Optimizing Resource Management in Cloud Analytics Services

Thesis by
Xiaoqi Ren

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2018
Defended May 15th, 2018

Xiaoqi Ren
ORCID: 0000-0002-1121-9046

ACKNOWLEDGEMENTS

First, I would like to express my deepest gratitude to my advisor, Professor Adam Wierman. He has been a great advisor to me throughout my Ph.D. It is really an invaluable experience working with him and learning from him. He has always been enthusiastic about research. He encouraged us to explore projects based on our interests instead of being driven by funding. He engaged deeply into our projects and provided excellent guidance. His insightful vision, rigorous thinking, and effective presentations have profoundly influenced me to be a better scholar. Professor Wierman always supported me to overcome the obstacles and encouraged me to be brave and step out of my safe zone. He also gave enormous help with planning my career path. I sincerely appreciate the amazing impact he has had on me, which makes me stronger and more confident as a researcher.

Next, I would like to thank my thesis committee members, Professor Steven Low, Professor Mani Chandy and Professor Yisong Yue, for all of their guidance through this process. Their ideas and feedbacks have been absolutely invaluable. Also, I am grateful to my mentors during my internships at Microsoft Research: Yuxiong He, Sameh Elnikety, Kathryn S McKinley and Christian Konig. And I would like to thank my collaborators: Palma London, Juba Ziani, Mohammad A. Islam, Shaolei Ren, Xiaorui Wang, Ganesh Ananthanarayanan, Minlan Yu, Niangjun Chen, Michael Chien-Chun Hung, and Ion Stoica. It has been a great pleasure to work with them. They have always provided insightful discussions and constructive suggestions.

I have greatly enjoyed the opportunity to study in the Department of Computing and Mathematical Sciences at Caltech, which provides a supportive environment in which students can fully focus on research. It is wonderful to interact with so many intelligent professors and outstanding students. I would also like to thank the helpful administrative staff in our department, especially Sheila Shull, Sydney Garstang, and Maria Lopez.

Finally, I would like to thank my family for providing a loving and supportive environment for me. I want to thank my parents for their understanding and belief in me during the past few years. Their support and encouragement was what made this thesis possible.

ABSTRACT

The fundamental challenge in the cloud today is how to build and optimize machine learning and data analytical services. Machine learning and data analytical platforms are changing computing infrastructure from expensive private data centers to easily accessible online services. These services pack user requests as jobs and run them on thousands of machines in parallel in geo-distributed clusters. The scale and the complexity of emerging jobs lead to increasing challenges for the clusters at all levels, from power infrastructure to system architecture and corresponding software framework design.

These challenges come in many forms. Today's clusters are built on commodity hardware and hardware failures are unavoidable. Resource competition, network congestion, and mixed generations of hardware make the hardware environment complex and hard to model and predict. Such heterogeneity becomes a crucial roadblock for efficient parallelization on both the task level and job level. Another challenge comes from the increasing complexity of the applications. For example, machine learning services run jobs made up of multiple tasks with complex dependency structures. This complexity leads to difficulties in framework designs. The scale, especially when services span geo-distributed clusters, leads to another important hurdle for cluster design. Challenges also come from the power infrastructure. Power infrastructure is very expensive and accounts for more than 20% of the total costs to build a cluster. Power sharing optimization to maximize the facility utilization and smooth peak hour usages is another roadblock for cluster design.

In this thesis, we focus on solutions for these challenges at the task level, on the job level, with respect to the geo-distributed data cloud design and for power management in colocation data centers.

At the task level, a crucial hurdle to achieving predictable performance is stragglers, i.e., tasks that take significantly longer than expected to run. At this point, speculative execution has been widely adopted to mitigate the impact of stragglers in simple workloads. We apply straggler mitigation for approximation jobs for the first time. We present GRASS, which carefully uses speculation to mitigate the impact of stragglers in approximation jobs. GRASS's design is based on the analysis of a model we develop to capture the optimal speculation levels for approximation jobs. Evaluations with production workloads from Facebook and Microsoft Bing in an

EC2 cluster of 200 nodes show that GRASS increases accuracy of deadline-bound jobs by 47% and speeds up error-bound jobs by 38%.

Moving from task level to job level, task level speculation mechanisms are designed and operated independently of job scheduling when, in fact, scheduling a speculative copy of a task has a direct impact on the resources available for other jobs. Thus, we present Hopper, a job-level speculation-aware scheduler that integrates the tradeoffs associated with speculation into job scheduling decisions based on a model generalized from the task-level speculation model. We implement both centralized and decentralized prototypes of the Hopper scheduler and show that 50% (66%) improvements over state-of-the-art centralized (decentralized) schedulers and speculation strategies can be achieved through the coordination of scheduling and speculation.

As computing resources move from local clusters to geo-distributed cloud services, we are expecting the same transformation for data storage. We study two crucial pieces of a geo-distributed data cloud system: data acquisition and data placement. Starting from developing the optimal algorithm for the case of a data cloud made up of a single data center, we propose a near-optimal, polynomial-time algorithm for a geo-distributed data cloud in general. We show, via a case study, that the resulting design, Datum, is near-optimal (within 1.6%) in practical settings.

Efficient power management is a fundamental challenge for data centers when providing reliable services. Power oversubscription in data centers is very common and may occasionally trigger an emergency when the aggregate power demand exceeds the capacity. We study power capping solutions for handling such emergencies in a colocation data center, where the operator supplies power to multiple tenants. We propose a novel market mechanism based on supply function bidding, called COOP, to financially incentivize and coordinate tenants' power reduction for minimizing total performance loss while satisfying multiple power capping constraints. We demonstrate that COOP is "win-win", increasing the operator's profit (through oversubscription) and reducing tenants' costs (through financial compensation for their power reduction during emergencies).

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] G. Ananthanarayanan, M. C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu. "GRASS: Trimming Stragglers in Approximation Analytics". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA: USENIX Association, 2014, pp. 289–302. ISBN: 978-1-931971-09-6.
Adapted into Chapter 2 of this thesis. X. Ren contributed to the conception of the project, proposing the model and analyzing the performance, and writing the manuscript.

[2] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu. "Hopper: Decentralized Speculation-aware Cluster Scheduling at Scale". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. London, United Kingdom: ACM, 2015, pp. 379–392. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787481.
Adapted into Chapter 3 of this thesis. X. Ren contributed to the conception of the project, proposing the model and analyzing the performance, and writing the manuscript.

[3] X. Ren, P. London, J. Ziani, and A. Wierman. "Datum: Managing Data Purchasing and Data Placement in a Geo-Distributed Data Market". In: *IEEE/ACM Transactions on Networking* 26.2 (2018), pp. 893–905. DOI: 10.1109/TNET.2018.2811374.
Adapted into Chapter 4 of this thesis. X. Ren contributed to the conception of the project, proposing the model and analyzing the performance, and writing the manuscript.

[4] N. Chen, X. Ren, S. Ren, and A. Wierman. "Greening Multi-Tenant Data Center Demand Response". In: *SIGMETRICS Perform. Eval. Rev.* 43.2 (Sept. 2015), pp. 36–38. ISSN: 0163-5999. DOI: 10.1145/2825236.2825252.
Adapted into Chapter 5 of this thesis. X. Ren contributed to the conception of the project, proposing the model and analyzing the performance, and writing the manuscript.

[5] M. A. Islam, X. Ren, S. Ren, A. Wierman, and X. Wang. "A market approach for handling power emergencies in multi-tenant data center". In: *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 432–443. DOI: 10.1109/HPCA.2016.7446084. URL: https://ieeexplore.ieee.org/document/7446084/.
Adapted into Chapter 5 of this thesis. X. Ren contributed to the conception of the project, proposing the model and analyzing the performance, and writing the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Number *Page*

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In the last two decades, the emerging of big data has driven enormous push in the technology developments of both distributed systems and machine learning. In the era of big data, the requirement to provide predictable and scalable services has led to a remarkable evolution of large scale data analytics frameworks. From MapReduce [1] to Spark [2] then to Flink [3], from support for simple batch jobs to support for steam processing to integrated machine learning and graph libraries, there is a trend towards broader and more general system design. Meanwhile, the fast evolution of system frameworks inevitably leads to significant challenges from scalability to compatibility with existing systems and new applications. In the following, we first investigate the evolution of the data analytics frameworks , and then highlight the challenges to the design of large scale data analytics frameworks. Finally we give an overview of the contributions this thesis make toward these challenges.

1.1 THE EVOLUTION OF LARGE SCALE DATA ANALYTICS FRAMEWORKS

About twenty years ago, to handle the challenge of processing massive amount of data in special-purpose computations such as web index computing, engineers in Google designed a new data processing framework called MapReduce [1]. MapReduce can be viewed as an abstraction that allows users to perform logically simple computations without taking care of the messy details of distributed computing. In general, data processing in MapReduce has three phases, map phase, shuffle phase, and reduce phase. In map phase, input data is divided into small data chunks, and each data chunk is sent to a machine slot (i.e., mapper) for execution. The output data of the mappers are in the form of key-value pairs, and are sorted and grouped by keys in the shuffle phase. The shuffled data are fed to reduce machine slots (i.e., reducers) in the reduce phase for further processing. After having been published in 2004 by Google Labs, this simple but powerful programming paradigm attracted enormous attention.

MapReduce quickly expanded beyond Google. In 2006, Hadoop [4], an open source version of MapReduce, was developed by Doug Cutting and Mike Cafarella. The core of Hadoop consists of two parts, (1) the storage architecture, Hadoop Distributed File System (HDFS) [5], which is inspired by the Google File System (GFS) [6], and (2) the processing model, MapReduce. Hadoop MapReduce quickly became the most popular open source implementation of MapReduce. Yahoo!, Facebook and many others soon adopted Hadoop to build their large cloud computing clusters [7]. In 2010, Facebook announced that they had the largest Hadoop cluster in the world with 21 PB of storage. The data volume had grown to 100 PB in mid 2012 and was reported to continuously grow by roughly half a PB per day later that year. Hadoop adoption had become widespread as of 2013: more than half of the Fortune 50 used Hadoop [8].

As Hadoop became widely used, the early adopters started to notice the limitations of the classical MapReduce framework, including lack of scalability, resource underutilization and the lack of support for complex applications.

These issues are mainly due to two design choices. (1) MapReduce uses a single centralized JobTracker process to constantly track all the resource management and allocation and perform coordination of all jobs and their tasks running on the cluster. Besides the JobTracker, a number of TaskTracker processes are used to assign tasks and periodically report the progresses to the JobTracker. The highly centralized structure of the JobTracker results in scalability issue, especially for real time data analytics. (2) To ease the workload of the JobTracker and TaskTracker, in MapReduce the computational resources on each node are divided into fixed number of map slots and reduce slots, which lead to significant resource underutilization, especially when a job requires an unequal number of mappers and reducers.

To address these issues, YARN [9] was integrated into Hadoop in 2014. In the YARN architecture, JobTracker is replaced by a global ResourceManager which is in charge of inter-application level resource management and a number of ApplicationMasters which are responsible for intra-application level resource management for each application respectively. NodeManagers replace the previous TaskTrackers for resource container control. Such a decoupling reduces the overhead of the resouce management and also supports applications besides traditional MapReduce jobs given that ResouceManager and NodeManagers are designed to be independent of the running tasks. From then on, Hadoop MapReduce, Hadoop YARN, Hadoop Distributed File System and Hadoop Common (the common utilities that support the

other Hadoop modules) become the core modules of the Apache Hadoop project [4].

While MapReduce is well suited for batch processing, its deficiency for real time data analytics on complicated applications, especially for applications that reuse the same working set of data across multiple MapReduce jobs, is also widely noticed by researchers and developers. For example, iterative machine learning jobs experience significant delay in MapReduce because each iteration is considered as an individual MapReduce job and, in the MapReduce framework, each job must load data from the disk independently. To handle this challenge, M. Zaharia et al. developed Apache Spark in 2010 [2]. Spark shares a similar programming model to MapReduce, but adopts an abstraction called resilient distributed dataset (RDD) which allows users to explicitly cache data sets in memory across machines and reuse them across multiple MapReduce jobs (MapReduce-like parallel operations). Spark can run either in stand-alone mode or combined with Hadoop as a replacement for MapReduce. Unlike MapReduce which is best suited for batch processing, Spark is designed to handle both batch processing and stream processing. Its in-memory data processing ensures fast speed and thus makes Spark one of the most popular real time data analytic frameworks.

The broad success of Hadoop as well as the emerging of many other challenges have led an enormous number of developers to engage in Hadoop related projects. All these projects comprise a so-called “Hadoop ecosystem." Systems in the the Hadoop ecosystem cover a variety of aspects and target a lot of different applications. For example, Pig [10] and Hive [11] offer SQL-style high-level data manipulation constructed on top of Hadoop. Storm [12] provides stream processing for real time data analytics. Flink [3] aims at both batch and stream processing with a unified architecture based on data flow abstraction. Tez [13] is built atop YARN and adds support to jobs with DAGs. Kafka [14] is a distributed publish-subscribe system for processing large amounts of streaming data with coordination support from Zookeeper. Also various NoSQL or SQL databases are designed and developed to provided data storage support. The boom of Hadoop and its corresponding ecosystem provides rich tools to handle big data challenges. However, there is an emerging trend towards the design of more general data analytics systems. The inter- and intra- systems optimization as well as the exponentially increasing data volume and the emerging of new data with different characteristics pose new challenges every day. We focus on these challenges in the next section.

1.2 Challenges to the Design of Analytics Frameworks

Even with the success of existing various industrial systems, as the volume of data continues to grow, the scale of the clusters expands and complicated applications impose new requirements on the systems, which lead to new challenges to the design of analytics frameworks.

One significant goal for data analytics frameworks is to provide predictable performance. As the scale and complexity of clusters increase, hard-to-model systemic interactions that degrade the performance of tasks become common. Consequently, many tasks become “stragglers”, i.e., running slower than expected, leading to significant unpredictability (and delay) in job completion times. Dealing with stragglers is a crucial design component that has received widespread attention across prior studies [15, 16]. The dominant technique to mitigate stragglers is speculation, which works by launching speculative copies for the slower tasks, where a speculative copy is simply a duplicate of the original task. It then becomes a race between the original and the speculative copies. Such techniques are state of the art and deployed in production clusters at Facebook and Microsoft Bing, thereby significantly speeding up jobs [1, 15, 17, 18]. However, current straggler mitigation algorithms are mainly focused on the task level without considering how speculation will affect job level scheduling. Speculation policies deployed today are all designed and operated independently of job scheduling; schedulers simply allocate slots to speculative copies in a best-effort" fashion, e.g., [1, 17, 19, 20]. Also the existing speculation algorithms are lack of support for complicated job structures, such as DAG, and only focus on performing speculation mitigation on current wave of tasks [1, 17, 19].

Another hurdle to system optimization is the increasing complexity of the applications. Fifteen years ago, MapReduce was a huge success in regenerating the Google’s index of the World Wide Web. But its lack of support for complicated applications such as machine learning iterative jobs and graph processing jobs with DAGs motivated the development of other systems/modules such as Spark [2], YARN [9] and Tez [13]. These systems/modules are combined to provide generic frameworks to run complicated jobs, but fine-granularity optimization targeting different job types is still a challenge. Take an emerging new type of job, approximation jobs, as an example [21–23]. With the deluge of data, analytics applications no longer require the processing of entire datasets. Instead, they choose to trade off accuracy for response time and obtain approximate results early from just part of the dataset.

Such approximation jobs require schedulers to prioritize the appropriate subset of their tasks depending on the approximation criteria, while distributed systems are generally designed to *evenly* prioritize each tasks. Balancing between these two provides new challenges for system design.

Achieving lower latency is an increasing challenge as large scale data analytics frameworks are shifting towards shorter task duration and higher degrees of parallelization. In 2004, the scale of task duration was more than 10 min, while in 2010, for Spark in memory query, the scale of task duration was already shortened to hundreds of milliseconds [20]. We expect the trend to continue in the future generations of frameworks with task duration being even smaller. In this situation, controlling system overheads becomes of importance for real time data analytics design. Today's state-of-the-art frameworks use distributed schedulers to achieve lightweight scheduling while maintaining low latency [16, 20, 24]. In such frameworks, each scheduler only knows partial information about jobs and working machines. Optimizing such systems, especially with regards to efficient straggler mitigation, is still challenging.

The most widely used scheduling approach in clusters today is based on fairness, which can be thought of as equal sharing (or weighted sharing) of the available resources among jobs (or cluster users) [25]. The equal sharing strategy guarantees isolation in resource management, in the sense that users are guaranteed to receive their fair shares and no starvation can exist. However, fairness, of course, comes with the cost of performance inefficiencies. How to balance fairness and efficiency to avoid starvation for jobs while achieving high efficiency is another system design challenge.

As cloud servers are widely located in geo-distributed systems, analysis and optimization of data stored in geographically distributed data centers has received increasing attention [26–28]. Bandwidth constraints as well as latency are the two main challenges for system designs in this context, and a number of system designs have been proposed [26–28]. Data acquisition and data pricing have been studied extensively [29–32]. However how to cost-effectively combine the data acquisition with data placement imposes new challenges to geo-distributed system designs.

Power management is another challenge that has been studied widely [33–36]. Data centers are power hogs. In 2014, data centers in the U.S. consumed an estimated 70 billion kWh, which accounts for 1.8% of total U.S. electricity consumption [37]. On the other hand, data centers usually have the flexibility of decreasing electricity

consumption for a short period of time with power techniques such as load migration/scheduling [38]. Such flexibility makes data centers promising resources for demand response, particularly for emergency demand response, which saves the power grid from incurring blackouts during emergency situations. Existing studies mostly focus on owner operated data centers (e.g., Google) whose operators have full control over both servers and facilities. But multi-tenant colocation data centers have been investigated much less frequently. In a colocation data center (simply called "colocation" or "colo"), multiple tenants deploy and keep full control of their own physical servers in a shared space, while the colo operator only provides facility support (e.g., high-availability power and cooling). Colos are less studied than owner-operated data centers, but they are actually more common in practice. Colos offer data center solutions to many industry sectors, and serve as physical home to many private clouds, medium-scale public clouds (e.g., VMware) [39], and content delivery providers (e.g., Akamai). Further, a recent study shows that colos consume nearly 40% of data center energy in the U.S., while Google-type data centers collectively account for less than 8%, with the remaining going to enterprise in-house data centers [40]. With such huge potential, efficient power management for colos becomes an important challenge in data center design.

1.3 Overview of This Thesis

This thesis is divided into four components. In Chapter 2, we focus on the task-level data analytics framework optimization for approximation jobs. In Chapter 3, we study the job-level optimization with a joint design of job scheduling and speculation mitigation. In Chapter 4, we investigate the design of the geo-distributed data cloud with joint optimization of data acquisition and data placement. Finally in Chapter 5, we study power management in collocation data centers.

Chapter 2: speculation-aware cluster scheduling at the task level

In this chapter, we focus on the task-level data analytics framework optimization for approximation jobs. In big data analytics, timely results, even if based on only part of the data, are often good enough. For this reason, approximation jobs, which have deadline or error bounds and require only a subset of their tasks to complete, are projected to dominate big data workloads. Straggler tasks are an important hurdle when designing approximate data analytic frameworks, and the widely adopted approach to deal with them is speculative execution.

In this chapter, we present GRASS, which carefully uses speculation to mitigate the impact of stragglers in approximation jobs. We develop an analytic model to analyze the optimal speculation level for approximation jobs. GRASS's design is based on the guidelines derived from the analysis. GRASS delicately balances immediacy of improving the approximation goal with the long term implications of using extra resources for speculation. Evaluations with production workloads from Facebook and Microsoft Bing in an EC2 cluster of 200 nodes shows that GRASS increases accuracy of deadline-bound jobs by 47% and speeds up error-bound jobs by 38%. GRASS's design also speeds up exact computations (zero error-bound), making it a unified solution for straggler mitigation. This work summarizes the result in [19].

Chapter 3: speculation-aware cluster scheduling on the job level

In this chapter, we study the job-level data analytics framework optimization with a joint design of job scheduling and speculation mitigation. As clusters continue to grow in size and complexity, providing scalable and predictable performance is an increasingly important challenge. At this point, speculative execution has been widely adopted to mitigate the impact of stragglers. However, speculation mechanisms are designed and operated independently of job scheduling when, in fact, scheduling a speculative copy of a task has a direct impact on the resources available for other jobs. In this work, we present Hopper, a job scheduler that is speculation-aware, i.e., that integrates the tradeoffs associated with speculation into job scheduling decisions. We generalize the model in Chapter 2 from task level to job level and design Hopper based on that. A knob to balance fairness and efficiency and solutions to jobs with DAGs and heterogeneous jobs are also provided in the design. We implement both centralized and decentralized prototypes of the Hopper scheduler and show that 50% (66%) improvements over state-of-the-art centralized (decentralized) schedulers and speculation strategies can be achieved through the *coordination* of scheduling and speculation. This work summarizes the result in [41].

