving locality in scheduling (for map tasks) was an important aspect as we translated our scheduling guidelines into Hopper in §5.1. As Figure 6.6a shows, a small relaxation of $k = 3\%$ achieves appreciable increase in locality. Gains steady for a bit but then start dropping beyond a k value of 7%. This is because the deviation from the theoretical guidelines overshadows any increase in gains from locality. The fraction of data local tasks, naturally, increase with k (Figure 6.6a, right axis).

An interesting aspect is that not all the gains with k are attributed to only increase in locality. To see this, we slice the gains of individual phases—map phase, which is directly affected by locality, and reduce phase. Figure 6.6b, shows that the map phases in Spark speed up significantly as locality increases; Hadoop jobs' map phases do not. This is because data locality is more significant in Spark's in-memory system as opposed to Hadoop's disk-based storage; fast networks make *disk* locality less useful [27]. Hadoop jobs gain by improvement in their reduce phases due to lesser network contention during

across all the resources for the task demands and cluster capacity.



(a) Hadoop



(b) Spark

Figure 6.7: **Value of considering heterogeneity of task durations across job groups (shape parameter, β).**

transfers of intermediate data (Figure 6.6c).

6.4.2 Heterogeneous Stragglers

We now analyze the effect of using different straggler distributions for different job *groups* in our scheduling; we base our groups on simple higher-level organizational semantics. Within each group of jobs, we periodically model their distribution of task durations (β_i). Thus, every job arrives with a unique group i, and we allocate resources to groups as described in §4.2.

Figure 6.7 shows the benefits of considering heterogeneous task distributions as opposed to a homogeneous distribution (β=1.2) across all the tasks. We see a valuable difference of $\sim$ 10% in the overall completion time. The job groups with the most gains are those whose β_i values are much farther from the homogeneous β of 1.2. Note that since β is a function of task durations, they vary between our Hadoop and Spark deployments.

6.4.3 Scheduler Scalability

The guidelines in §3 rely on the total number of tasks in the system. Therefore, they result in a recalculation and change in allocation at the completion of *every* task. Since schedulers in current frameworks are optimized to handle tens of thousands of tasks per second [28], making expensive scheduling decisions are problematic. Measurements of the scheduler's throughput show no more than 1.6% drop: the throughput drops from 12,200 tasks/second with stock Hadoop and Spark schedulers to $12,010$ tasks/second with Hopper. We attribute this to Hopper's lightweight design decisions on all its aspects.

Chapter 7

Related Work

The problem of stragglers was first identified in the original MapReduce paper [1], and since then there have been many works that propose to mitigate the stragglers by utilizing speculative copies, e.g., [12, 13, 14]. These solutions, however, aim at mitigating the stragglers within each job, and lack coordination of resource allocation among all concurrently running jobs.

Job scheduling, on the other hand, is often done via algorithms that do not integrate straggler mitigation. Specifically, FIFO [4], the default scheduler for Hadoop and Spark, suffers from well known head-of-line blocking in multi-user cloud sharing settings. The inefficiency of FIFO inspired two different approaches: 1) introducing fairness among jobs; 2) prioritizing small jobs.

Based on the first approach, widely used solutions include the Fair Scheduler [8], Capacity Scheduler [29], DRF [9], FLEX [30], and Quincy [31]. While these schedulers guarantee fair sharing among jobs, fairness comes with its performance inefficiencies. Thus, advancements such as the Delay scheduler [32], and the Coupling Scheduler [33] have focused on mitigating some of these issues (e.g., locality) but still within a job.

On the second approach, the optimality of SRPT scheduling in both single and multi-server settings in queuing theory motivates a focus on prioritizing small jobs. Variations of SRPT that accommodate a variety of workload properties have been proposed. The dependency of two adjacent phases is highlighted in [5, 7]. Various queuing models (such as two-stage flexible flow-shop model [34], overlapping tandem queue model [5]) inspire new algorithmic designs.

Importantly, none of the systems provide job level scheduling algorithms considering speculative copies for stragglers. Further, as the examples in §2 illustrate, integrating straggler mitigation with job scheduling is challenging. We believe we are the first design to propose such a joint scheduler that focuses on performance while limiting the fallout due to unfairness using a simple knob.