Chapter 4: network-aware geo-distributed cluster scheduling

This chapter studies two design tasks faced by a geo-distributed cloud data market: which data to purchase (data purchasing) and where to place/replicate the data for delivery (data placement). We show that the joint problem of data purchasing and data placement within a data cloud can be viewed as a facility location problem, and is thus NP-hard. However, we give a provably optimal algorithm for the case of a

data cloud made up of a single data center, and then generalize the structure from the single data center setting in order to develop a near-optimal, polynomial-time algorithm for a geo-distributed data cloud. The resulting design, Datum, decomposes the joint purchasing and placement problem into two subproblems, one for data purchasing and one for data placement, using a transformation of the underlying bandwidth costs. We show, via a case study, that Datum is near-optimal (within 1.6%) in practical settings. This work summarizes the result in [42].

Chapter 5: Power Capping in Colocation Data Centers

This chapter focuses on power management in data centers. Power oversubscription in data centers may occasionally trigger an emergency when the aggregate power demand exceeds the capacity. Handling such an emergency requires a graceful power capping solution that minimizes the performance loss. In this chapter, we study power capping in a colocation data center where the operator supplies power to multiple tenants who manage their own servers. Unlike owner-operated data centers, the operator lacks control over tenants' servers. To address this challenge, we propose a novel market mechanism based on supply function bidding, called COOP, to financially incentivize and coordinate tenants' power reduction for minimizing total performance loss (quantified in performance cost) while satisfying multiple power capping constraints. We first provide the theoretical analysis of our mechanism under a simple case with data center level power capping constraint only and then generalize our mechanism to the multi-level power capping problem. We build a prototype to show that COOP is efficient in terms of minimizing the total performance cost, even compared to the ideal but infeasible case that assumes the operator has full control over tenants' servers. We also demonstrate that COOP is "win-win", increasing the operator's profit (through oversubscription) and reducing tenants' costs (through financial compensation for their power reduction during emergencies). This work summarizes the result in [43, 44].

CHAPTER 2

SPECULATION-AWARE CLUSTER SCHEDULING AT THE TASK LEVEL

Large scale data analytics frameworks automatically compose *jobs* operating on large data sets into many small *tasks* and execute them in parallel on *compute slots* on different machines. A key feature catalyzing the widespread adoption of these frameworks is their ability to guard against failures of tasks, both when tasks fail outright as well as when they run slower than the other tasks of the job. Dealing with the latter, referred to as *stragglers*, is a crucial design component that has received widespread attention across prior studies [15, 17, 18].

The dominant technique to mitigate stragglers is *speculation*—launching speculative copies for the slower tasks, where a speculative copy is simply a duplicate of the original task. It then becomes a race between the original and the speculative copies. Such techniques are state of the art and deployed in production clusters at Facebook and Microsoft Bing, thereby significantly speeding up jobs.

Approximation jobs are starting to see considerable interest in data analytics clusters [21–23]. These jobs are based on the premise that providing a timely result, even if only on part of the dataset, is more important than processing the entire data. These jobs tend to have *approximation bounds* on two dimensions—deadline and error [45]. *Deadline-bound* jobs strive to maximize the accuracy of their results within a specified time deadline. *Error-bound* jobs, on the other hand, strive to minimize the time taken to reach a specified error limit in the result. Typically, approximation jobs are launched on a large dataset and require only a *subset* of their tasks to finish based on the bound [46–48].

Our focus in this chapter is on the problem of task-level speculation for approximation jobs.[1] Traditional speculation techniques for straggler mitigation face a

[1]Note that an error-bound job with error of zero is the same as an exact job that requires all its tasks to complete. Hence, by focusing on approximation jobs, we automatically subsume exact computations.

fundamental limitation when dealing with approximation jobs, since they do not take into account approximation bounds. Ideally, when the job has many more tasks than compute slots, we want to prioritize those tasks that are likely to complete within the deadline or those that contribute the earliest to meeting the error bound. By not considering the approximation bounds, state-of-the-art straggler mitigation techniques in production clusters at Facebook and Bing fall significantly short of optimal mitigation. They are 48% lower in average accuracy for deadline-bound jobs and 40% higher in average duration of error-bound jobs.

Optimally prioritizing tasks of a job to slots is a classic scheduling problem with known heuristics [49–51]. These heuristics, unfortunately, do not directly carry over to our scenario for the following reasons. First, they calculate the optimal ordering statically. Straggling of tasks, on the other hand, is unpredictable and necessitates dynamic modification of the priority ordering of tasks according to the approximation bounds. Second, and most importantly, traditional prioritization techniques assign tasks to slots assuming every task occupies only one slot. Spawning a speculative copy, however, leads to the same task using two (or multiple) slots simultaneously. Hence, this distills our challenge to *achieving the approximation bounds by dynamically weighing the gains due to speculation against the cost of using extra resources for speculation*.

Scheduling a speculative copy helps make immediate progress by finishing a task faster. However, while not scheduling a speculative copy results in the task running slower, many more tasks may be completed using the saved slot. To understand this *opportunity cost*, consider a cluster with one unoccupied slot and a straggler task. Letting the straggler complete takes five more time units while a new copy of it would take four time units. While scheduling a speculative copy for this straggler speeds it up by one time unit, if we were not to, that slot could finish another task (taking five time units too).

This simple intuition of opportunity cost forms the basis for our two design proposals. First, Greedy Speculative (GS) scheduling is an algorithm that *greedily* picks the task to schedule next (original or speculative) that most improves the approximation goal at that point. Second, Resource Aware Speculative (RAS) scheduling considers the opportunity cost and schedules a speculative copy only if doing so saves both time *and* resources.

These two designs are motivated by first principles analysis within the context of a theoretical model for studying speculative scheduling. An important guideline from

our model is that the value of being greedy (GS) increases for smaller jobs while considering opportunity cost of speculation (RAS) helps for larger jobs. As our model is generic, a nice aspect is that the guideline holds not only for approximation jobs but also for exact jobs that require all their tasks to complete.

We use the above guideline to dynamically combine GS and RAS, which we call GRASS. At the beginning of a job's execution, GRASS uses RAS for scheduling tasks. Then, as the job gets *close* to its approximation bound, it switches to GS, since our theoretical model suggests that the opportunity cost of speculation diminishes with fewer unscheduled tasks in the job. GRASS learns the point to switch from RAS to GS using job and cluster characteristics.

We demonstrate the generality of GRASS by implementing it in both Hadoop [4] (for batch jobs) and Spark [52] (for interactive jobs). We evaluate GRASS using production workloads from Facebook and Bing on an EC2 cluster with 200 machines. GRASS increases accuracy of deadline-bound jobs by 47% and speeds up error-bound jobs by 38% compared to state-of-the-art straggler mitigation techniques deployed in these clusters (LATE [18] and Mantri [17]). In fact, GRASS results in near-optimal performance. In addition, GRASS also speeds up exact jobs, that require all their tasks to complete, by 34%. Thus, it is a *unified speculation solution for both approximation as well as exact computations*.

2.1 Challenges and Opportunities

Before presenting our system design, it is important to understand the challenges and opportunities for speculating straggler tasks in the context of approximation jobs.

2.1.1 Approximation Jobs

Increasingly, with the deluge of data, analytics applications no longer require processing the entire datasets. Instead, they choose to trade off accuracy for response time. *Approximate* results obtained early from just part of the dataset are often *good enough* [21–23]. Approximation is explored across two dimensions—time for obtaining the result (deadline) and error in the result [45].

- *Deadline-bound* jobs strive to maximize the accuracy of their result within a specified time limit. Such jobs are common in real-time advertisement

systems and web search engines. Generally, the job is spawned on a large dataset and accuracy is proportional to the fraction of data processed [46–48] (or tasks completed, for ease of exposition).

- *Error-bound* jobs strive to minimize the time taken to reach a specified error limit in the result. Again, accuracy is measured in the amount of data processed (or tasks completed). Error-bound jobs are used in scenarios where the value in reducing the error below a limit is marginal, *e.g.,* counting the number of cars crossing a section of a road to the nearest thousand is sufficient for many purposes.

Approximation jobs require schedulers to *prioritize the appropriate subset of their tasks* depending on the deadline or error bound. Prioritization is important for two reasons. First, due to cluster heterogeneities [15, 18, 53], tasks take different durations even if assigned the same amount of work. Second, jobs are often *multi-waved*, i.e., their number of tasks is much greater than the available compute slots, thereby they run only a fraction of their tasks at a time [54]. For example, when a job with 1000 tasks is given only 100 slots simultaneously (due to, say, fair scheduling), it runs only one-tenth of its tasks at a time. These tasks, though, are independent and can be scheduled in any order. The trend of multi-waved jobs is expected to grow with smaller tasks [55].

2.1.2 Challenges

The main challenge in prioritizing tasks of approximation jobs arises due to some of them *straggling*. Even after applying many proactive techniques, in production clusters in Facebook and Microsoft Bing, the average job's slowest task is eight times slower than the median.[2] It is difficult to model all the complex interactions in clusters to prevent stragglers [15, 57]. Ananthanarayanan et al. (Section 2.1.2 in [15]) also show that blacklisting machines based on their likeliness to cause stragglers (in both the short- as well as long-term) has little benefits; machines are neither consistently problematic nor exhibit simple correlations with task durations.

The widely adopted technique to deal with straggler tasks is *speculation*. This is a reactive technique that spawns speculative copies for tasks deemed to be straggling. The earliest among the original and speculative copies is picked while the rest are

[2]Task durations are normalized by their input sizes to be resistant to data skews [17, 56].

killed. While scheduling a speculative copy makes the task finish faster and thereby increases accuracy, they also compete for compute slots with the unscheduled tasks.

Therefore, our problem is to *dynamically prioritize tasks based on the deadline/error-bound while choosing between speculative copies for stragglers and unscheduled tasks*. This problem is, unfortunately, NP-Hard and devising good heuristics (i.e., with good approximation factors) is an open theoretical problem.

2.1.3 Potential Gains

Given the challenges posed by stragglers discussed above, it is not surprising that the potential gains from mitigating their impact are significant. To highlight this we use a simulator with an optimal bin-packing scheduler. Our baselines are the the state-of-the-art mitigation strategies (LATE [18] and Mantri [17]) in the production clusters. Optimally prioritizing the tasks while correctly balancing between speculative copies and unscheduled tasks presents the following potential gains. Deadline-bound jobs improve their accuracy by 48% and 44%, in the Facebook and Bing traces, respectively. Error-bound jobs speed up by 32% and 40%. We next develop two online heuristics to achieve these gains.

2.2 Speculation Algorithm Design

The key choice made by a cluster scheduling algorithm is to pick the next task to schedule given a vacant slot. Traditionally, this choice is made among the set of tasks that are queued; however when speculation is allowed, the choice also includes speculative copies of tasks that are already running. This extra flexibility means that a design must determine a prioritization that carefully weighs the gains from speculation against the cost of extra resources while still meeting the approximation goals. Thus, we first focus on tradeoffs in the design of the speculation policy. Specifically, using both small examples and analytic modeling we motivate the use of two simple heuristics, Greedy Speculative (GS) scheduling and Resource Aware Speculative (RAS) scheduling that together make up the core of GRASS.

2.2.1 Speculation Alternatives

For simplicity, we first introduce GS and RAS in the context of deadline-bound jobs and then briefly describe how they can be adapted to error-bound jobs.

2.2.1.1 Deadline-bound Jobs

If speculation was not allowed, there is a natural, well-understood policy for the case of deadline-bound jobs: Shortest Job First (SJF), which schedules the task with the smallest processing time. In many settings, SJF can be proven to minimize the number of incomplete tasks in the system, and thus maximize the number of tasks completed, at all points of time among the class of non-preemptive policies [49, 50]. Thus, without speculation, SJF finishes the most tasks before the deadline.

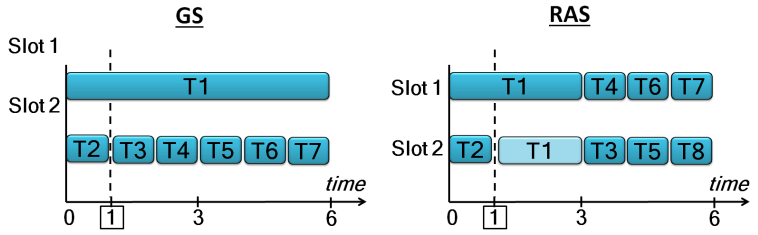
If one extends this idea to the case where speculation is allowed, then a natural approach is to allow the currently running tasks to also be placed in the queue, and to choose the task with the smallest size, i.e., t_{new} (requiring, of course, that the task finishes before the deadline). If the chosen task has a copy currently running, we check that the speculative copy being considered provides a benefit, i.e., $t_{new} < t_{rem}$. So, the next task to run is still chosen according to SJF, only now speculative copies are also considered. We term this policy *Greedy Speculative (GS) scheduling*, because it picks the next task to schedule *greedily*, i.e., the one that will finish the quickest, and thus improve the accuracy the earliest *at present*.

Figure 2.1 (left) presents an illustration of GS for a simple job with nine tasks and two concurrent slots. Tasks T1 and T2 are scheduled first, and when T2 finishes, the t_{rem} and t_{new} values are as indicated. At this point, GS schedules T3 next as it is the one with the lowest t_{new}, and so forth. Assuming the deadline was set to 6 time units, the obtained accuracy is $\frac{7}{9}$ (or 7 completed tasks).

Picking the earliest task to schedule next is often optimal when a job has no unscheduled tasks (i.e., either single-waved jobs or the last wave of a multi-waved job). However, when there are unscheduled tasks it is less clear. For example, in Figure 2.1 (right) if we schedule a speculative copy of T1 when T2 finished, instead of T3, 8 tasks finish by the deadline of 6 time units.

The previous example highlights that running a speculative copy has resource implications which are important to consider. If the speculative copy finishes early, both slots (of the speculative copy and the original) become available sooner to start the other tasks. This *opportunity cost* of speculation is an important tradeoff to consider, and leads to the second policy we consider: *Resource Aware Speculative (RAS) scheduling*.

To account for the opportunity cost of scheduling a speculative copy, RAS speculates only if it saves both time *and* resources. Thus, not only must t_{new} be less than t_{rem}



Task ID	T1	T2	T3	T4	T5	T6	T7	T8	T9
t_{rem}	5	---	--	---	---	---	---	---	---
t_{new}	2	---	1	1	1	1	1	1	3

Figure 2.1: GS and RAS for a deadline-bound job with 9 tasks. The t_{rem} and t_{new} values are when T2 finishes. The example illustrates deadline values of 3 and 6 time units.

to spawn a speculative copy but the sum of the resources used by the speculative and original copies, when running simultaneously, must be less than letting just the original copy finish. In other words, for a task with c running copies, its resource savings, defined as $c \times t_{rem} - (c + 1) \times t_{new}$, must be positive.

By accounting for the opportunity cost of resources, RAS can out-perform GS in many cases. As mentioned earlier, an example is given in Figure 2.1 where RAS achieves an accuracy of $\frac{8}{9}$ versus GS's $\frac{7}{9}$ in the deadline of 6 time units. This improvement comes because, when T2 finishes, speculating on T1 saves 1 unit of resource.

However, RAS is not uniformly better than GS. In particular, RAS's cautious approach can backfire if it overestimates the opportunity cost. In the same example in Figure 2.1, if the deadline of the job were reduced from 6 time units to 3 time units instead, GS performs better than RAS. At the end of 3 time units, GS has led to three completed tasks while RAS has little to show for its resource gains by speculating T1.

As the example alludes to, the value of the deadline and the number of waves are two important factors that impact whether GS or RAS is a better choice. A third important factor, which we discuss later in Section 2.4.1, is the estimation accuracy of t_{rem} and t_{new}.

Pseudocode 1 describes the details of GS and RAS. The set T consists of all the running and unscheduled tasks of the jobs. There are two stages in the scheduling process: (i) *Pruning Stage:* In this stage (lines 5 – 12), tasks that are not slated to complete by the deadline are removed from consideration. Further, GS removes

```
1:  procedure DEADLINE(⟨Task⟩ T, float δ, bool OC)
                                          ▷ OC = 1 → use RAS; 0 → use GS
2:      if OC then
3:          for each Task t in T do
4:              if t.running then
        t.saving = t.c × t.t_rem − (t.c+1) × t_new
                                                        ▷ PRUNING STAGE
        δ' ← Remaining Time to δ
        ⟨Task⟩Γ ← φ
5:      for each Task t in T do
6:          if t.t_new > δ' then continue                ▷ Exceeds deadline
7:          if OC then
8:              if t.saving > 0 then Γ.add(t)
9:          else
10:             if t.running then
11:                 if t.t_new < t.t_rem then Γ.add(t)
12:             else Γ.add(t)
                                                        ▷ SELECTION STAGE
13:     if Γ ≠ null then
14:         if OC then SortDescending(Γ, "saving")
15:         else SortAscending(Γ, t_new)
        return Γ.first()
```

Pseudocode 1: GS and RAS algorithms for deadline-bound jobs (deadline of δ). T is the set of unfinished tasks with the following fields per task: t_{rem}, t_{new}, and a boolean "running" to denote if a copy of it is currently executing. RAS is used when OC is set. At default, both algorithms schedule the task with the lowest t_{new} within the deadline.

those tasks whose speculative copy is not expected to finish earlier than the running copy. RAS removes those tasks which do not save on resources by speculation. (*ii*) *Selection Stage:* From the pruned set, GS picks the task with the lowest t_{new} while RAS picks the task with the highest resource savings (lines 13 – 15).

2.2.1.2 Error-bound Jobs

Though error-bound jobs require a different form of prioritization than deadline-bound jobs, the speculative core of the GS and RAS algorithms are again quite natural. Specifically, the goal of error-bound jobs is to minimize the makespan of the tasks needed to achieve the error limit. Thus, instead of SJF, Longest Job First (LJF) is the natural prioritization of tasks. In particular, LJF minimizes the makespan among the class of non-preemptive policies in many settings [49, 50].

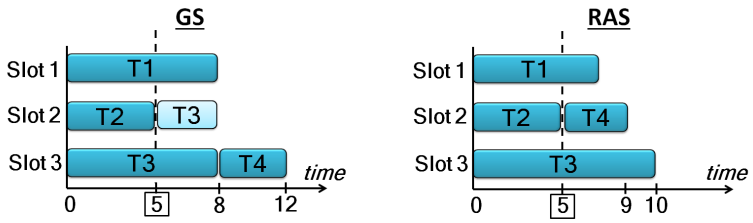
```
1:  procedure ERROR(⟨Task⟩ T, float ϵ, bool OC)
                                          ▷ OC = 1 → use RAS; 0 → use GS
                                          ▷ Error ϵ is in #tasks
2:      for each Task t in T do
    t.duration = min(t.t_rem, t.t_new)
3:          if OC then
4:              if t.running then
    t.saving = t.c ×t.t_rem − (t.c+1) × t_new
                                          ▷ PRUNING STAGE
    SortAscending(T, "duration")
    ⟨Task⟩Γ ← ϕ
5:      for each Task t in T[0 : T.count() (1 − ϵ)] do
                                          ▷ Earliest tasks
6:          if OC then
7:              if t.saving > 0 then Γ.add(t)
8:          else
9:              if t.running then
10:                 if t.t_new < t.t_rem then Γ.add(t)
11:             else Γ.add(t)
                                          ▷ SELECTION STAGE
12:     if Γ ≠ null then
13:         if OC then SortDescending(Γ, "saving")
14:         else SortDescending(Γ, t_rem)
    return Γ.first()
```

Pseudocode 2: GS and RAS speculation algorithms for error-bound jobs (error-bound of ϵ). T is the set of unfinished tasks with the following fields per task: t_{rem}, t_{new}, and a boolean "running" to denote if a copy of it is currently executing. The t_{rem} of the task is the minimum of all its running copies. RAS is used when OC is set. At default, both algorithms schedule the task with the highest t_{rem}.

This again represents a "greedy" prioritization for this setting.

Despite the above change to the prioritization of which task to schedule, the form of GS and RAS remain the same as in the case of deadline-bound jobs. In particular, speculative copies are evaluated in the same manner, e.g., RAS's criterion is still to pick the task whose speculation leads to the highest resource savings. Pseudocode 2 presents the details. The pruning stage (lines 5 – 11) will remove from consideration those tasks that are not the earliest to contribute to the desired error bound. The list of earliest tasks is based on the effective duration of every task, i.e., the minimum of t_{rem} and t_{new}. During selection (lines 12 – 14), GS picks the task with the highest t_{rem} while RAS picks the task with the highest saving.

Figure 2.2 presents an illustration of GS and RAS for an error-bound job with 6



Task ID	T1	T2	T3	T4	T5
t_{rem}	3	---	5	---	---
t_{new}	3	---	3	4	4

Figure 2.2: GS and RAS for error-bound job with 6 tasks. The t_{rem} and t_{new} values are when T2 finishes. The example illustrates error limit of 40% (3 tasks) and 20% (4 tasks).

tasks and 3 compute slots. The t_{rem} and t_{new} values are at 5 time units. GS decides to launch a copy of T3 as it has the highest t_{rem}. RAS conservatively avoids doing so. Consequently, when the error limit is high (say, 40%) GS is quicker, but RAS is better when the limit decreases (to, say, 20%).

2.2.2 Contrasting GS and RAS

To this point, we have seen that GS and RAS are two natural approaches for integrating speculation into a cluster scheduler for approximation jobs. However, the examples we have considered highlight that neither GS nor RAS is uniformly better. Natural questions to study include are these two natural approaches optimal and if they are optimal, when to apply which. In order to answer those questions and develop a better understanding of these two algorithms as well as other possible alternatives, we have developed an analytic model for speculation in approximation jobs. The model assumes wave-based scheduling and constant wave-width for a job. We present the analytic model along with formal results in the following section. The same model also inspires our design for the job-level speculation-aware cluster scheduler in Chapter 3.

2.3 Modeling and Analyzing Speculation

In this section we introduce the model and analysis which leads to the design of our scheduler GRASS. We develop the model in Section 2.3.1 and Section 2.3.2, and summarize the design guidelines from our analysis in Section 2.3.3.

Figure 2.3: Hill plot of Facebook task durations.

The model focuses on a system with S slots and one job that has T tasks[3]. Each slot can have one task scheduled to it at any time. And due to the short task duration and fast processing rate, preemption is not allowed in our system. Each of the tasks has an an i.i.d. random task completion time τ.We denote the remaining number of tasks for the job at time t by $T(t)$.

The key piece of our model is the characterization of the completion rate of the job, $\mu(t)$, as a function of the average number of speculative copies per task at time t, $k(t)$. Note that $\mu(t)$ should be interpreted as the completion rate of the ith job. By focusing on the service rate we are ignoring ordering of the tasks and focusing primarily on the impact of speculation.

The statistic characteristics of the task completion time distribution have significanlt impact on the scheduler design. Our analysis of the task durations in the Facebook and Bing traces suggests that task durations have a Pareto tail (i.e., $\mathbb{P}(\tau > x) = \Theta(x^{-\beta})$) with shape parameter $\beta = 1.259$ as shown in the Hill plot in Figure 2.3. A Hill plot provides a more robust estimation of Pareto distributions than the, more commonly used, regression on a log-log plot [58]. To interpret the plot, a flat region corresponds to an estimate of β. The fact that the curve in Figure 2.3 is flat over a large range of order statistics (on the x-axis), but not all order statistics, indicates that the distribution of task sizes is not exactly Pareto distribution in its body, but is well-approximated by a Pareto (power-law) tail. Thus, we assume the task duration τ follows a Pareto distribution with shape parameter β and scale parameter x_m in the following discussion, where we assume the distribution is strongly heavy-tailed, i.e., $1 < \beta \leq 2$.

In our analysis we begin with proactive speculation, and then move to reactive speculation. In proactive speculation, any task is speculated immediately to $k(t)$

[3]For approximation jobs, T should be interpreted as the number of tasks that are completed before the deadline or error limit is reached.

copies upon scheduling, where $k(t)$ is decided by the remaining number of tasks $T(t)$ and task completion time distribution τ. While in the reactive approach, any task is launched for a single copy at the beginning, then is speculated to $k(t)$ copies if necessary. Besides $T(t)$ and τ, $k(t)$ is also decided by the expected completion time calculated approximately by the test run of the first copy. This progression is natural since the analysis of proactive speculation serves as a stepping stone to the design of reactive speculation policies. Further, in the case of proactive speculation, we can precisely specify the optimal policy, whereas in the case of reactive speculation, we must resort to numerical optimization.

2.3.1 Proactive speculation

We start by considering a general class of proactive policies that launch $k(t)$ speculative copies of tasks when the job has remaining size $T(t)$. We propose the following *approximate model* for $\mu(t)$ in this case.

$$\mu(t) = \min\left(S, T(t)k(t)\right) \times \left(\frac{\mathbb{E}[\tau]}{k(t)\mathbb{E}\left[\min(\tau_1, \ldots, \tau_{k(t)})\right]}\right), \tag{2.1}$$

where τ is a random task completion time.

To understand this approximate model, note that the first term approximates the number of slots the job occupied and the second term approximates the "blow up factor," i.e., the ratio of the expected work completed without duplications to the amount of work done with duplications. To approximate the number of slots the job occupied, note that there are $T(t)k(t)$ tasks available to schedule at time t, including speculative copies. Given that the maximum capacity that can be allocated is S, we obtain the first term in (2.1). The second term 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 ($k(t)E[\min(\tau_1, \tau_2, \ldots, \tau_{k(t)})]$), since $k(t)$ copies are created and then they are stopped when the first copy completes. Perhaps the most important aspect of this approximation is the fact that task durations are i.i.d., and this is what leads both to stragglers and to the benefits of replication.

While our focus in this chapter is on scheduling to minimize completion times, the model described above is not well suited toward analytic results about that metric. Instead, our analysis focuses on scheduling to maximize completion rate, in other words, 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.

Given the model in (2.1), the question is: What proactive speculation policy maximizes the job completion rate? As discussed above, the distribution of task sizes shows considerable evidence of a Pareto-tail, and so we focus our analysis on this setting. The following theorem states how the optimal speculation level $k(t)$ behaves.

Theorem 1. *When task duration follows Pareto(x_m,β) distribution with $1 < \beta \leq 2$, the proactive speculation policy that maximizes the completion rate $\mu(t)$ of the job is*

$$k(t) = \begin{cases} \frac{2}{\beta}, & S \leq \frac{2}{\beta}T(t) \\ S/T(t), & S > \frac{2}{\beta}T(t). \end{cases} \tag{2.2}$$

Proof. Under the assumption of Pareto distribution, we have,

$$\begin{aligned} \mathbb{E}[\tau] &= \frac{\beta x_m}{\beta - 1} \\ \mathbb{E}[min(\tau_1, \ldots, \tau_{k(t)}] &= \frac{k\beta x_m}{k\beta - 1}. \end{aligned} \tag{2.3}$$

Substitute (2.3) to (2.1),

$$\mu(t) = \begin{cases} \frac{k(t)\beta - 1}{k(t)^2(\beta - 1)}S, & S \leq T(t)k(t) \\ \frac{k(t)\beta - 1}{k(t)(\beta - 1)}T, & S > T(t)k(t) \end{cases}$$

$$= \begin{cases} -(\frac{1}{k} - \frac{\beta}{2})^2 \frac{S}{\beta - 1} + \frac{\beta^2 S}{4(\beta - 1)}, & S \leq T(t)k(t) \\ (\beta - \frac{1}{k})\frac{T}{\beta - 1}, & S > T(t)k(t). \end{cases}$$

Thus, when $k(t)$ follows (2.2), $\mu(t)$ achieves its maximum. □

This theorem indicates that the optimal speculation strategy should change from conservative to aggressive as the job processes. Specifically the first line corresponds to the "early waves" and the second line corresponds to the "last wave". During the "early waves" the optimal policy speculates conservatively corresponding to the task duration shape parameter β. In contrast, during the "last wave", regardless of the task duration distribution, the optimal policy speculates to ensure all slots are used.

2.3.2 Reactive speculation

We now turn to reactive speculation policies, which launch copies of a task only after it has completed ω work. Both GS and RAS are examples of such policies and can be translated into choices for ω. Proactive speculation is also a special case of the reactive speculation with $\omega = 0$.

Our analysis of proactive policies provides important insights into the design of reactive policies. In particular, during early waves, the the optimal proactive policy runs at most two copies of each task, and so we limit our reactive policies to this level of speculation. Additionally, the previous analysis highlights that during the last wave it is best to speculate aggressively in order to use up the full capacity, and thus it is best to speculate immediately without waiting ω time. This yields the following approximation for $\mu(t)$:

$$\mu(t) = \begin{cases} \frac{E[\tau_1]}{E[\tau_1 | 0 \le \tau_1 < \omega] \Pr(0 \le \tau_1 < \omega) + (2E[Z - \omega | \tau_1 \ge \omega] + \omega) \Pr(\tau_1 > \omega)} S, \\ \quad \text{when } S \le T(t)(\Pr(0 \le \tau_1 < \omega) + 2\Pr(\tau_1 \ge \omega)), \\ \\ \text{optimal proactive speculation (from (2.1)),} \\ \quad \text{when } S > T(t)(\Pr(0 \le \tau_1 < \omega) + 2\Pr(\tau_1 \ge \omega)), \end{cases} \tag{2.4}$$

where τ_1, τ_2 are random task durations and $Z = \min(\tau_1, \tau_2 + \omega)$.

Again, the first line in (2.4) approximates the completion rate during the early waves of the job, while the second line approximates the completion rate during the final wave of the job. To understand the first line, note that during early waves there are enough tasks to spawn over all the available slots S as long as $T(t)(\Pr(0 \le \tau_1 < \omega) + 2\Pr(\tau_1 \ge \omega)) \ge S$. Thus, all that remains is the "blow up factor." As before, the numerator is the expected amount of work per task without

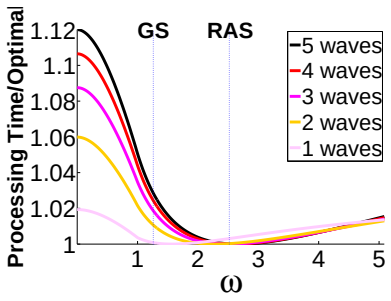


Figure 2.4: Near-optimality of GS & RAS under Pareto task durations ($\beta = 1.259$).

speculation ($E[\tau]$) and the denominator is the expected amount of work per task with reactive speculation. This is $E[\tau|\tau < \omega]$ if the initial copy finishes before ω, and $2E[Z - \omega|\tau_1 > \omega] + \omega$ if the initial copy takes longer than ω.

Within this model, our design problem can now be reduced to finding ω *that minimizes the response time of the job*. GS and RAS both correspond to particular rules for how to choose ω. To see this, we can define $t_{\text{new}} = E[\tau]$ and $t_{\text{rem}} = E[\tau - \omega|\tau > \omega]$, where τ is a random task duration. Then, under GS, ω is the time when $E[\tau] = E[\tau - \omega|\tau > \omega]$, and, under RAS, ω is the time when $2E[\tau] = E[\tau - \omega|\tau > \omega]$. The complicated form of (2.4) makes it difficult to understand the optimal ω analytically, and thus we use numerical calculations.

Figure 2.4 contrasts the performance of all the replication policies in this more general class. Specifically, it shows the ratio of the completion rates of the replication policies with parameter ω normalized with respect to the optimal completion rate. It illustrates this ratio for jobs of differing numbers of waves, and for ω in a wide normalized range. To highlight GS and RAS, they are shown via vertical lines. The completion rates shown in the figure are computed using the model and analysis described above. The main conclusion from this figure is neither GS or RAS is universally optimal, but each is near-optimal for jobs with a certain number of waves: RAS for jobs with large numbers of waves and GS for jobs with small numbers of waves.

2.3.3 Optimal speculation design guidelines

To this point, we have presented a simple analytic model for speculation in approximation jobs. The model assumes wave-based scheduling and constant wave-width for a job. Here we summarize the three major guidelines from our analysis. Most importantly, these guidelines highlight that different speculation policies are required

during the early waves of a job than during the final wave.

Guideline 1. *During the early waves of a job, speculation is only valuable if task durations are extremely heavy tailed, e.g., Pareto with infinite variance (i.e., with shape parameter $\beta < 2$). In this case, it is optimal to speculate conservatively, using ≤ 2 copies of a task.*

This guideline is relevant because task durations are indeed heavy-tailed for the Facebook and Bing traces (see the Hill plot in Figure 2.3), which suggests that task durations have a Pareto tail (i.e., $P(\tau > x) = \theta(x^{-\beta})$) with shape parameter $\beta = 1.259$. While both GS and RAS speculate during early waves, RAS is more conservative than GS and thus outperforms it during early waves.

Guideline 2. *During the final wave of a job, speculate aggressively to fully utilize the allotted capacity.*

This guideline says that, even if all tasks are currently scheduled, if a slot becomes available it should be filled with a speculative copy. While both GS and RAS do this to some extent, GS speculates more aggressively than RAS and thus, outperforms RAS during the final wave.

The previous two guidelines highlight a tradeoff between RAS and GS, which we formalize next.

Guideline 3. *For jobs that require more than two waves RAS is near-optimal, while for jobs that require fewer than two waves GS is near-optimal.*

This guideline is direct conclusion from our analysis in Section 2.3.2 based on numerical optimization shown in Figure 2.4 .

2.4 GRASS Speculation Algorithm

In this section, we build our speculation algorithm, GRASS.[4] Our theoretical analysis summarized in Section 2.3 motivates a design that uses RAS during the early waves of jobs and GS during the final two waves. A simple *strawman* solution to achieve this would be as follows. For deadline-bound jobs, switch from RAS to GS when the time to the deadline is sufficient for at most two waves of tasks. Similarly, for

[4] GRASS is a concatenation of GS and RAS.

error-bound jobs, switch when the number of (unique) scheduled tasks needed to satisfy the error-bound makes up two waves.

Identifying the final two waves of tasks is difficult in practice. Tasks are not scheduled at explicit wave boundaries but rather as and when slots open up. In addition, the wave-width of jobs does not stay constant but varies considerably depending on cluster utilization. Finally, task durations are varied and hard to estimate.

In light of these difficulties, we interpret the guideline as follows: RAS is better when the deadline is loose or the error limit is low, while otherwise GS performs better. This mimics the intuition from the examples in Section 2.2.1. Therefore, GRASS seeks to switch from RAS to GS as it gets *close to the job's approximation bound*.

The complexities in these systems mean that precise estimates of the optimal switching point cannot be obtained from our model. Instead, we adopt an indirect learning based approach where inferences are made based on executions of previous jobs (with similar number of tasks) and cluster characteristics (utilization and estimation accuracy). We compare our learning approach to the strawman described above in Section 2.6.3.

2.4.1 Learning the Switching Point

An ideal approach would accumulate enough samples of job performance (accuracy or completion time) based on switching to GS at different points. For deadline-bound jobs, this is decided by the remaining time to the deadline. For error-bound jobs, this is decided by the number of tasks to complete towards meeting the error. To speed up our sample collection, instead of accumulating samples of switching to GS, we simply generate samples of job performance using GS or RAS *throughout* the job (described shortly in Section 2.4.2).

An incoming job starts with RAS and periodically compares samples of jobs smaller than its size during its execution to check if it is better to switch to GS. It checks by using its remaining work at any point (measured in time remaining or tasks to complete). It steps through all possible points in its remaining work at which it could switch and estimates the optimal point using job samples of appropriate sizes. It continues with RAS until the optimal switching point turns out to be *at present*. The

above calculation for the optimal switching point is performed periodically during the job's execution.

For example, when a deadline-bound job has 6s of its deadline remaining, GRASS compares the potential accuracy obtained if it were to switch at each point in its future (at 1s granularity). The accuracy if it were to switch after, say, 2s is the sum of accuracies of jobs with deadlines of 2s that used only RAS and those with 4s that used only GS. Switching happens if among all such points, the best accuracy is obtained by switching now.

The size of the job alone is insufficient to calculate the optimal switching point. Even jobs of comparable size might have different numbers of waves depending on the number of available slots. Therefore, we augment our samples of job performance with the number of waves, simply approximated using current *cluster utilization*.

Finally, *estimation accuracy* of t_{rem} and t_{new} also decides the optimal switching point. RAS's cautious approach of considering the opportunity cost of speculating a task is valuable when task estimates are erroneous. In fact, at low estimation accuracies (along with certain values of utilization and deadline/error-bound), it is better to not switch to GS at all and to employ RAS all along. Section 2.6.3.2 analyzes the impact of these three factors.

Therefore, GRASS obtains samples of job performance with both GS and RAS across values of deadline/error-bound, estimation accuracy of t_{rem} and t_{new}, and cluster utilization. It uses these three factors collectively to decide when (and if) to switch from RAS to GS. We next describe how the samples are collected.

2.4.2 Generating Samples

As described above, GRASS compares samples of job performance that use only GS or RAS throughout, to decide when to switch. These samples have to be updated continuously to stay abreast with dynamic changes in clusters. To continuously generate such samples, we introduce a *perturbation* in GRASS's switching decision. With a small probability ξ, GRASS decides to *not* switch and instead picks one of GS or RAS for the entire duration of the job (both GS and RAS are equally probable). Such perturbation helps us obtain comparable samples.

The crucial trade-off in setting ξ is in balancing the benefit of obtaining such comparable samples with the performance loss incurred by the job due to not making the right switching decision. Theoretical analyses of such multi-armed bandit problems

in prior work defines an optimal value of ξ by making stochastic assumptions about the distribution of the costs and the associated rewards [59]. Our setup, however, does not yield itself to such assumptions as the underlying distribution can be arbitrary. Another class of techniques that we considered modified ξ with time [60]. Over time, the value of ξ is gradually reduced using a damping function, thus indicating higher confidence in the learned value. We decided against such damping of ξ because clusters constantly evolve with new software and hardware modules, leading to newer interactions between them.

Therefore, we pick a constant value of ξ using empirical analysis. A job is marked for generating performance samples with a probability of ξ, and we pick GS or RAS with equal probability. In practice, we bucket jobs by their number of tasks and compare only within jobs of the same bucket.

2.5 Implementation

We implement GRASS on top of two data-analytics frameworks, Hadoop (version 0.20.2) [4] and Spark (version 0.7.3) [52], representing batch jobs and interactive jobs, respectively. Hadoop jobs read data from HDFS while Spark jobs read from in-memory RDDs. Consequently, Spark tasks finished quicker than Hadoop tasks, even with the same input size. Note that while Hadoop and Spark use LATE[18] currently, we also implement Mantri[17] to use as a second baseline.

Implementing GRASS required two changes: task executors and job scheduler. Task executors were augmented to periodically report progress. We piggyback on existing update mechanisms of tasks that conveyed only their start and finish. Progress reports were configured to be sent every 5% of data read/written. The job scheduler collects these reports, maintains samples of completed tasks and jobs, and decides the switching point.

2.5.1 Task Estimators

GRASS uses two estimates for tasks: remaining duration of a running task (t_{rem}) and duration of a new copy (t_{new}).

Estimating t_{rem}: Tasks periodically update the scheduler with *progress reports* containing the fraction of input read and output written. Since tasks of analytics jobs are IO-intensive, we extrapolate the remaining duration of the task based on the time elapsed thus far.

Estimating t_{new}: We estimate the duration of a new task by sampling from durations of completed tasks (normalized to input and output sizes). The t_{new} values of all tasks are updated whenever a task completes.

Accuracy of estimation: While the above techniques are simple, the downside is the error in estimation. Our estimates of t_{rem} and t_{new} achieve moderate accuracies of 72% and 76%, respectively, on average. When a task completes, we update the accuracy using the estimated and actual durations. GRASS uses the accuracy of estimation to appropriately switch from RAS to GS.

2.5.2 DAG of Tasks

Jobs are typically composed as a DAG of tasks with *input* tasks (*e.g.,* map or extract) reading data from the underlying storage and *intermediate* tasks (*e.g.,* reduce or join) aggregating their outputs. Even in DAGs of tasks, the accuracy of the result is decided by the fraction of completed input tasks. This makes GRASS's functioning straightforward in error-bound jobs—complete as many input tasks as required to meet the error-bound and all intermediate tasks further in the DAG.

For deadline-bound jobs, we use a widely occurring property that intermediate tasks perform similar functions across jobs. Further, they have relatively fewer stragglers. Thus, we estimate the time taken for intermediate tasks by comparing jobs of similar sizes and then subtract it to obtain the deadline for the input tasks.

Input tasks of a job, typically, read equal amounts of data. Thus, the fraction of tasks completed represents fraction of data processed too, thus making it a good indicator of the result's accuracy.

2.6 Evaluation

We evaluate GRASS on a 200 node EC2 cluster. Our focus is on quantifying the performance improvements compared to current designs, i.e., LATE [18] and Mantri [17], and on understanding how close to the optimal performance GRASS comes. Our main results can be summarized as follows.

1. GRASS increases accuracy of deadline-bound jobs by 47% and speeds up error-bound jobs by 38%. Even non-approximation jobs (i.e., error-bound of zero) speed up by 34%. Further, GRASS nearly matches the optimal performance (Section 2.6.2).

	Facebook	Microsoft Bing
Dates	Oct 2012	May-Dec 2011
Framework	Hadoop	Dryad
Script	Hive [11]	Scope [24]
Jobs	575K	500K
Cluster Size	3,500	Thousands
Straggler–mitigation	LATE [18]	Mantri [17]

Table 2.1: Details of Facebook and Bing traces.