Chapter 8

Conclusions

Guided by analytic scheduling guidelines, this thesis proposes Hopper, a job scheduler that is aware of speculative copies for stragglers. We believe Hopper is the first scheduler that combines the hitherto decoupled solutions for job scheduling and straggler mitigation. Hopper's design also incorporates practically important features like pipelining of intermediate data and data locality for tasks. Our evaluation using Hadoop and Spark prototypes show that Hopper speeds up average job completion time by $50\% - 70\%$ compared to SRPT scheduling, fair scheduling and straggler mitigation solutions. Further, Hopper's simple knob to trade fairness and performance results in fewer than 4% of jobs slowing down by $\leq 5\%$ compared to fairness-based schedulers.

Bibliography

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 2008.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *ACM Eurosys*, 2007.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *USENIX NSDI*, 2012.

[4] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, pages 15–1, 2004.

[5] M. Lin, L. Zhang, A. Wierman, and J. Tan. Joint Optimization of Overlapping Phases in MapReduce. *Performance Evaluation*, 2013.

[6] W. Wang, K. Zhu, L. Ying, J. Tan, L. Zhang . A Throughput Optimal Algorithm for Map Task Scheduling in Mapreduce with Data Locality. In *ACM SIGMETRICS*, 2013.

[7] Y. Wang, J. Tan, W. Yu, L. Zhang, X. Meng. Preemptive ReduceTask Scheduling for Fast and Fair Job Completion. *USENIX ICAC*, 2013.

[8] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica.

Job scheduling for multi-user mapreduce clusters. In *UC Berkeley Technical Report UCB/EECS-2009-55*, 2009.

[9] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *USENIX NSDI*, 2011.

[10] J. Dean. Achieving Rapid Response Times in Large Online Services. In *Berkeley AMPLab Cloud Seminar*, 2012.

[11] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective Straggler Mitigation: Attack of the Clones. In *USENIX NSDI*, 2013.

[12] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, E. Harris, and B. Saha. Reining in the Outliers in Map-Reduce Clusters Using Mantri. In *USENIX OSDI*, 2010.

[13] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *USENIX OSDI*, 2008.

[14] G. Ananthanarayanan, M. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu. GRASS: Trimming Stragglers in Approximation Analytics. In *USENIX NSDI*, 2014.

[15] Hadoop. http://hadoop.apache.org.

[16] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[17] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems (TOCS)*, 21(2):207–233, 2003.

[18] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an m/gi/1. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 238–249. ACM, 2003.

[19] A. Wierman. Fairness and scheduling in single server queues. *Surveys in Operations Research and Management Science*, 16(1):39–48, 2011.

[20] H. Chen, J. Marden, and A. Wierman. On the Impact of Heterogeneity and Back-end Scheduling in Load Balancing Designs. In *INFOCOM*. IEEE, 2009.

[21] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris. Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters. In *EuroSys*, 2011.

[22] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, C. R. Das. Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters. In *ACM SOCC*, 2011.

[23] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: Coordinated Memory Caching for Parallel Jobs. In *USENIX NSDI*, 2012.

[24] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica. The Case for Tiny Tasks in Compute Clusters. In *USENIX HotOS*, 2013.

[25] Hadoop Distributed File System. http://hadoop.apache.org/hdfs.

[26] Hadoop Slowstart. https://issues.apache.org/jira/browse/MAPREDUCE-1184/.

[27] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica. Disk Locality Considered Irrelevant. In *USENIX HotOS*, 2011.

[28] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *USENIX NSDI*, 2011.

[29] Hadoop Capacity Scheduler. http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html.

[30] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K. Wu, and A. Balmin. FLEX: a Slot Allocation Scheduling Optimizer for MapReduce Workloads. In *Middleware 2010*. Springer, 2010.

[31] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair Scheduling for Distributed Computing Clusters. In *ACM SOSP*, 2009.

[32] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *ACM EuroSys*, 2010.

[33] J. Tan, X. Meng, and L. Zhang. Delay Tails in MapReduce Scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 2012.

[34] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On Scheduling in Map-reduce and Flow-shops. In *ACM SPAA*, 2011.

[35] Hopper Technical Report. https://sites.google.com/site/hoppertechreport2014/.