2. GRASS's learning based approach for determining when to switch from RAS to GS is over 30% better than simple strawman techniques. Further, the use of all three factors discussed in Section 2.4.1 is crucial for inferring the optimal switching point. (Section 2.6.3)

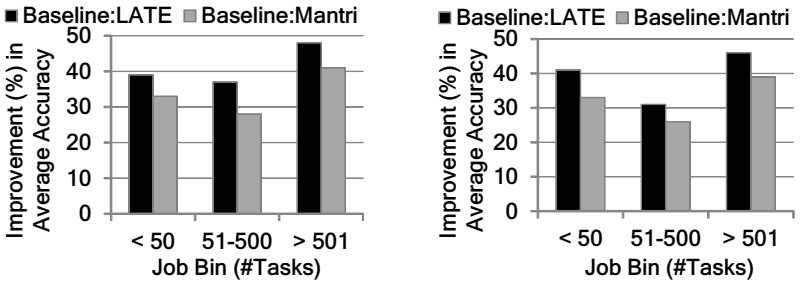
2.6.1 Methodology

Workload: Our evaluation is based on traces from Facebook's production Hadoop [4] cluster and Microsoft Bing's production Dryad [61] cluster. The traces capture over half a million jobs running across many months (Table 2.1). The clusters run a mix of interactive and production jobs whose performance have significant impact on productivity and revenue. The jobs had diverse resource requirements of CPU, memory and IO. To create our experimental workload, we retain the inter-arrival times, input files and number of tasks of jobs. The jobs were, however, not approximation queries and required all their tasks to complete. Hence, we convert the jobs to mimic deadline- and error-bound jobs as follows.

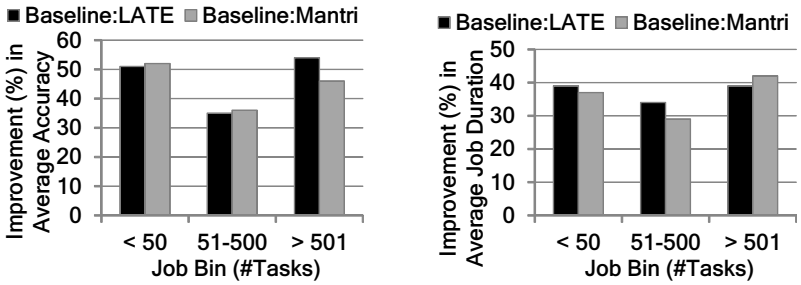
For experiments on error-bound jobs, we pick the error tolerance of the job randomly between 5% and 30%. This is consistent with the experimental setup in recently reported research [21, 62]. Prior work also recommends setting deadlines by calibrating task durations [21, 47]. For the purpose of calibration, we obtain the ideal duration of a job in the trace by substituting the duration of each of its task by the median task duration in the job, again, as per recent work on straggler mitigation [15]. We set the deadline to be an additional factor (randomly between 2% to 20%) on top of this ideal duration.

Job Bins: We bin jobs by their number of tasks. We use three distinctions "small" (< 50 tasks), "medium" ($51 - 500$ tasks), and "large" (> 500 tasks).

EC2 Deployment: We deploy our Hadoop and Spark prototypes on a 200-node EC2 cluster and evaluate them using the workloads described above. Each experiment



(a) FB Workload–Hadoop (b) Bing Workload–Hadoop



(c) FB Workload–Spark (d) Bing Workload–Spark

Figure 2.5: Accuracy Improvement in deadline-bound jobs with LATE [18] and Mantri [17] as baselines.

is repeated five times and we pick the median. We measure improvement in the average accuracy for deadline-bound jobs and average duration for error-bound jobs.

We also use a trace-driven simulator to evaluate at larger scales and over longer durations. The simulator replays all the task properties including their straggling.

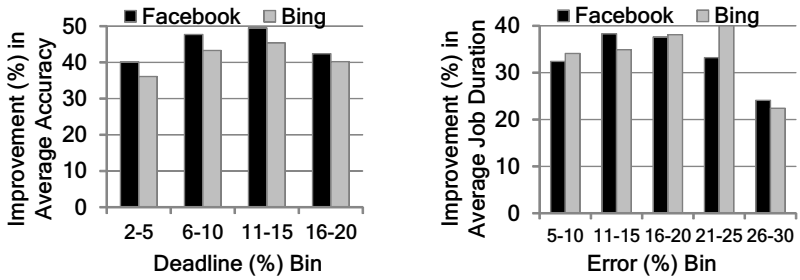
Baseline: We contrast GRASS with two state-of-the-art speculation algorithms—LATE [18] and Mantri [17].

2.6.2 Improvements from GRASS

We contrast GRASS's performance with that of LATE [18], Mantri [17], and the optimal scheduler.

2.6.2.1 Deadline-bound jobs

GRASS improves the accuracy of deadline-bound jobs by 34% to 40% in the Hadoop prototype. Gains in both the Facebook and Bing workloads are similar. Figure 2.5a and 2.5b split the gains by job size. The gains compared to LATE as baseline are consistently higher than Mantri. Also, the gains in large jobs are pronounced



(a) Deadline Bins (b) Error Bins

Figure 2.6: GRASS's overall gains (compared to LATE) binned by the deadline and error bound. Deadlines are binned by the factor over ideal job duration (see Section 2.6.1)

compared to small and medium sized jobs because their many waves of tasks provide plenty of potential for GRASS.

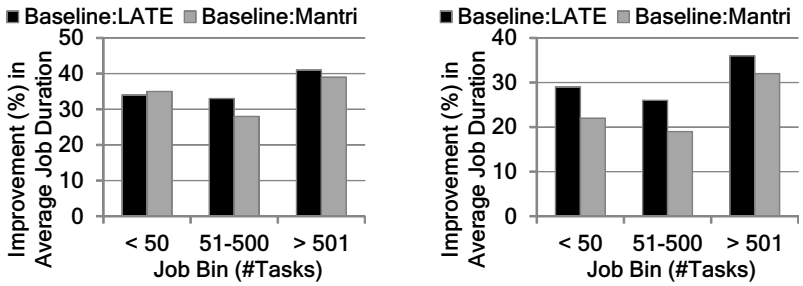
The Spark prototype improves accuracy by 43% to 47%. The gains are higher because Spark's task sizes are much smaller than Hadoop's due to in-memory inputs. This makes the effect of stragglers more distinct. Again, large jobs gain the most, improving by over 50% (Figure 2.5c and 2.5d). Large multi-waved jobs improving more is encouraging because smaller task sizes in future [55] will ensure that multi-waved executions will be the norm. Unlike the Hadoop case, gains compared to both LATE and Mantri are similar because both have only limited effect when the impact of stragglers is high.

Figure 2.6a dices the improvements by the deadline (specifically, the additional factor over the ideal job duration (see Section 2.6.1)). Note that gains are nearly uniform across deadline values. This indicates that GRASS can not only cope with stringent deadlines but be valuable even when the deadline is lenient.

Gains with simulations are consistent with deployment, indicating not only that GRASS's gains hold over longer durations but also the simulator's robustness.

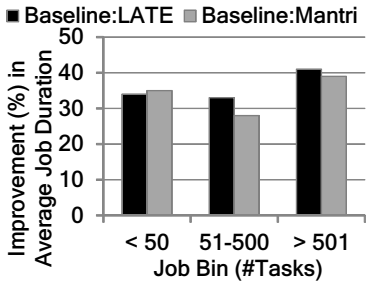
2.6.2.2 Error-bound jobs

Similar to deadline-bound jobs, improvements with the Spark prototype (33% to 37%) are higher compared to the Hadoop prototype (24% to 30%). This shows that GRASS works well not only with established frameworks like Hadoop but also upcoming ones like Spark.

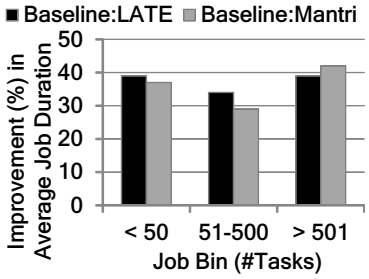


(a) FB Workload–Hadoop

(b) Bing Workload–Hadoop



(c) FB Workload–Spark



(d) Bing Workload–Spark

Figure 2.7: Speedup in error-bound jobs with LATE [18] and Mantri [17] as baselines.

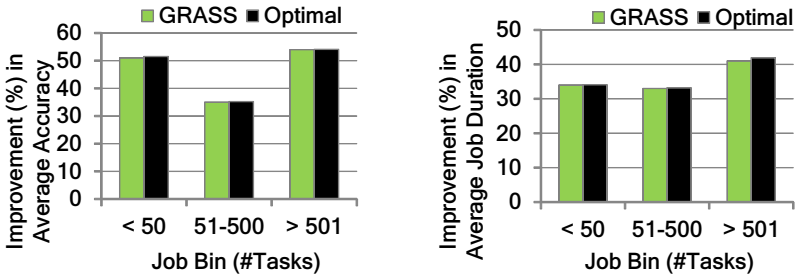
Note that the gains are indistinguishable among different job bins (Figures 2.7a and 2.7b) in the Spark prototype; large jobs gain a touch more in the Hadoop prototype (Figures 2.7c and 2.7d). Again, our simulation results are consistent with deployment, and so are omitted.

As Figure 2.6b shows, GRASS's gains persist at both tight as well as moderate error bounds. At high error bounds, there is smaller scope for GRASS beyond LATE. The gains at tight error bounds are noteworthy because these jobs are closer to *exact* jobs that require all (or most of) their tasks to complete. In fact, exact jobs speed up by 34%, thus making GRASS valuable even in clusters that are yet to deploy approximation analytics.

2.6.2.3 Optimality of GRASS

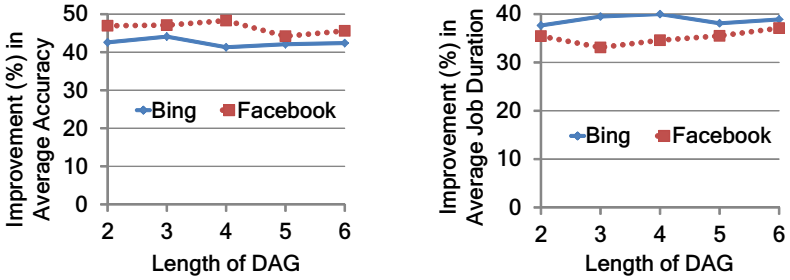
While the results above show the speed up GRASS provides, the question remains as to whether further improvements are possible. To understand the room available for improvement beyond GRASS, we compare its performance with an optimal scheduler that knows task durations and slot availabilities in advance.

Figure 2.8 shows the results for the Facebook workload with Spark. It highlights that



(a) Deadline-bound Jobs (b) Error-bound Jobs

Figure 2.8: GRASS's gains matches the optimal scheduler.



(a) Deadline-bound Jobs. (b) Error-bound Jobs.

Figure 2.9: GRASS's gains hold across job DAG sizes.

GRASS's performance matches the optimal for both deadline as well as error-bound jobs. Thus, GRASS is an efficient near-optimal solution for the NP-hard problem of scheduling tasks for approximation jobs with speculative copies.

2.6.2.4 DAG of tasks

To complete the evaluation of GRASS we investigate how performance gains depend on the length of the job's DAG. Intuitively, as long as our estimation of intermediate phases is accurate, GRASS's handling of the input phase should remain unchanged, and Figure 2.9 confirms this for both deadline and error-bound jobs. Gains from GRASS remain stable with the length of the DAG.

2.6.3 Evaluating GRASS's Design Decisions

To understand the impact of the design decisions made in GRASS, we focus on three questions. First, how important is it that GRASS switches from RAS to GS? Second, how important is it that this switching is learned adaptively rather than fixed statically? Third, how sensitive is GRASS to the perturbation factor ξ? In the

(a) Hadoop

(b) Spark

Figure 2.10: GRASS's switching is 25% better than using GS or RAS all through for deadline-bound jobs. We use the Facebook workload and LATE as baseline.

(a) Facebook Workload–Hadoop

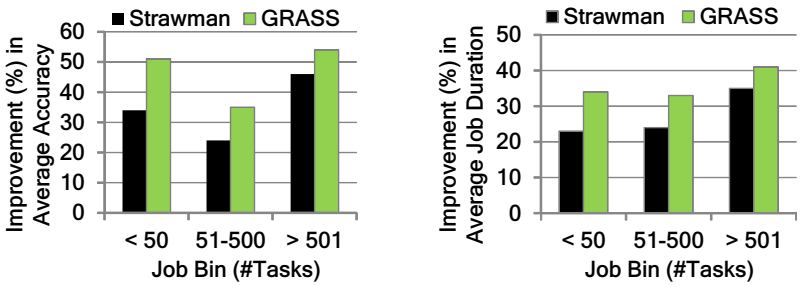
(b) Facebook Workload–Spark

Figure 2.11: GRASS's switching is 20% better than using GS or RAS all through for error-bound jobs. We use the Facebook workload and LATE as baseline.

interest of space, we present results on these topics for only the Facebook workload using LATE as a baseline; results for the Bing workload with Mantri are similar.

2.6.3.1 The value of switching

To understand the importance of switching between RAS and GS we compare GRASS's performance with using only GS and RAS all through the job. Figure 2.10 performs the comparison for deadline-bound jobs. GRASS's improvements, both on average as well as in individual job bins, are strictly better than GS and RAS. This shows that if using only one of them is the best choice, GRASS automatically avoids switching. Further, GRASS's overall improvement in accuracy is over 20% better than the best of GS or RAS, demonstrating the value of switching as the job nears its deadline. The above trends are consistent with error-bound jobs as well (Figure 2.11), though GRASS's benefit is slightly lower.



(a) Deadline-bound Jobs (b) Error-bound Jobs

Figure 2.12: Comparing GRASS's learning based switching approach to a strawman that approximates two waves of tasks. GRASS is 30%−40% better than the strawman.

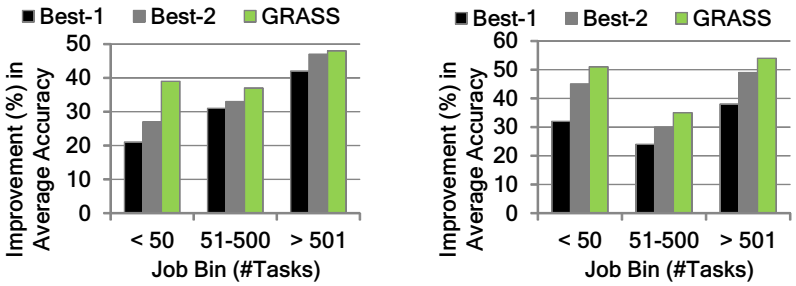
The contrast of GS and RAS is also interesting. GS outperforms RAS for small jobs but loses out as job sizes increase. The significant degradation in performance in the unfavorable job bin (medium and large jobs for GS, versus small jobs for RAS) illustrates the pitfalls of statically picking the speculation algorithm.

2.6.3.2 The value of learning

Given the benefit of switching, the question becomes when this switching should occur. GRASS does this adaptively based on three factors: deadline/error-bound, cluster utilization and estimation accuracy of t_{rem} and t_{new}. Now, we illustrate the benefit of this approach compared to simpler options, i.e., choosing the switching point statically or based on a subset of these three factors. Note that we have already seen that these three factors are enough to be near optimal (Figure 2.8).

Static switching: First, when considering a static design, a natural "strawman" based on our theoretical analysis is to estimate the point when there are two remaining waves as follows. For deadline-bound jobs, it is the point when the time to the deadline is sufficient for at most two waves of tasks. For error-bound jobs, it is when the number of (unique) scheduled tasks sufficient to satisfy the error-bound makes up two waves. The strawman uses the current wave-width of the job and assumes task durations to be the median of completed tasks.

Figure 2.12 compares GRASS with the above strawman. Gains with the strawman are 66% and 73% of the gains with GRASS for deadline-bound and error-bound jobs, respectively. Small and medium jobs lag the most as wrong estimation of switching point affects a large fraction of their tasks. Thus, the benefit of adaptively determining the switching point is significant.



(a) Hadoop (b) Spark

Figure 2.13: Using all three factors for deadline-bound jobs compared to only one or two is 18% – 30% better.

Adaptive switching: Next, we study the impact of the three factors used to adaptively learn the switching threshold. To do this, Figures 2.13 and 2.14 compare the designs using the best one or two factors with GRASS.

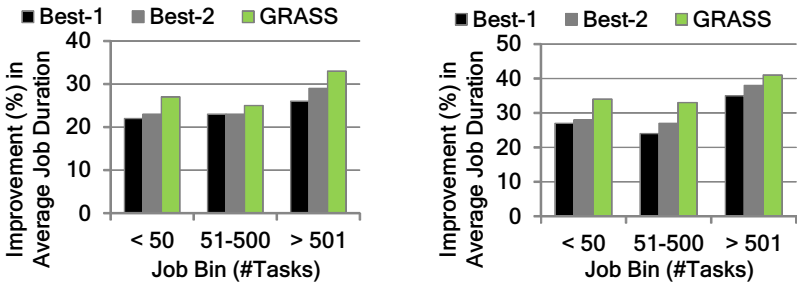
When only one factor can be used to switch, picking the deadline/error-bound provides the best results. This is intuitive given the importance of the approximation bound to the ordering of tasks. When two factors are used, in addition to the deadline/error-bound, cluster utilization matters more for the Hadoop prototype while estimation accuracy is important for the Spark prototype. Tasks of Hadoop jobs are longer and more sensitive to slot allocations, which is dictated by the utilization. While the smaller Spark tasks are more fungible, this also makes them sensitive to estimation errors.

Using only one factor is significantly worse than using all three factors. The performance picks up with deadline-bound jobs when two factors are used, but error-bound jobs' gains continue to lag until all three are used. Thus, in the absence of a detailed model for job executions, the three factors act as good predictors.

2.6.3.3 Sensitivity to Perturbation

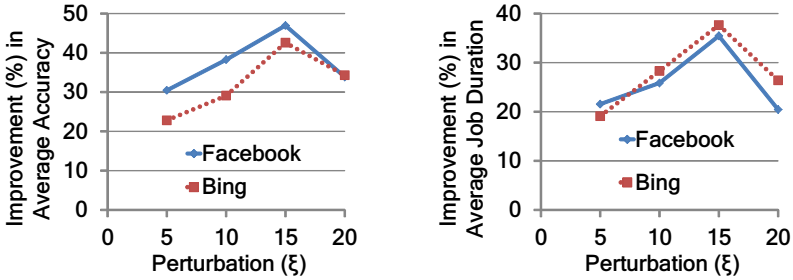
The final aspect of GRASS that we evaluate is the perturbation factor, ξ, which decides how often the scheduler does *not* switch during a job's execution (described in Section 2.4.2). This perturbation is crucial for GRASS's learning of the optimal switching point. All results shown previously set ξ to 15%, which was picked empirically.

Figure 2.15 highlights the sensitivity of GRASS to this choice. Low values of ξ hamper learning because of the lack of sufficient samples, while high values incur



(a) Hadoop (b) Spark

Figure 2.14: Using all three factors for error-bound jobs compared to one or two factors is 15% – 25% better.



(a) Deadline-bound Jobs (b) Error-bound Jobs

Figure 2.15: Sensitivity of GRASS's performance to the perturbation factor ξ. Using $\xi = 15\%$ is empirically best.

performance loss resulting from not switching from RAS to GS often enough. Our results show, that this exploration–exploitation tradeoff is optimized at $\xi = 15\%$, and that performance drops off sharply around this point. Deadline-bound jobs are more sensitive to poor choice of ξ than error-bound jobs. Using ξ of 15% is consistent with studies on multi-armed bandit problems [63], which is related to our learning problem.

2.7 Related Work

The problem of stragglers was identified in the original MapReduce paper [1]. Since then solutions have been proposed to mitigate them using speculative executions [17, 18, 61]. These solutions, however, are not for approximation jobs. These jobs require proritizing the right subset of tasks by carefully considering the opportunity cost of speculation. Further, our evaluations show that GRASS speeds up even for exact jobs that require all their tasks to complete. Thus, it is a unified solution that

cluster schedulers can deploy for both approximation as well as non-approximation computations.

Prioritizing tasks of a job is a classic scheduling problem with known heuristics [49, 50]. These heuristics assume accurate knowledge of task durations and hence do not require speculative copies to be scheduled dynamically. Estimating task durations accurately, however, is still an open challenge as acknowledged by many studies [15, 57]. This makes speculative copies crucial and we develop a theoretically backed solution to optimally prioritize tasks with speculative copies.

Modeling real world clusters has been a challenge faced by other schedulers too. Recently reported research has acknowledged the problem of estimating task durations [53], predicting stragglers [15, 57], and modeling multi-waved job executions [54]. Their solutions primarily involve sidestepping the problem by not predicting stragglers and replicating the tasks up front [15], or by approximating number of waves to file sizes [54]. Such sidestepping, however, is not an option for GRASS and hence we build tailored approximations.

Finally, replicating tasks in distributed systems have a long history [64–66] with extensive studies in prior work [67–69]. These studies assume replication *up front* as opposed to *dynamic* replication in reaction to stragglers. The latter problem is both harder and unsolved. In this work, we take a stab at this problem that yields near-optimal results in our production workloads.

2.8 Concluding Remarks

This chapter explores speculative task scheduling in the context of approximation jobs. From the analysis of a generic analytic model, we develop a speculation algorithm, GRASS, that uses opportunity cost to determine when to speculate early in the job and then switches to more aggressive speculation as the job nears its approximation bound. Prototypes on Hadoop and Spark, deployed on a 200 node EC2 cluster, shows that GRASS improves accuracy for deadline-bound jobs by 47% and speeds up error-bound jobs by 38%, in production workloads from Facebook and Bing. Further, the evaluation highlights that GRASS is a unified speculation solution for both approximation and exact computations, since it also provides a 34% speed up for exact jobs.

CHAPTER 3

SPECULATION-AWARE CLUSTER SCHEDULING ON THE JOB LEVEL

Data analytics frameworks have successfully realized the promise of "scaling out" by automatically composing user-submitted scripts into *jobs* of many parallel *tasks* and executing them on large clusters. However, as clusters increase in size and complexity, providing *scalable* and *predictable* performance is an important ongoing challenge for interactive analytics frameworks [70, 71]. Indeed, production clusters at Google and Microsoft [16, 72] acknowledge this as a prominent goal.

As the scale and complexity of clusters increase, hard-to-model systemic interactions that degrade the performance of tasks become common [15, 16]. Consequently, many tasks become "stragglers", i.e., running slower than expected, leading to significant *unpredictability* (and delay) in job completion times – tasks in Facebook's Hadoop cluster can run up to 8× slower than expected [15]. The most successful and widely deployed straggler mitigation solution is *speculation*, i.e., speculatively running extra copies of tasks that have become stragglers (or likely to), and then picking the earliest copy that finishes, e.g., [1, 15, 17–19]. Speculation is commonplace in production clusters, e.g., in our analysis of Facebook's Hadoop cluster, speculative tasks account for 25% of all tasks and 21% of resource usage.

We studied task-level speculation scheduling design in Chapter 2. In this chapter, we extend from task level to job level. Speculation is intrinsically intertwined with job scheduling because spawning a speculative copy of a task has a direct impact on the resources available for *other* jobs. Aggressive speculation can improve the performance of the job at hand but hurt the performance of other jobs. Despite this, speculation policies deployed today are all designed and operated *independently* of job scheduling; schedulers simply allocate slots to speculative copies in a "best-effort" fashion, e.g., [1, 17, 19, 20].

Coordinating speculation and scheduling decisions is an opportunity for significant performance improvement. However, achieving such coordination is challenging,

particularly as schedulers themselves *scale* out. Schedulers are increasingly becoming *decentralized* in order to scale to hundreds of thousands of machines with each machine equipped with tens of compute slots for tasks. This helps them make millions of scheduling decisions per second, a requirement about two orders of magnitude beyond the (already highly-optimized) centralized schedulers, e.g., [9, 52, 73]. In decentralized designs multiple schedulers operate *autonomously*, with each of them scheduling only a subset of the jobs, e.g., [16, 20, 24]. Thus, the coordination between speculation and scheduling must be achieved without maintaining central information about all the jobs.

In this chapter we present the design of the *first speculation-aware job scheduler*, Hopper, which dynamically allocates slots to jobs keeping in mind the speculation requirements necessary for *predictable* performance. Hopper incorporates a variety of factors such as data locality, estimates of task execution times, fairness, dependencies (DAGs) between tasks, etc. Further, Hopper is compatible with all current speculation algorithms and can operate as either a centralized or decentralized scheduler; achieving *scalability* by not requiring any central state.

The key insight behind Hopper is that a scheduler must anticipate the speculation requirements of jobs and dynamically allocate capacity depending on the *marginal* value (in terms of performance) of extra slots which are likely used for speculation. A novel observation that leads to the design of Hopper is that there is a sharp "threshold" in the marginal value of extra slots – an extra slot is always more beneficial for a job below its threshold than it is for any job above its threshold. The identification of such a threshold then allows Hopper to use different resource allocation strategies depending on whether the system capacity is such that all jobs can be allocated more slots than their threshold or not. This leads to a dynamic, adaptive, online scheduler that reacts to the current system load in a manner that appropriately weighs the value of speculation.

Importantly, the core components of Hopper can be *decentralized* effectively. The key challenge to avoiding the need to maintain a central state is the fact that stragglers create heavy-tailed task durations, e.g., see [15, 19, 74]. Hopper handles this by adopting a "power of many choices" viewpoint to approximate the global state, which is fundamentally more suited than the traditional "power of two choices" viewpoint due to the durations and frequency of stragglers.

To demonstrate the potential of Hopper, we have built three demonstration prototypes by augmenting the *centralized* scheduling frameworks Hadoop [4] (for batch jobs)

and Spark [52] (for interactive jobs), and the *decentralized* framework Sparrow [20]. Hopper incorporates many practical features of jobs into its scheduling. Among others, it estimates the amount of *intermediate* data produced by the job and accounts for their pipelining between phases, integrates *data locality* requirements of tasks, and provides *fairness* guarantees.

We have evaluated our three prototypes on a 200 node private cluster using workloads derived from Facebook's and Microsoft Bing's production traces. The decentralized and centralized implementations of Hopper reduce the average job completion time by up to 66% and 50% compared to state-of-the-art scheduling and straggler mitigation techniques. The gains are consistent across common speculation algorithms (LATE [18], GRASS [19], and Mantri [17]), DAGs of tasks, and locality constraints, while providing fine-grained control on fairness. Importantly, the gains *do not* result from improving the speculation mechanisms but from improved *coordination* of scheduling and speculation decisions.

3.1 Background & Related Work

We begin by presenting a brief overview of existing cluster schedulers: how they allocate resources across jobs, both centralized and decentralized (Section 3.1.1), and how they handle straggling tasks (Section 3.1.2). This overview highlights the lack of coordination that currently exists between scheduling and straggler mitigation strategies such as speculation.

3.1.1 Cluster Schedulers

Job scheduling – allotting compute slots to jobs for their tasks – is a classic topic with a large body of work.

The most widely-used scheduling approach in clusters today is based on *fairness* which, without loss of generality, can be defined as equal sharing (or weighted sharing) of the available resources among jobs (or their users) [25, 75–78]. Fairness, of course, comes with performance inefficiencies, e.g., [79, 80].

In contrast, the performance-optimal approach for job scheduling is *Shortest Remaining Processing Time (SRPT)*, which assigns slots to jobs in ascending order of their remaining duration (or, for simplicity, the remaining number of tasks). SRPT's optimality in both single [81] and multi-server [82] settings motivates a focus on prioritizing small jobs and has led to many schedulers such as [83–85].

The schedulers mentioned above are all centralized; however, motivated by scalability, many clusters are beginning to adopt *decentralized* schedulers, e.g., at Google [16], Apollo [72] at Microsoft, and the recently proposed Sparrow [20] scheduler. The scalability of decentralized designs allows schedulers to cope with growing cluster sizes and increasing parallelism of jobs (due to smaller tasks [55]), allowing them to scale to millions of scheduling decisions (for tasks) per second.

Importantly, the literature on cluster scheduling (both centralized and decentralized) ignores an important aspect of clusters: straggler mitigation via speculation. No current schedulers coordinate decisions with speculation mechanisms, while our analysis shows that speculative copies account for a sizeable fraction of all tasks in production clusters, e.g., in Facebook's Hadoop cluster, speculative tasks account for 25% of all tasks and 21% of resource usage.

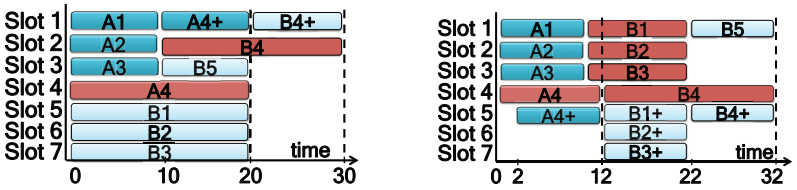
3.1.2 Straggler Mitigation via Speculation

Dealing with *straggler* tasks, i.e., tasks that take significantly longer than expected to complete, is an important challenge for cluster schedulers, one that was called out in the original MapReduce paper [1], and a topic of significant subsequent research [15, 17–19].

Clusters already blacklist *problematic* machines (e.g., faulty disks or memory errors) and avoid scheduling tasks on them. However, despite blacklisting, stragglers occur frequently, often due to intrinsically complex causes such as IO contention, interference by periodic maintenance operations, and hardware behaviors which are hard to model and circumvent [15, 57, 86]. Straggler prevention based on comprehensive root-cause analyses is an open research challenge.

The most effective, and indeed the most widely deployed, technique has been *speculative* execution. Speculation techniques monitor the progress of running tasks, compare them to the progress of completed tasks of the job, and spawn speculative copies for those progressing much slower, i.e., straggling. It is then a race between the original and speculative copies of the task and on completion of one, the other copies are killed.[1]

[1] Schedulers avoid checkpointing a straggling task's current output and spawning a new copy for just the remaining work due to the overheads and complexity of doing so. In general, even though the speculative copy is spawned on the expectation that it would be faster than the original, it is extremely hard to guarantee that in practice. Thus, both are allowed to run until the first completes.



(a) Best-effort Speculation. (b) Budgeted Speculation

Figure 3.1: Combining SRPT scheduling and speculation for two jobs A (4 tasks) and B (5 tasks) on a 7-slot cluster. The + suffix indicates speculation. Copies of tasks that are killed are colored red.

There is considerable (statistical and systemic) sophistication in speculation techniques, e.g., ensuring early detection of stragglers [17], predicting duration of new (and running) tasks [87], and picking lightly loaded machines to spawn speculative copies [18]. The techniques also take care to avoid speculation when a new copy is unlikely to benefit, e.g., when the single input source's machine is the cause behind the straggling [88].

Speculation has been highly effective in mitigating stragglers, bringing the ratio of the progress rates of the median task of a job to its slowest down from 8× (and 7×) to 1.08× (and 1.1×) in Facebook's production Hadoop cluster (and Bing's Dryad cluster).

Speculation has, to this point, been done independently of job scheduling. This is despite the fact that when a speculative task is scheduled it takes resources away from other jobs; thus there is an intrinsic tradeoff between scheduling speculative copies and scheduling new jobs. In this chapter, we show that integrating these two via speculation-aware job scheduling can speed up jobs considerably, even on average. Note that these gains are *not* due to improving the speculative execution techniques, but instead come purely from the integration of speculation and job scheduling decisions.

3.2 Motivation

The previous section highlights that speculation and scheduling are currently designed and operated independently. Here, we illustrate the value of coordinated speculation and scheduling using simple examples.

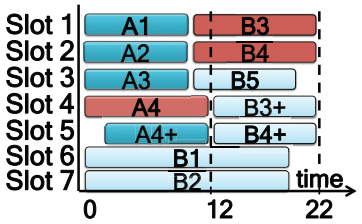


Figure 3.2: Hopper: Completion time for jobs A and B are 12 and 22. The + suffix indicates speculation.

A	A1	A2	A3	A4
t_{orig}	10	10	10	30
t_{new}	10	10	10	10

B	B1	B2	B3	B4	B5
t_{orig}	20	20	20	40	10
t_{new}	10	10	10	10	10

Table 3.1: Task sets. t_{orig} and t_{new} are durations of the original and speculative copies of each task.

3.2.1 Strawman Approaches

We first explore two baselines that characterize how scheduling and speculation interact today. In our examples we assume that stragglers can be detected after a task has run for 2 time units and that, at this point, a speculation is performed if the remaining running time (t_{rem}) is longer than the time to run a new copy (t_{new}). When the fastest copy of a task finishes, other running copies of the same task are killed. Note that while these examples have all jobs arrive at time 0, Hopper is designed to work in an online setting.

Best-Effort Speculation: A simple approach, which is also the most common in practice, is to treat speculative tasks the same as regular tasks. The job scheduler allocates resources for speculative tasks in a "best effort" manner, i.e., *whenever there is an open slot*.

Consider the example in Figure 3.1a with two jobs A (4 tasks) and B (5 tasks) that are scheduled using the SRPT policy. The scheduler has to wait until time 10 to find an open slot for the speculative copy of A4, despite detecting it was straggling at time 2.[2] Clearly, the scheduler can do better. If it had allocated a slot to A's speculative task at time 2 (instead of letting *B* use it), then job A's completion time

[2]At time 10, when A1 finishes, the job scheduler allocates the slot to job A because its remaining processing is smaller than job B's. Job A speculates task A4 because A4's $t_{rem} = t_{orig} - \text{currentTime} = 30 - 10 = 20 > t_{new} = 10$ (see Table 3.1).

would have reduced, without slowing job B (see Table 3.1 for task durations).

Note that similar inefficiencies occur under Fair scheduling in this example.

Budgeted Speculation: The main problem for best-effort speculation is a lack of available slots for speculation when needed. Thus, an alternative approach is to have the job scheduler *reserve* a fixed "budget" of slots for speculative tasks. Budgeting the right size of the resource pool for speculation, however, is challenging because of time-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.

Figure 3.1b illustrates budgeted speculation with three slots (slot 5−7) being reserved for speculation. This, unfortunately, leads to slots 6 and 7 lying fallow from time 0 to 12. If the wasted slot had been used to run a new task, say B1, then job B's completion time would have been reduced. It is easy to see that similar wastage of slots occurs with the Fair scheduler. Note that reserving one or two instead of three slots will not solve the problem, since three speculative copies are required to run simultaneously at a later time.

3.2.2 Challenges in Coordination

In contrast to the two baselines discussed above, Figure 3.2 shows the benefit of coordinated decision making.

At time 0 – 10, we allocate 1 extra slot to job A (for a total of 5 slots), thus allowing it to speculate task A4 promptly. After time 10, we can *dynamically reallocate* the slots to job B. This reduces the average completion time compared to both the budgeted and best-effort strategies. The joint design budgeted slot 5 until time 2 but after task A4 finished, it used all the slots.

Doing such dynamic allocation is already challenging in a centralized environment, and it becomes more so in a decentralized setting. In particular, decentralized speculation-aware scheduling has additional constraints. Since the schedulers are autonomous, there is no central state and thus, no scheduler has complete information about all the jobs in the cluster. Further, every scheduler has information about only a subset of the cluster (the machines it probed). Since decentralization is mainly critical for interactive jobs (sub-second or a few seconds), time-consuming gossiping between schedulers is infeasible. Finally, running *all* the schedulers on one multi-

core machine cramps that machine and caps scalability, the original drawback they aim to alleviate.

In the above example, this means making the allocation as in Figure 3.2 when jobs A and B autonomously schedule their tasks without complete knowledge of utilizations of the slots or even each other's existence.

Thus, the challenges for speculation-aware job scheduling are: (i) *dynamically* allocating/budgeting slots for speculation based on the distribution of stragglers and cluster utilization while being (approximately) *fair* and, in decentralized settings, (ii) using *incomplete information* about the machines and jobs in the cluster. In order to handle those challenges, we have developed an analytical model for speculation-aware job scheduling. The model is based on the model of speculation for approximation jobs as described in Section 2.3. We present the model and its analysis in Section 3.3.

3.3 Modeling and Analyzing Speculation

In this section we introduce the model and analysis which lead to the design of our scheduler Hopper.

Our model is based on the completion rate model described in Chapter 2. Unlike the previous model which only focuses on a single job, this model focuses on a system with multiple jobs arriving over time. We assume the system has S slots, each of which can have one task scheduled to it. Jobs arrive over time and the ith arrival as 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 completion 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 $k_i(t)$.

The key piece of our model is the characterization of the completion 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, $k_i(t)$. Though most task-level speculation algorithms are reactive (i.e., spawn speculative copies for a task only after observing its performance for a short duration) rather than proactive (i.e., spawn speculative copies immediately after the first copy is launched in the system), for ease of the analysis and given the observing time is relatively small compared to the task duration, we adopt the proactive model for the completion rate $\mu_i(t)$ (described in Chapter 2) in the following analysis. Under such simplification, $k_i(t)$ can be

viewed as the average replication level of the job J_i. Equation (3.1) formulates $\mu_i(t)$ as the following:

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

As in (2.1), the first term of (3.1) approximates the number of slots occupied by the job 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 ($k(t)E[\min(\tau_1, \tau_2, \ldots, \tau_{k_i(T_i(t))})]$, since $k_i(T_i(t))$ copies are created and then they are terminated when the first copy completes.

In this chapter, our goal is to minimize the averaging completion time of all jobs. Similar to Chapter 2, we use maximizing the total completion rate/throughput across all jobs to approximate minimizing average completion time. Mathematically, the problem can be formulated as the following:

$$\begin{aligned} &\text{maximize} \sum_i \mu_i(t) \\ &\text{s.t.,} \quad \sum_i S_i \leq S, \end{aligned} \tag{3.2}$$

where $\mu_i(t)$ follows the form of Equation (3.1).

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

3.3.1 Model Features

Our model incorporates both straggler mitigation policies per job 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

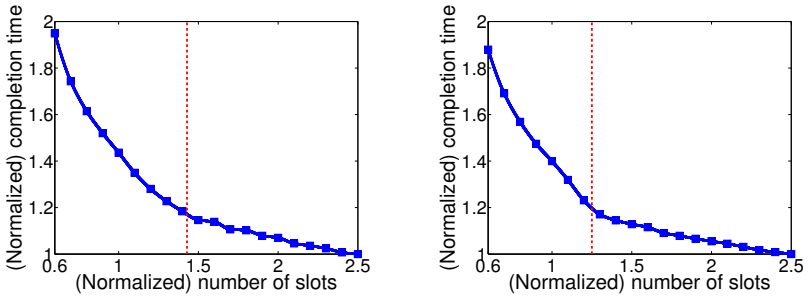
issues are ignored. For example, data locality is not considered. Additionally, it is assumed that the scheduler has perfect knowledge of the remaining work in jobs and that the allocation of slots to jobs can be adjusted dynamically at every point in time. Because of these simplifications, one should 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 Section 3.6.

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 duration distribution and the job's DAGs of tasks). The examples in Section 3.2 illustrate the value of *dynamic* allocation of slots for speculation. Our analysis indicates that this dynamic allocation can be separated into two regimes: whether the cluster is in "high" or "low" load.

The distinction between these two regimes follows from the behavior of the marginal return (in terms of performance) that jobs receive from being allocated slots. It is perhaps natural to expect that the performance of a job will always improve when it is given additional slots (because these can be used for additional speculative copies) and that the value of additional slots has a decreasing marginal return (because an extra slot is more valuable when the job is given few slots than when the job already has many slots). However, surprisingly, a novel observation that leads to the design of Hopper is that the marginal return of an extra slot has a sharp threshold (a.k.a., knee) where, below the threshold, the marginal return is large and (nearly) constant and, above the threshold, the marginal return is small and decreasing.

Figure 3.3 illustrates this threshold using a simulation of a sample job with 200 tasks (with Pareto sizes, common in production traces) and LATE [18] speculation when assigned various numbers of slots. Crucially, there is a marked change in slope beyond the vertical dashed line, indicating the change in the marginal value of a slot. Note that such a threshold exists for different job sizes, speculation algorithms, etc. Further, in the context of a simple model, we can prove the existence of a sharp threshold as shown in Section 3.3.2. We refer to this threshold as the "desired (minimum) allocation" for a job or simply the "virtual job size".

Besides the notation of "virtual job size", our analytic results also highlight 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



(a) $\beta = 1.4$ (b) $\beta = 1.6$

Figure 3.3: The impact of number of slots on single job performance. The number of slots is normalized by job size (number of tasks within the job). β is the Pareto shape parameter for the task size distribution. (In our traces $1 < \beta < 2$.) The red vertical line shows the threshold point.

cluster and the virtual job sizes, which lead to the prototype of the design of our job-level speculation scheduler Hopper (Section 3.3.3). Then we generalize the prototype to provide a knob for balancing performance to fairness in Section 3.3.4.

We focus on homogeneous single-phased jobs in this section and generalize our scheduler design to handle multi-phase jobs (jobs with DAGs) and heterogeneous jobs (jobs with different task duration distributions), as well as data locality in Section 3.4.

3.3.2 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.[3]

A key observation is that the "optimal number of speculative copies", i.e., $k_i(t)$, 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.

[3]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.

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. See Section 2.3 for more details. The Pareto tail parameter β represents the likelihood of stragglers. Roughly, 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 $1 < \beta \leq 2$, our analytic model shows that the optimal (average) speculation per task of a job J_i ($k_i(t)$) 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:

$$k_i(t) = \begin{cases} \frac{2}{\beta}, & S_i \leq \frac{2}{\beta} T_i(t) \\ S_i / T_i(t), & S_i > \frac{2}{\beta} T_i(t). \end{cases} \tag{3.3}$$

Equation 3.3 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. Also, $2/\beta$ corresponds exactly to the vertical line in Figure 3.3.

The first case in Equation 3.3, which corresponds to the early waves 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 3.3), it should not leave slots unused. So, it should speculate aggressively to make use of the capacity available.

Given Equation 3.3, 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":

$$V_i(t) = \frac{2}{\beta} T_i(t). \tag{3.4}$$

In practice, since β may vary over time, it is learned online by Hopper (see Section 3.7) making it adaptive to different threshold points as in Figure 3.3.

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.

3.3.3 Hopper: speculation-aware job scheduler

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 assign every job its virtual size? (ii) How should slots be allocated if there are more than enough slots to assign every job its virtual size? In the following, we first present the analytic results for those two cases in Section 3.3.3.1 and Section 3.3.3.2, then provide the overall design of Hopper in Section 3.3.3.3.

3.3.3.1 Allocation when the cluster is highly utilized

When 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 sizes. Thus, the scheduler should process jobs in ascending order of their virtual sizes $V_i(t)$, giving each job its desired (minimum) allocation until all the slots are exhausted.

Guideline 4. *At all points in time, if there are not enough slots for every job to get its desired (minimum) allocation, i.e., a number of slots equal to its virtual size, then slots should be dedicated to the smallest jobs and each should be given a number of slots equal to its virtual size.*

Intuitively, 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 Section 3.2 highlight, this leads to improved performance. Note that prioritization of small jobs may lead to unfairness for larger jobs, an issue we address shortly in Section 3.3.4.

3.3.3.2 Allocation when the cluster is lightly utilized

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.

In contrast to the high utilization setting, in this situation 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

$$\left(\frac{V_i(t)}{\sum_j V_j(t)}\right) S = \left(\frac{T_i(t)}{\sum_j T_j(t)}\right) S \text{ slots}, \tag{3.5}$$

where S is the number of slots available in the system.

Guideline 5. *At all points in time, if there are enough slots to give every job its desired (minimum) allocation, then the slots should be shared "proportionally" 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 intuition for this design is as follows. Given that all jobs are already receiving their (minimum) desired level of speculation, scheduling is less important than speculation. Thus, prioritization of small jobs is not crucial, and the goal should be to extract the maximum value from speculation. Since stragglers are more likely to occur in larger jobs (stragglers occur in proportion to the number of tasks in a job, on average[4]), the marginal improvement in performance due to an additional slot is proportionally higher for large jobs. Thus, they should get prioritization in proportion to their size when allocating the extra slots.

Since the guidelines specify allocations at the granularity of every job, it is easy to cope with any fluctuations in cluster load (say, from lightly to highly utilized) in an online system.

```
procedure HOPPER(⟨Job⟩ J, int S, float β)
totalVirtualTasks ← 0
    for each Job j in J do
j.V_rem = (2/β) j.T_rem
                                          ▷ j.T_rem: remaining number of tasks
                                  ▷ j.V_rem: virtual remaining number of tasks
totalVirtualTasks += j.V_rem
    SortAscending(J, V_rem)
    if S < totalVirtualTasks then
        for each Job j in J do
j.slots ← ⌊min(S, j.V_rem)⌋
S ← max(S − j.slots, 0)
    else
        for each Job j in J do
j.slots ← ⌊(j.V_rem/totalVirtualTasks) S⌋
```

Pseudocode 3: Hopper (simple version) for jobs in set J with S slots in the cluster and shape parameter β.

3.3.3.3 The design of Hopper

Algorithm 1 (also see Pseudocode 3) combines the above two guidelines for homogeneous single phase jobs. And, we have the following theorem.

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 ascending 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 $\frac{2}{\beta}\sum T_i(t) < S$, then assign $S_i = \left(\frac{T_i(t)}{\sum T_j(t)}\right) S$ for all jobs $J_i \in J(t)$.*

Theorem 2. *Algorithm 1 is throughput optimal for single-phased jobs, i.e., it maximizes $\sum \mu_i(t)$.*

Proof. Recall that

$$\mu_i(t) = \min(S_i, T_i(t)k_i(t)) \times \left(\frac{E[\tau]}{k_i(t)E\left[\min(\tau_1, \ldots, \tau_{k_i(t)}\right]}\right). \tag{3.6}$$

[4]Machines in the cluster are equally likely to cause a straggler [15]; known problematic machines are already blacklisted (see Section 3.1).

And the optimal speculation level $k_i(t)$ satisfies,

$$k_i(t) = \begin{cases} \frac{2}{\beta}, & S_i \leq \frac{2}{\beta} T_i(t) \\ S_i / T_i(t), & S_i > \frac{2}{\beta} T_i(t). \end{cases} \tag{3.7}$$

Plugging the optimal speculation policy given in (3.7) into the model for $\mu_i(t)$ in (3.6) yields the following model for the completion rate:

$$\mu(t) = \begin{cases} \frac{\beta^2}{4(\beta-1)} S_i, & S_i \leq \frac{2}{\beta} T_i(t) \\ \frac{\beta}{\beta-1} T_i(t) - \frac{1}{\beta-1} \frac{T_i(t)^2}{S_i}, & S_i > \frac{2}{\beta} T_i(t). \end{cases} \tag{3.8}$$

We divide the problem into two cases based on the relationship of the total number of slots, S, and the sum of virtual sizes for all jobs, $\frac{2}{\beta} \sum T_i(t)$.

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,

$$\begin{aligned} & \frac{\beta}{\beta-1} T_i(t) - \frac{1}{\beta-1} \frac{T_i(t)^2}{S_i} \\ =& \frac{1}{\beta-1} \frac{1}{S_i} \left(-T_i(t)^2 + \beta T_i(t) S_i - (\frac{\beta S_i}{2})^2 \right) + \frac{1}{\beta-1} \frac{1}{S_i} (\frac{\beta S_i}{2})^2 \\ =& - \frac{1}{(\beta-1) S_i} (T_i(t) - \frac{\beta S_i}{2})^2 + \frac{\beta^2}{4(\beta-1)} S_i \\ \leq& \frac{\beta^2}{4(\beta-1)} S_i. \end{aligned} \tag{3.9}$$

The above inequality implies that if any job that is assigned fewer 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 no slots remains.

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

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

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

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

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

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

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 $J_i \in J_2(t)$, $\frac{T_i(t)}{S_i} = \text{const.}$, i.e., for all $J_i \in J_2(t)$, $S_i = \frac{T_i(t)}{\sum_{J_2(t)} T_j(t)} \sum_{J_2(t)} S_i$.

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 satisfy,

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 $J_i \in J(t)$.

□

3.3.4 Incorporating Fairness

While fairness is an important constraint in clusters, conversations with data center operators reveal that it is not an absolute requirement. Thus, we relax the notion of fairness currently employed by cluster schedulers, e.g., [25], which enforce that if there are $N(t)$ active jobs and S available slots at time t, then each job is assigned $S/N(t)$ slots.

Specifically, to allow some flexibility while still tightly controlling unfairness, 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 $(1 - \epsilon)S/N(t)$ slots at *all* times t. The fairness knob $\epsilon \to 0$ indicates absolute fairness while $\epsilon \to 1$ focuses on performance.

Hopper can be adjusted to guarantee ϵ-fairness in a very straightforward manner. In particular, if a job receives slots fewer than its fair share, i.e., fewer than $(1 - \epsilon)S/N(t)$ slots, the job's capacity assignment is increased to $(1 - \epsilon)S/N(t)$. Next, the remaining slots are allocated to the remaining jobs according to Guidelines 4 or 5, as appropriate. 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* $N = \sum_{i=1}^{n} T_i(t)$. *And define* m_1 *such that* $i \leq m_1$ *implies* $\frac{2}{\beta}T_i(t) \leq \frac{S}{N} - \epsilon$.

1. *If* $S \leq \frac{2}{\beta}\sum_{i=m_1+1}^{n} T_i(t) + m_1\left(\frac{S}{N} - \epsilon\right)$, *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*
$$\frac{T_{m_2+1}(t)}{\sum_{i=m_2+1}^{N} T_i(t)}(S - m_2(\frac{S}{N} - \epsilon)) \geq \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta}T_{m_2+1}(t)\}.$$
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* $m_2 + 1 \leq i \leq N$.

Theorem 3. *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: $S \leq \frac{2}{\beta}\sum_{i=m_1+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m_1$.

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 slots remain. 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 there are insufficient slots 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 $S \leq \frac{2}{\beta}\sum_{i=m_1+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m_1$, 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 no slots remain.

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

When $S \geq \frac{2}{\beta}\sum_{i=m+1}^{N} T_i(t) + (\frac{S}{N} - \epsilon)m$, 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,

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

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,

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{N}\frac{T_i(t)^2}{S_i} \qquad (3.10) \\
\text{subject to} \quad & \sum_{i=1}^{N} S_i = S \\
& S_i \geq \frac{2}{\beta}T_i(t), i = m+1, \ldots, N \\
& S_i \geq \frac{S}{N} - \epsilon, i = 1, \ldots, m.
\end{aligned}$$

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

$$\sum_{i=1}^{N}\frac{T_i(t)^2}{S_i} + v(\sum_{i=1}^{N} S_i - S) + \sum_{i=1}^{m}\lambda_i(\frac{S}{N} - \epsilon - S_i) + \sum_{i=m+1}^{N}\lambda_i(\frac{2}{\beta}T_i(t) - S_i). \qquad (3.11)$$

From KKT condition, in optimal solution,

$$-\frac{T_i(t)^2}{S_i^2} + v - \lambda_i = 0, \qquad (3.12)$$

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

$$\lambda_i \neq 0 \Leftrightarrow S_i = \max\{\frac{2}{\beta}T_i(t), \frac{S}{N} - \epsilon\}. \qquad (3.13)$$

Substituting (3.13) into (3.12), 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 are 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 (3.10). □

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

   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 $\frac{T_i(t)}{\sum\limits_{J_2(t)+J_3(t)} T_i(t)}(S_2 + S_3) \geq \frac{2}{\beta} T_i(t)$, and $\frac{T_i(t)}{\sum\limits_{J_2(t)+J_3(t)} T_i(t)}(S_2 + S_3) \geq \frac{S}{N} - \epsilon$ (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 2, 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 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. $1 \le m_2 \le m_1$,
2. $\frac{T_i(t)}{\sum_{i=n_1+1}^{N} T_i(t)} (S - n_1(\frac{S}{N} - \epsilon)) \ge \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta} T_i(t)\}$, for all $i \ge n_1 + 1$.

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

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

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

In summary, the optimal scheduling should:

1. If $S \le \frac{2}{\beta} \sum_{i=m_1+1}^{n} T_i(t) + m_1 \left(\frac{S}{N} - \epsilon\right)$, 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
$$\frac{T_{m_2+1}(t)}{\sum_{i=m_2+1}^{N} T_i(t)} (S - m_2(\frac{S}{N} - \epsilon)) \ge \max\{\frac{S}{N} - \epsilon, \frac{2}{\beta} T_{m_2+1}(t)\}.$$

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 $m_2 + 1 \leq i \leq N$.

□

Our experimental results (Section 3.7.3) highlight that even at moderate values of ϵ, *nearly all jobs finish faster than they would have under fair scheduling*. This fact, though initially surprising, is similar to the conclusions about SRPT-like policies. Despite being intuitively unfair to large job sizes, it in fact improves the average response time of every job size (when job sizes are heavy-tailed) compared to fair schedulers [89–91].

3.4 Hopper in real systems

The design guidelines we have discussed so far are based on homogeneous single-phased jobs. In this section, we extend Hopper to handle more complex real-world DAGs of jobs (Section 3.4.1) with heterogeneous distributions (β) of task durations (Section 3.4.2). Further, we discuss how to incorporate data locality in Hopper in Section 3.4.3. Our generic model ensures that the guidelines require only minor adjustments.

3.4.1 Incorporating DAGs of Tasks

In practice, many jobs are defined by multiple-phased DAGs, where the phases are typically *pipelined*. That is, downstream tasks do not wait for *all* the upstream tasks to finish but read the upstream outputs as the tasks finish, e.g., [92]. Pipelining is beneficial because the upstream tasks are typically bottlenecked on other *non-overlapping* resources (CPU, memory), while the reading downstream takes network resources. The additional complexity DAGs create for our guidelines is the need to balance the gains due to overlapping network utilization with the improvements that come from favoring upstream phases with fewer remaining tasks.

We integrate this tradeoff into Hopper using a weighting factor, α per job, set to be the ratio of remaining work in the downstream phase's network transfer to the remaining work in the upstream phase. Specifically, α favors jobs with higher remaining communication and lower remaining tasks in the current phase. The exact details of estimating α are deferred to Section 3.6.3.

Given the weighting factor α, there are two key adjustments that we make to the guidelines discussed so far. First, in Guideline 4, the prioritization of jobs based on the virtual size $V_i(t)$ is replaced by a prioritization based on $\max\{V_i(t), V_i'(t)\}$, where $V_i(t)$ is the virtual remaining number of tasks in the current phase and $V_i'(t)$ is the virtual remaining work in communication in the downstream phase.[5] Second, we redefine the virtual size itself as $V_i(t) = \frac{2}{\beta} T_i(t)\sqrt{\alpha_i}$.

Mathematically, following a similar proof to Theorem 1, it can be shown that to maximize the α-weighted throughput, i.e. $\sum_i \alpha_i \mu_i(t)$, the virtual size/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 the 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 $S \leq \frac{V_1(t)}{\sqrt{\alpha_{\min}^{(2)}}}$, assign $S_1 = S$ and $S_i = 0$ for $i > 1$.*
2. *If $\exists\ k < n$ such that $\sum_{i=1}^{k} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}} < S \leq \sum_{i=1}^{k+1} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}}$, 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 slots remain, and $S_i = 0$ for $i > k + 1$.*
3. *If $\exists\ k < n - 1$ such that $\sum_{i=1}^{k+1} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+1)}}} < S \leq \sum_{i=1}^{k+1} \frac{V_i(t)}{\sqrt{\alpha_{\min}^{(k+2)}}}$, 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 Section 3.7.3. 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 [93].

[5] Results in [83] show that picking the $\max\{T_i(t), T_i'(t)\}$ is 2-speed optimal for completion times when stragglers are not considered.

Estimating α: Note that 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 up front. 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 [54, 55], Hopper can do better. It uses the ratio of intermediate to input data of the completed tasks as a predictor for the future (incomplete) tasks. Data from Facebook's and Microsoft Bing's clusters (described in Section 3.7.1) shows that while the ratio of input to output data size of tasks varies from 0.05 all the way to 18, the ratios *within* tasks of a phase have coefficient-of-variation of only 0.07 and 0.24 at the median and 90$^{\text{th}}$ 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).

Our simple approach to estimating α works sufficiently well for our evaluations (accuracy of 92%, on average). However, we realize that workloads without many multi-waved or recurring jobs and without tasks whose duration is dictated by their input sizes need more sophisticated models of task executions.

3.4.2 Incorporating heterogeneous jobs

In Section 3.3, we assume all jobs have the same task duration distributions, i.e., have the same likelihood of stragglers. In real system, different types of jobs may have different task duration distributions. Data locallity and cluster characteristics such as resource contentions due to utilization and hotspots [94] can also lead to different task duration distributions.

Heterogeneous task duration distributions can have a significant impact on scheduling. Speculation is more beneficial for jobs that have stragglers more often, but at the expense of the launching of other jobs.

For ease of presentation, here we only discuss the the case when there are two classes of jobs in the system. The general scenario with multiple classes of jobs can be easily extended from the two classes case.

Given the two classes of jobs with the task duration distributions $\beta_i, i \in \{I, II\}$ respectively, the key adjustment that we make to the guidelines discussed so far is to redefine the virtual sizes based on the relative stragglers' likelihood across two job classes.

Algorithm 4 (Heterogenous stragglers)**.**
Let $I(t)$ and $II(t)$ denote the sets 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 $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum_{I(t)} T_i(t) < S \leq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, 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 $S \geq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}} \sum_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, 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 3 to assign capacity within each type.

Theorem 4. *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 1. The remaining question is how to allocate slots across different types.

Similar to the proof for Theorem 2, we divide the problem into three cases.

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

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 $S \leq \frac{2}{\beta_1} \sum_{I(t)} T_i(t)$, from (3.9), we know we should assign all slots to type 1 jobs.

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

When $\frac{2}{\beta_1} \sum_{I(t)} T_i(t) \leq S \leq \frac{2}{\beta_1} \sum_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, 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 (3.9), 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 $S_1 \geq \frac{2}{\beta_1} \sum_{I(t)} T_i(t)$ and $S_2 \leq \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$. The total throughput is,

$$\begin{aligned}
&\frac{\beta_1}{\beta_1 - 1} \sum_{I(t)} T_i(t) - \frac{1}{(\beta_1 - 1)S_1} (\sum_{I(t)} T_i(t))^2 + \frac{\beta_2^2}{4(\beta_2 - 1)} S_2 \\
=&\frac{\beta_1}{\beta_1 - 1} \sum_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2 - 1)} S - \frac{1}{(\beta_1 - 1)} (\sum_{I(t)} T_i(t))^2 \frac{1}{S_1} - \frac{\beta_2^2}{4(\beta_2 - 1)} S_1 \qquad (3.14) \\
\leq&\frac{\beta_1}{\beta_1 - 1} \sum_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2 - 1)} S - \frac{\beta_2}{\sqrt{(\beta_1 - 1)(\beta_2 - 1)}} \sum_{I(t)} T_i(t),
\end{aligned}$$

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

Equality is achieved when $\frac{\beta_2^2}{4(\beta_2 - 1)} S_1 = \frac{1}{(\beta_1 - 1)S_1} (\sum_{I(t)} T_i(t))^2$, i.e., $S_1 = \frac{2}{\beta_2} \sqrt{\frac{\beta_2 - 1}{\beta_1 - 1}} \sum_{I(t)} T_i(t)$. And (3.14) increases for $S_1 \leq \frac{2}{\beta_2} \sqrt{\frac{\beta_2 - 1}{\beta_1 - 1}} \sum_{I(t)} T_i(t)$ and decreases afterwards. Also note that, if $S_1 = \frac{2}{\beta_2} \sqrt{\frac{\beta_2 - 1}{\beta_1 - 1}} \sum_{I(t)} T_i(t)$, then $S_1 \geq \frac{2}{\beta_1} \sum_{I(t)} T_i(t)$. Thus, when $\frac{2}{\beta_1} \sum_{I(t)} T_i(t) \leq S \leq \frac{2}{\beta_1} \sum_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, the optimal scheduling should:

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

2. when $\frac{2}{\beta_2} \sqrt{\frac{\beta_2 - 1}{\beta_1 - 1}} \sum_{I(t)} T_i(t) < S \leq \frac{2}{\beta_1} \sum_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, 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.

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

When $S \geq \frac{2}{\beta_1} \sum_{I(t)} T_i(t) + \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, similarly, from (3.9), in optimal scheduling, the number of slots assigned to type 1 jobs should be no less than the optimal speculation scheduling level, so $S_1 \geq \frac{2}{\beta_1} \sum_{I(t)} T_i(t)$. Depending on how many slots we have, S_2 can be either less than or more than the optimal speculation level. We discuss the two cases separately in the following.

1. If $S_2 \leq \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, which implies $S - S_1 \leq \frac{2}{\beta_2} \sum_{II(t)} T_i(t)$, as we already proved, $S_1 = \max\{\frac{2}{\beta_2} \sqrt{\frac{\beta_2 - 1}{\beta_1 - 1}} \sum_{I(t)} T_i(t), S - \frac{2}{\beta_2} \sum_{II(t)} T_i(t)\}$, and $S_2 = S - S_1$. Specifically,

a) when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) \geq S - \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, if $S_1 \geq \frac{2}{\beta_1}\sum_{I(t)} T_i(t)$ and $S_2 \leq \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$ in optimal scheduling, then $S_1 = \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t)$ and $S_2 = S - S_1$.

b) when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) < S - \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, if $S_1 \geq \frac{2}{\beta_1}\sum_{I(t)} T_i(t)$ and $S_2 \leq \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$ in optimal scheduling , then $S_1 = S - \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$ and $S_2 = S - S_1$.

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

$$
\begin{aligned}
&\frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) - \frac{1}{(\beta_1-1)S_1}(\sum_{I(t)} T_i(t))^2 + \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) - \frac{1}{(\beta_2-1)S_2}(\sum_{II(t)} T_i(t))^2 \\
=&\frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) - \frac{1}{(\beta_1-1)S_1}(\sum_{I(t)} T_i(t))^2 - \frac{1}{(\beta_2-1)S_2}(\sum_{II(t)} T_i(t))^2 \\
\leq&\frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) - \frac{1}{S_1+S_2}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2 \\
=&\frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) - \frac{1}{S}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2,
\end{aligned}
\tag{3.15}
$$

where the inequality follows from the Cauchy-Schwartz inequality.

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

Combining above two results, when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) < S - \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, the optimal assignment from the first case is a boundary point for the second case. Obviously, the optimal assignment from case 2 is the global optimal assignment. But, when $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) \geq S - \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, it still remains 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 (3.14), throughput $\mu_s = \frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2^2}{4(\beta_2-1)}S - \frac{\beta_2}{\sqrt{(\beta_1-1)(\beta_2-1)}}\sum_{I(t)} T_i(t)$.

b) In case 2, from (3.15), throughput $\mu_l = \frac{\beta_1}{\beta_1-1}\sum_{I(t)} T_i(t) + \frac{\beta_2}{\beta_2-1}\sum_{II(t)} T_i(t) - \frac{1}{S}\left(\sqrt{\frac{1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \sqrt{\frac{1}{\beta_2-1}}\sum_{II(t)} T_i(t)\right)^2$.

Then we have,

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

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

In summary, the optimal scheduling should:

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

2. If $\frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) < S \leq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, 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 $S \geq \frac{2}{\beta_2}\sqrt{\frac{\beta_2-1}{\beta_1-1}}\sum_{I(t)} T_i(t) + \frac{2}{\beta_2}\sum_{II(t)} T_i(t)$, 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.

□

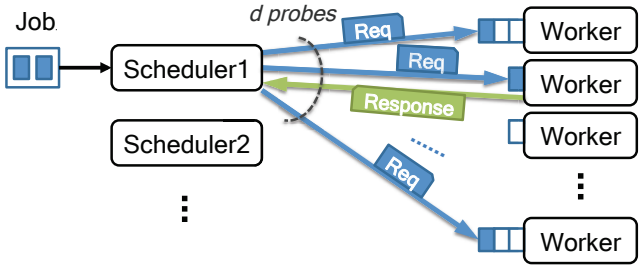


Figure 3.4: Decentralized scheduling architecture.

3.4.3 Incorporating Data Locality

As such, the guidelines presented do not consider data locality [80, 94] in the scheduling of tasks. Tasks reading their data from remote machines over the network run slower. In addition, such remote reads also increase contention with other intermediate tasks (like reduce tasks) that are bound to read over the network.

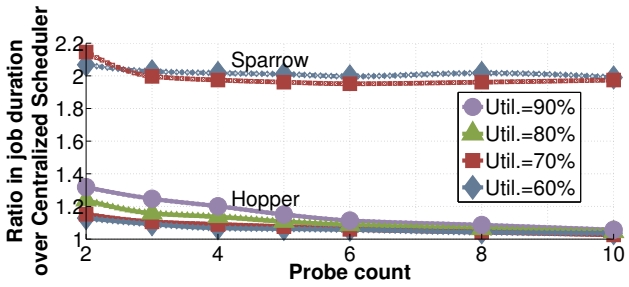
We devise a simple relaxation approach for balancing adherence to our guidelines and locality. Specifically, we adjust the ordering of jobs in Guideline 4 to include information about locality. Instead of allotting slots to the jobs with the smallest virtual sizes, we allow for picking any of the smallest k% of jobs whose tasks can run with data locality on the available slots. In practice, a small value of k (≤ 5%) suffices due to high churn in task completions and slot availabilities (Section 3.7.4).

3.5 Decentralized Hopper

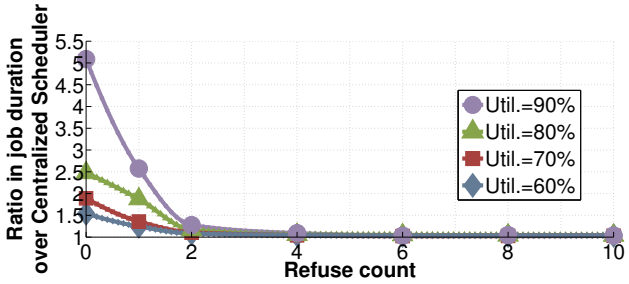
In this section, we adapt the guidelines described in Section 3.3 to design a decentralized (online) scheduler. Decentralized schedulers are increasingly prominent as cluster sizes grow. As we explain in this section, a key benefit of our guidelines in Section 3.3 is that they can be decentralized with little performance loss.

Decentralized schedulers, like the recently proposed Sparrow [20], broadly adopt the following design (see Figure 3.4). There are multiple independent *schedulers* each of which is responsible for scheduling one or a subset of jobs; for simplicity, a single job never spans across schedulers. Every scheduler assigns the tasks of its jobs to machines in the cluster (referred to as *workers*) that execute the tasks. The architecture allows for an incoming job to be assigned to any of the available schedulers, while also seamlessly allowing new schedulers to be dynamically spawned.

A scheduler first pushes *reservation requests* for its tasks to workers; each request



(a) Number of probes, d



(b) Number of refusals

Figure 3.5: The impact of number of probes and number of refusals on Hopper's performance.

contains the identifier of the scheduler placing the request along with the remaining number of unscheduled tasks in the job. When a worker is vacant, it pulls a task from the corresponding scheduler based on the reservation requests in its waiting queue. In this framework, workers decide which job's task to run and the scheduler for the corresponding job decides which task to run within the chosen job. This decoupling naturally facilitates the design of Hopper.

Though we adopt an overall design structure similar to Sparrow for the decentralization of Hopper, it is important to note that Hopper's design is fundamentally different because it integrates straggler mitigation based on the guidelines behind Hopper introduced in Section 3.3.

Decentralizing Hopper involves the following steps: approximating worker-wide information at each scheduler (Section 3.5.1), deciding if the number of slots is constrained (Section 3.5.2), and calculating virtual sizes (Section 3.5.3).

3.5.1 Power of Many Choices

Decentralized schedulers have to *approximate* the global state of the cluster – the states of all the workers – since they are unaware of other jobs in the system. A

common way to accomplish this is via the "power of two choices" [95]. This celebrated and widely used result highlights that, in many cases, one nearly matches the performance of a centralized implementation by querying two workers for their queue lengths, and choosing the shorter of the queues. In fact, this intuition underlies the design of Sparrow as well, which combines the idea with a form of "late binding"; schedulers send reservation requests for every task to two workers and then let workers pull a task from the corresponding scheduler when they have a free slot. We adopt "late binding", as used in Sparrow, but replace the "power of two choices" with the "power of many choices".

The reason for this change is that the effectiveness of the "power of two choices" relies on having *light-tailed task size distributions*. The existence of stragglers means that, in practice, task durations are heavy-tailed, e.g., [15, 19, 74]. Recent theoretical results have proven that, when task sizes are heavy-tailed, probing $d > 2$ choices can provide orders-of-magnitude improvements [96]. The value in using $d > 2$ comes from the fact that large tasks, which are more likely under heavy-tailed distributions, can cause considerable backing up of worker queues. Two choices may not be enough to avoid such backed-up queues, given the high frequency of straggling tasks. More specifically, $d > 2$ allows the schedulers to have a view of the jobs that is closer to the global view.

We use simulations in Figure 3.5a to highlight the benefit of using $d > 2$ probing choices in Hopper and to contrast this benefit with Sparrow, which relies on the power of two choices. Our simulation considers a cluster of 50 schedulers and 10,000 workers and jobs with Pareto distributed ($\beta = 1.5$) task sizes. Job performance with decentralized Hopper is within just 15% of the centralized scheduler; the difference plateaus beyond $d = 4$. Note that Sparrow (which does not coordinate scheduling and speculation) is > 100% off for medium utilizations and even further off for high utilizations (not shown on the figure in order to keep the scale visible). Further, workers in Sparrow pick tasks in their waiting queues in a FCFS fashion. The lack of coordination between scheduling and speculation results in a long waiting time for speculative copies in the queues which diminishes the benefits of multiple probes. Thus Sparrow cannot extract the same benefit Hopper has from using more than two probes. Of course, these are rough estimates since the simulations do not capture overheads due to increased message processing, which are included in the evaluations in Section 3.7.

```
procedure ResponseProcessing(Response response )
    Job j ← response.job
    if response.type = non-refusable then
        Accept()
    else
if (j.current_occupied < j.virtual_size) Accept ()
else Refuse()
```

Pseudocode 4: Scheduler Methods.

3.5.2 Is the system capacity constrained?

In the decentralized setting workers implement our scheduling guidelines. Recall that Guideline 4 or Guideline 5 is applied depending on whether the system is constrained for slots or not. Thus, determining which to follow necessitates comparing the sum of virtual sizes of all the jobs and the number of slots in the cluster, which is trivial in a centralized scheduler but requires communication in an decentralized setting.

To keep overheads low, we avoid costly gossiping protocols among schedulers regarding their states. Instead, we use the following adaptive approach. Workers start with the conservative assumption that the system is capacity constrained (this avoids overloading the system with speculative copies), and thus each worker implements Guideline 4, i.e., enforces an SRPT priority on its queue. Specifically, when a worker is idle, it sends a *refusable* response to the scheduler corresponding to the reservation request of the job it chooses from its queue. However, since the scheduler queues many more reservation requests than tasks, it is possible that its tasks may have all been scheduled (with respect to virtual sizes). A refusable response allows the scheduler to refuse sending any new task for the job if the job's tasks are all already scheduled to the desired speculation level (ResponseProcessing in Pseudocode 4). In its refusal, it sends information about the job with the smallest virtual size in its list which still has unscheduled tasks (if such an "unsatisfied" job exists).

Subsequently, the worker sends a refusable response to the scheduler corresponding to second smallest job in its queue, and so forth till it gets a threshold number of refusals. Note that the worker avoids probing the same scheduler more than once. Several consecutive refusals from schedulers without information about any unsatisfied jobs suggests that the system is not capacity constrained. At that point, it switches to implementing Guideline 5. Once it is following Guideline 5, the worker randomly picks a job from the waiting queue based on the distribution of job

```
procedure RESPONSE(⟨Job⟩ J, int refused_count)
                    ▷ J: list of jobs in queue of the worker excluding already refused jobs
    if refused_count ≥ refusal_threshold then
        j ← J.PickAtRandom()
        SendResponse(j, non-refusable)
    else
        j ← J. min(virtual_size)
        SendResponse(j, refusable)
```

Pseudocode 5: Worker: choosing the next task to schedule.

virtual sizes. If there are still unsatisfied jobs at the end of the refusals, the worker sends a *non-refusable* response (which cannot be refused) to the scheduler whose unsatisfied job is the smallest. Pseudocode 5 explains the Response method.

The higher the threshold for refusals, the better the view of the schedulers for the worker. Our simulations (with 50 schedulers and 10,000 workers) in Figure 3.5b show that performance with two or three refusals is within 10% – 15% of the centralized scheduler.

3.5.3 Updating Virtual Job Sizes

Computing the remaining virtual job size at a scheduler is straightforward. However, since the remaining virtual size of a job changes as tasks complete, virtual sizes need to be updated dynamically. Updating virtual sizes accurately at the workers that have queued reservations for tasks of this job would require frequent message exchanges between workers and schedulers, which would create significant overhead in communication and processing of messages. So, our approach is to piggyback updates for virtual sizes on other communication messages that are anyway necessary between a scheduler and a worker (e.g., schedulers sending reservation requests for new jobs, workers sending responses to probe system state and ask for new tasks). While this introduces a slight error in the virtual remaining sizes, our evaluation shows that the approximation provided by this approach is enough for the gains associated with Hopper.

Crucially, the calculation of virtual sizes is heavily impacted by the job specifics. Job specific properties of the job DAG and the likelihood of stragglers are captured through α and β, respectively, which are learned online. Note that jobs from different applications may have heterogeneous α and β.

3.6 Implementation Overview

We now give an overview of the implementation of Hopper in decentralized and centralized settings.

3.6.1 Decentralized Implementation

Our decentralized implementation uses the Sparrow [20] framework, which consists of many schedulers and workers (one each on every machine) [97]. Arbitrarily many schedulers can operate concurrently; though we use 10 in our experiments. Schedulers allow submissions of jobs using Thrift RPCs [98].

A job is broken into a set of tasks with their dependencies (DAG), binaries and locality preferences. The scheduler places requests at the workers for its tasks; if a task has locality constraints, its requests are only placed on the workers meeting its constraints [52, 54, 99]. The workers talk to the client executor processes (e.g., Spark executor). The executor processes are responsible for executing task binaries and are long-lived to avoid startup overheads (see [20] for a more detailed explanation).

Our implementation modifies the scheduler as well as the worker. The workers implement the core of the guidelines in Section 3.3 – determining if the system is slot-constrained and accordingly prioritizing jobs as per their virtual sizes. This required modifying the FIFO queue at the worker in Sparrow to allow for custom ordering of the queued requests. The worker, nonetheless, augments its local view by coordinating with the scheduler. This involved modifying the "late binding" mechanism both at the worker and scheduler. The worker, when it has a free slot, works with the scheduler in picking the next task (using Pseudocode 5). The scheduler deals with a response from the worker as per Pseudocode 4.

Even after *all* the job's tasks have been scheduled (including its virtual size), the job scheduler does not "cancel" its pending requests; there will be additional pending requests with any probe ratio over one. Thus, if the system is not slot-constrained, it would be able to use more slots (as per Guideline 5).

In our decentralized implementation, for tasks in the input phase (e.g., map phase), when the number of probes exceeds the number of data replicas, we queue up the additional requests at randomly chosen machines. Consequently, these tasks *may* run without data locality, and our results in Section 3.7 include such loss in locality.

3.6.2 Centralized Implementation

We implement Hopper inside two centralized frameworks: Hadoop YARN (version 2.3) and Spark (version 0.7.3). Hadoop jobs read data from HDFS [5] while Spark jobs read from in-memory RDDs.

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, which 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.

3.6.3 Estimating Intermediate Data Sizes

Recall from Section 3.4.1 that our scheduling guidelines recommend scaling every job's allocation by $\sqrt{\alpha}$ in the case of DAGs. The purpose of the scaling is to capture pipelining of the reading of upstream tasks' outputs.

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 up front. 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.

Our simple approach to estimating α works sufficiently well for our evaluations (accuracy of 92%, on average). However, we realize that workloads without many multi-waved or recurring jobs and without tasks whose duration is dictated by their input sizes need more sophisticated models of task executions.

3.7 Evaluation

We evaluate our prototypes of Hopper – with both decentralized and centralized scheduling – on a 200 machine cluster. We focus on the overall gains of the

decentralized prototype of Hopper in Section 3.7.2 and evaluate the design choices that led to Hopper in Section 3.7.3. Then, in Section 3.7.4, we evaluate the gains with Hopper in a centralized scheduler in order to highlight the value of coordinating scheduling and speculation. The key highlights are:

1. Hopper's decentralized prototype improves the average job duration by up to 66% compared to an aggressive decentralized baseline that combines Sparrow with SRPT (Section 3.7.2).

2. Hopper ensures that only 4% of jobs slow down compared to Fair scheduling, and jobs which do slow down do so by $\leq 5\%$ (Section 3.7.3).

3. Centralized Hopper improves job completion times by 50% compared to centralized SRPT (Section 3.7.4).

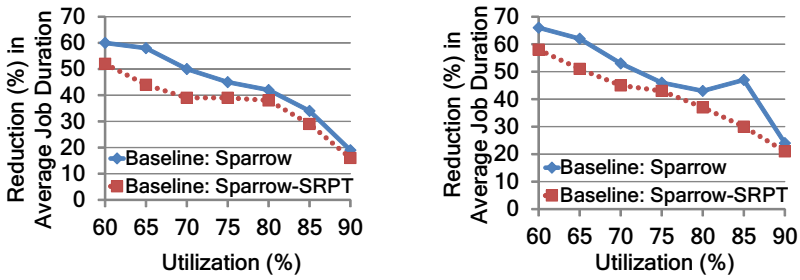
3.7.1 Setup

Cluster Deployment: We deploy our prototypes on a 200-node private cluster. Each machine has 16 cores, 34GB of memory, 1Gbps network and 4 disks. The machines are connected using a network with no over-subscription.[6]

Workload: Our evaluation runs jobs in traces from Facebook's production Hadoop [4] cluster ($3,500$ machines) and Microsoft Bing's Dryad cluster (O(1000) machines) from Oct-Dec 2012. The traces consist of a mix of experimental and production jobs. Their tasks have diverse resource demands of CPU, memory and IO, varying by a factor of 24× (refer to [100] for detailed quantification). We retain the inter-arrival times of jobs, their input sizes and number of tasks, resource demands, and job DAGs of tasks. Job sizes follow a heavy-tailed distribution (quantified in detail in [15]). Each experiment is a replay of a representative 6 hour slice from the trace. It is repeated five times and we report the median.

To evaluate our prototype of decentralized Hopper, we use in-memory Spark [52] jobs. These jobs are typical of interactive analytics whose tasks vary from sub-second durations to a few seconds. Since the performance of any decentralized scheduler depends on the cluster utilization, we speed up the trace appropriately, and evaluate on (average) utilizations between 60% and 90%, consistent with Sparrow [20].

[6]Results with a 10Gbps network are qualitatively similar.



(a) Facebook (b) Bing

Figure 3.6: Hopper's gains with cluster utilization.

Stragglers: The stragglers in our experiments are those that occur *naturally*, i.e., not injected via any model of a probability distribution or via statistics gathered from the Facebook and Bing clusters. Importantly, the frequency and lengths of stragglers observed in our evaluations are consistent with prior studies, e.g., [17–19]. While Hopper's focus is not on improving straggler mitigation algorithms, our experiments certainly serve to emphasize the importance of such mitigation.

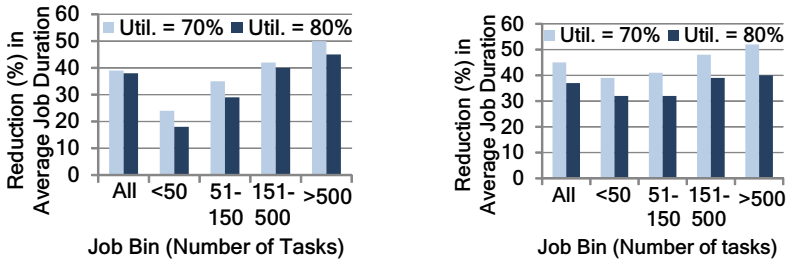
Baseline: We compare decentralized Hopper to Sparrot-SRPT, an augmented version of Sparrow [20]. Like Sparrow, it performs decentralized scheduling using "batched" power-of-two choices. In addition, it also includes an SRPT heuristic. In short, when a worker has a slot free, it picks the task of the job that has the least unfinished tasks (instead of the standard FIFO ordering in Sparrow). Finally, we combine Sparrow with LATE [18] using "best effort" speculation (Section 3.2); we do not consider "budgeted" speculation due to the difficulty of picking a fixed budget.

The combination of Sparrow-SRPT and LATE performs strictly better than Sparrow, and serves as an aggressive baseline. Our improvements over this aggressive benchmark highlight the importance of coordinating scheduling and speculation.

We compare centralized Hopper to a centralized SRPT scheduler with LATE speculation. Again, this is an aggressive baseline since it sacrifices fairness for performance. Thus, improvements can be interpreted as coming solely from better coordination of scheduling and speculation.

3.7.2 Decentralized Hopper's Improvements

In our experiments, unless otherwise stated, we set the fairness allowance ϵ as 10%, probe ratio as 4 and speculation algorithm in every job to be LATE [18]. Our



(a) Facebook (b) Bing

Figure 3.7: Hopper's gains by job bins over Sparrow-SRPT.

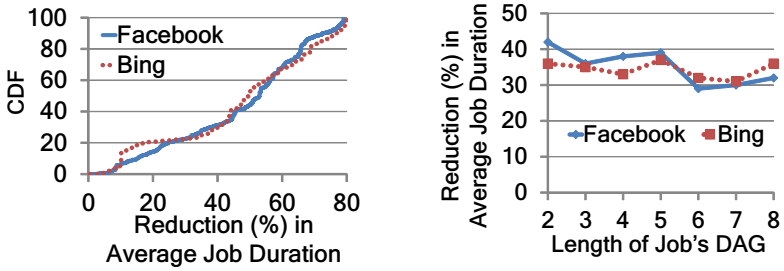
estimation of α (Section 3.6.3) has an accuracy of 92% on average. As the workload executes, we also continually fit the parameter β of task durations based on the completed tasks (including stragglers); the error in β's estimate falls to $\leq 5\%$ after just 6% of the jobs have executed.

Overall Gains: Figure 3.6 plots Hopper's gains for varying utilizations, compared to stock Sparrow and Sparrow-SRPT Jobs, overall, speed up by $50\% - 60\%$ at the utilization of 60%. The gains compared to Sparrow are marginally better than Sparrow-SRPT. When the utilization goes over 80%, Hopper's gains compared to both are similar. An interesting point is that Hopper's gains with the Bing workload in Figure 3.6b are a touch higher (difference of 7%), perhaps due to the larger difference in job sizes between small and large jobs, allowing more opportunity for Hopper. Gains fall to $< 20\%$ when utilization is high ($\geq 80\%$), naturally because there is not much room for any optimization at that occupancy. While not plotted, gains at utilizations $\leq 30\%$ are no more than 14%. Expectedly, at such low utilizations, there is little requirement for smarter speculation or probing.

Note that the above utilizations are averages and there is considerable variation. At 80% average utilization, Hopper allocates 53% of jobs using Guideline 4 (high utilization) and the remaining 47% of jobs using Guideline 5 (low utilization). This indicates that 53% of jobs in the experimental run arrived such that the cluster did not have enough slots to allocate every job its virtual size.

The results so far highlight that Sparrow-SRPT is a more aggressive baseline than Sparrow, and so we compare only to it for the rest of our evaluation.

Job Bins: Figure 3.7 dices the gains by job size (number of tasks). Gains for small jobs are less compared to large jobs. This is expected given that our baseline of Sparrow-SRPT already favors the small jobs. Nonetheless, Hopper's smart allocation



(a) Distribution (b) DAG

Figure 3.8: (a) CDF of Hopper's gains, and (b) gains as the length of the job's DAG varies; both at 60% utilization.

of speculative slots offers 18% – 32% improvement. Gains for large jobs, in contrast, are over 50%. This not only shows that there is sufficient room for the large jobs despite favoring small jobs (due to the heavy-tailed distribution of job sizes [15, 54]) 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 [55], Hopper's allocation becomes important.

Distribution of Gains: Figure 3.8a plots the distribution of gains across jobs. While the median gains are just higher than the average, there are $> 70\%$ gains at higher percentiles. Encouragingly, gains even at the 10^{th} percentile are 15% and 10%, which shows Hopper's ability to improve even worse case performance.

DAG of Tasks: The scripts in our Facebook (Hive scripts [11]) and Bing (Scope [24]) workloads produce DAGs of tasks which often pipeline data transfers of downstream phases with upstream tasks [92]. 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 3.8b shows, Hopper's gains hold across DAG lengths.

Speculation Algorithm: We now experimentally evaluate Hopper's performance with different speculation mechanisms. LATE [18] is deployed in Facebook's clusters, Mantri [17] is in operation in Microsoft Bing, and GRASS citegrass is a recently reported straggler mitigation system that was demonstrated to perform near optimal speculation. Our experiments still use Sparrow-SRPT as the baseline but pair with the different straggler mitigation algorithms. Figure 3.9 plots the results.

While the earlier results were achieved in conjunction with LATE, a remarkable point about Figure 3.9 is the similarity in gains even with Mantri and GRASS.

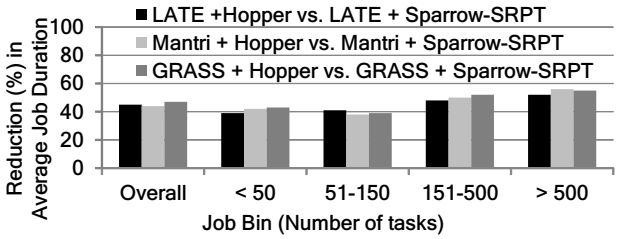


Figure 3.9: Hopper's results are independent of the straggler mitigation strategy.

This indicates that as long as the straggler mitigation algorithms are aggressive in asking for speculative copies, Hopper will appropriately balance speculation and scheduling. Overall, it emphasizes the aspect that resource allocation *across* jobs (with speculation) has a higher performance value than straggler mitigation *within* jobs.

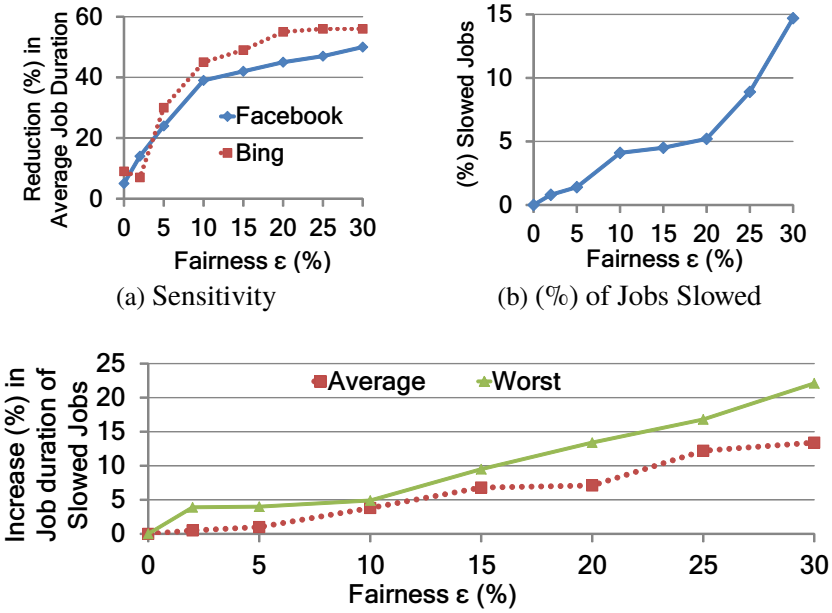
3.7.3 Evaluating Hopper's Design Decisions

We now evaluate the sensitivity of decentralized Hopper to our key design decisions: fairness and probe ratio.

Fairness: As we had described in Section 3.3.4, the fairness knob 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 3.10a plots the increase in gains as we increase ϵ from 0 to 30%. The gains quickly rise for small values of ϵ, and beyond $\epsilon = 15\%$ the increase in gains are flatter with both the Facebook as well as Bing workloads. Conservatively, we set ϵ to 10%.

An important concern, nonetheless, is the amount of *slowdown* of jobs compared to a perfectly fair allocation ($\epsilon = 0$), i.e., when all the jobs are guaranteed their fair share at all times. Any slowdown of jobs is because of receiving fewer slots. Figure 3.10b measures the number of jobs that slowed down, and for the slowed jobs, Figure 3.10c plots their average and worst slowdowns. Note that fewer than 4% of jobs slow down with Hopper compared to a fair allocation at $\epsilon = 10\%$. The corresponding number for the Bing workload is 3.8%. In fact, both the average and worst slowdowns are limited at $\epsilon = 10\%$, thus demonstrating that Hopper's focus on performance does *not* unduly slow down jobs.



(a) Sensitivity

(b) (%) of Jobs Slowed

(c) Magnitude (%) of Slowdown

Figure 3.10: 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).

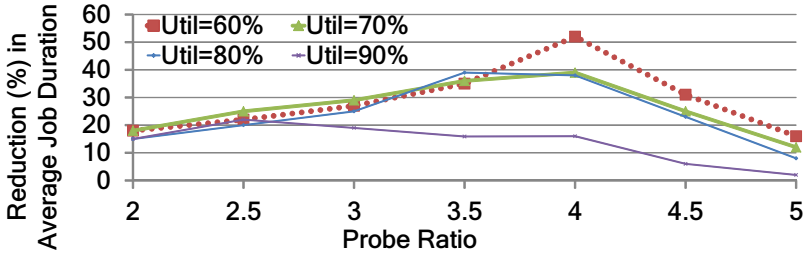
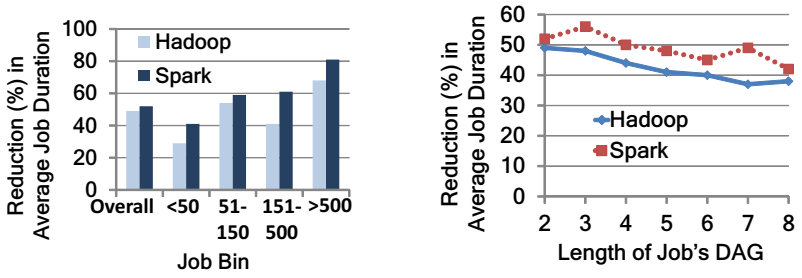


Figure 3.11: Power of d choices: Impact of the number of probes on job completion.

Probe Ratio: An important component of decentralized scheduling is the probe ratio – the number of requests queued at workers to number of tasks in the job. A higher probe ratio reduces the chance of a task being stuck in the queue of a busy machine, but also increases messaging overheads. While the power-of-two choices [95] and Sparrow [20] recommend a probe ratio of 2, we adopt a probe ratio of 4 based on our analysis in Section 3.5.

Figure 3.11 confirms that higher probe ratios are indeed beneficial. As the probe ratio increase from 2 onwards, the payoff due to Hopper's scheduling and straggler mitigation results in gains increasing until 4; at utilizations of 70% and 80%, using



(a) Gains (b) DAG

Figure 3.12: Centralized Hopper's gains over SRPT, overall and broken by DAG length (Facebook workloads).

3.5 works well too. At 90% utilization, however, gains start slipping even at a probe ratio of 2.5. However, the benefits at such high utilizations are smaller to begin with.

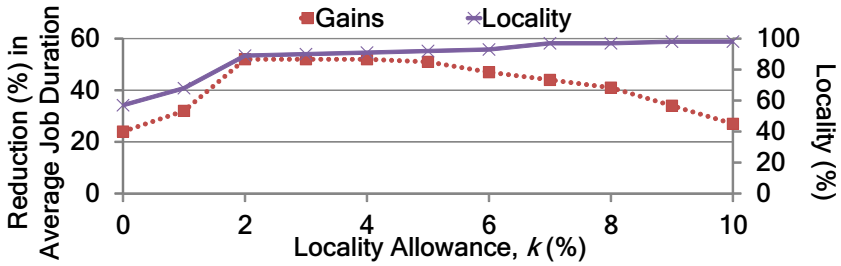
3.7.4 Centralized Hopper's Improvements

To highlight the fact that Hopper is a unified design, appropriate for both decentralized and centralized systems, we also evaluate Hopper in a centralized setting using Hadoop and Spark prototypes. Figure 3.12 plots the gains for the two prototypes with Facebook and Bing workloads. We achieve gains of $\sim$ 50% with the two workloads, with individual job bins improving by up to 80%.

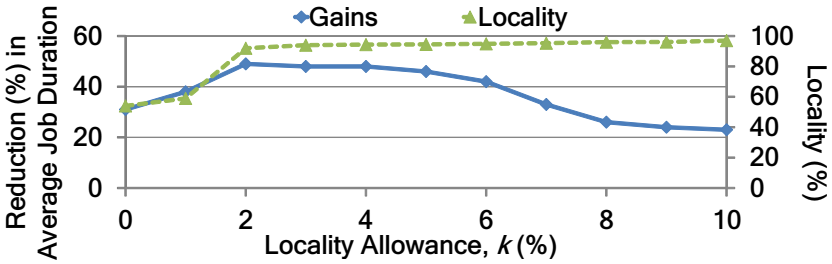
As with the decentralized setting, gains for small jobs are lower due to the baseline of SRPT already favoring small jobs. Between the two prototypes, gains for Spark are consistently higher (albeit, modestly). Spark's small task durations makes it more sensitive to stragglers and thus it spawns many more speculative copies. This makes Hopper's scheduling more crucial.

DAG of Tasks: Like in the decentralized implementation, Hopper's gains hold consistently over varying DAG lengths, see Figure 3.12. Note that there is a contrast between Spark jobs and Hadoop jobs. Spark jobs have fast in-memory map phases, thus making intermediate data communication the bottleneck. Hadoop jobs are less bottlenecked on intermediate data transfer, and spend more of their time in the map phase [54]. This difference is captured via α, which is learned as described in Section 3.6.3.

Data Locality: Recall from Section 3.4.3 that we achieve data locality using a relaxation heuristic to allow any k subsequent jobs (as a percentage of total jobs).



(a) Spark



(b) Hadoop

Figure 3.13: Centralized Hopper: Impact of Locality Allowance (k) (see Section 3.6.2) with Facebook workload.

As Figure 3.13a shows, a small relaxation of $k = 3\%$, which is what we have used so far, achieves appreciable increase in locality in Spark. Gains are 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, increases with k (Figures 3.13a). Hadoop results are similar (3.13b).

Note that even when we enhance a centralized SRPT scheduler to include the above locality heuristic, it gains no more than 20% compared to centralized SRPT (without the locality heuristic). This indicates that Hopper's gains are predominantly due to coordinated speculation and scheduling.

3.8 Concluding Remarks

With launching speculative copies of tasks being a common approach for mitigating the impact of stragglers, schedulers face a decision between scheduling speculative copies of some jobs versus original copies of other jobs. While this question is seemingly simple, we find that the problem is not only unsolved thus far, but also has significant performance implications.

This chapter proposes Hopper, the first speculation-aware job scheduler, and im-

plements both decentralized and centralized prototypes. We deploy our prototypes (built in Sparrow [20], Spark [52] and Hadoop [4]) on a 200 machine cluster, and see job speedups of 66% in decentralized settings and 50% in centralized settings compared to current state-of-the-art schedulers. In addition to providing performance improvements in both centralized and decentralized settings, Hopper is compatible with all current speculation algorithms and incorporates data locality, fairness, DAGs of tasks, etc.; thus, it represents a unified speculation-aware scheduling framework.

CHAPTER 4

NETWORK-AWARE GEO-DISTRIBUTED CLUSTER SCHEDULING

In the previous two chapters, we provide solutions to straggler speculation-aware scheduling design at both the task level and the job level. In this chapter, we switch our focus from a single data center to geo-distributed data centers and focus on optimizing data acquisition and data placement.

Ten years ago computing infrastructure was a *commodity* – the key bottleneck for new tech startups was the cost of acquiring and scaling computational power as they grew. Now, computing power and memory are *services* that can be cheaply subscribed to and scaled as needed via cloud providers like Amazon EC2, Microsoft Azure, etc.

We are beginning the same transition with respect to *data*. Data is broadly being gathered, bought, processed and sold in various marketplaces. However, it is still a commodity, often obtained through offline negotiations between providers and companies [101]. Thus, acquiring data is one of the key bottlenecks for new tech startups nowadays.

This is beginning to change with the emergence of *cloud data markets (data cloud)*, which offer a single, logically centralized point for selling, buying and processing data. Multiple data markets have recently emerged in the cloud, e.g., Qlik Datamarket [102], Factual [103], InfoChimps [104], Xignite [105], IUPHAR [106], etc. These marketplaces enable data providers to sell and upload data and clients to request data from multiple providers (often for a fee) through a unified query interface. Also, major cloud service providers such as Google [107], Microsoft [108] and Amazon [109] all host various public datasets covering geospacial, environmental, scientific and online services as an extra benefit for their cloud clients. Current data clouds provide a variety of basic services: (i) *aggregation* of data from multiple sources, (ii) *cleaning* of data to ensure quality across sources, (iii) *ease of*

use, through a unified API, and (iv) *low-latency delivery* through a geographically distributed content distribution network. As these market places mature they are increasingly adding other services as well. Besides providing raw data to clients, it is an inevitable trend for data clouds to carry out value-added services built upon the data, such as analytics and machine learning APIs.

Given the recent emergence of data clouds, there are widely differing designs in the marketplace today, especially with respect to pricing. For example, the Qlik Datamarket [102] sets prices with a subscription model that allows a maximum number of queries (API calls) per month and limits the size of records that can be returned for a single query. Other data clouds, e.g., Google BigQuery [107] and Infochimps [104], allow payments per query or per data set. In nearly all cases, the data provider and the data cloud operator each then get a share of the fees paid by the clients, though how this share is arrived at can differ dramatically across data clouds. The task of pricing is made even more challenging when one considers that clients may be interested in data with differing levels of precision/quality and privacy may be a concern.

Not surprisingly, the design of pricing (both on the client side and the data provider side) has received significant attention in recent years, including pricing of per-query access [29, 30] and pricing of private data [31, 32].

In contrast, the focus of this chapter is *not* on the design of pricing strategies for data clouds. Instead, *we focus on the engineering side of the design of a data cloud*, which has been ignored to this point. Supposing that prices are given, there are important challenges that remain for the operation of a data cloud. Specifically, two crucial challenges relate to *data purchasing* and *data placement*.[1]

Data purchasing: Given prices and contracts offered by data providers, which providers should a data cloud purchase from to satisfy a set of client queries with minimal cost?

Data placement: How should purchased data be stored and replicated throughout a geo-distributed data cloud in order to minimize bandwidth and latency costs? And which clients should be served from which replicas given the locations and data requirements of the clients?

[1] Here we assume either the "raw" data is delivered or the computational overhead to process the data is negligible compared to the purchasing cost and placement cost. This is the case for the most existing data clouds [102, 107–109], and also is consistent with literature [27, 28]. We leave joint optimization of computation, data purchasing, and data placement as future work.

Clearly, these two challenges are highly related: data placement decisions depend on which data is purchased from where, so the bandwidth and latency costs incurred because of data placement must be balanced against the purchasing costs. Concretely, less expensive data that results in larger bandwidth and latency costs is not desirable.

The goal of this chapter is to present a design for a geo-distributed data cloud that jointly minimizes data purchasing and data placement costs.

The combination of data purchasing and data placement decisions makes the task of operating a geo-distributed data cloud more complex than the task of operating a geo-distributed data analytics system [26–28, 110, 111]. Geo-analytics systems minimize the cost (in terms of latency and bandwidth) of moving the data needed to answer client queries, replacing the traditional operation mode where data from multiple data centers was moved to a central data center for processing queries. However, crucially, such systems do not consider the cost of obtaining the data (including purchasing and transferring) from data providers.

Thus, the design of a geo-distributed data cloud necessitates integrating data purchasing decisions into a geo-distributed data analytics system. To that end, our design builds on the model used in [28] by adding data providers that offer a menu of data quality levels for differing fees. The data placement/replication problem in [28] is already an integer linear program (ILP), and so it is no surprise that the addition of data providers makes the task of jointly optimizing data purchasing and data placement NP-hard (see Theorem 5).

Consequently, we focus on identifying structure in the problem that can allow for a practical and near-optimal system design. To that end, we show that the task of jointly optimizing data purchasing and data placement is equivalent to the uncapacitated facility location problem (UFLP) [112]. However, while constant-factor polynomial running time approximation algorithms are known for the *metric* uncapacitated facility location problem (see [113–115]), our problem is a *non-metric* facility location problem, and the best known polynomial running time algorithms achieve a $O(\log C)$ approximation via the greedy algorithm in [116] or the randomized rounding algorithm in [117], where C is the number of clients. Note that without any additional information on the costs, this approximation ratio is the smallest achievable for the non-metric uncapacitated facility location unless NP has slightly superpolynomial time algorithms [118]. While this is the best theoretical guarantee possible in the worst-case, some promising heuristics have been proposed for the

non-metric case, e.g., [119–124].

Though the task of jointly optimizing data purchasing and data placement is computationally hard in the worst case, in practical settings there is structure that can be exploited. In particular, we provide an algorithm with polynomial running time that gives an exact solution in the case of a data cloud with a single data center (Section 4.3.1). Then, using this structure, we generalize to the case of a geo-distributed data cloud and provide an algorithm, named Datum (Section 4.3.2) that is near optimal in practical settings.

Datum first optimizes data purchasing as if the data cloud was made up of a single data center (given carefully designed "transformed" costs) and then, given the data purchasing decisions, optimizes data placement/replication. The "transformed" costs are designed to allow an architectural decomposition of the joint problem into subproblems that manage data purchasing (external operations of the data cloud) and data placement (internal operations of the data cloud). This decomposition is of crucial operational importance because it means that internal placement and routing decisions can proceed without factoring in data purchasing costs, mimicking operational structures of geo-distributed analytics systems today.

We provide a case study in Section 4.4 which highlights that Datum is near-optimal (within 1.6%) in practical settings. Further, the performance of Datum improves upon approaches that neglect data purchasing decisions by $> 45\%$.

We initiate the study of jointly optimizing data purchasing and data placement decisions in geo-distributed data clouds. And we prove that the task of jointly optimizing data purchasing and data placement decisions is NP-hard and can be equivalently viewed as a facility location problem. For the case of a data cloud with a single data center, we provide an exact algorithm with polynomial running time. We extend our design to data clouds with multiple data centers and provide an algorithm, Datum, for jointly optimizing data purchasing and data placement in a geo-distributed data cloud that is within 1.6% of optimal in practical settings and improves by $> 45\%$ over designs that neglect data purchasing costs. Importantly, Datum decomposes into subproblems that manage data purchasing and data placement decisions separately.

4.1 Opportunities and Challenges

Data is now a traded *commodity*. It is being bought and sold every day, but most of these transactions still happen offline through direct negotiations for bulk purchases.

This is beginning to change with the emergence of cloud data markets such as Qlik Datamarket [102], Factual [103], InfoChimps [104], Xignite [105]. As cloud data markets become more prominent, data will become a *service* that can be acquired and scaled seamlessly, on demand, similarly to computing resources available today in the cloud.

4.1.1 The potential of data markets

The emergence of cloud data markets has the potential to be a significant disruptor for the tech industry, and beyond. Today, since computing resources can be easily obtained and scaled through cloud services, data acquisition has become the bottleneck for new tech startups.

For example, consider an emerging potential competitor for Yelp. The biggest development challenge is not algorithmic or computational. Instead, it is obtaining and managing high quality data at scale. The existence of a data market with detailed local information about restaurants, attractions, etc., would eliminate this bottleneck entirely. In fact, data markets such as Factual [103] are emerging to target exactly this need.

Another example highlighted in [30, 125] is language translation. Emerging data markets such as Infochimps sell access to data on word translation, word frequency, etc. across languages. This access is a crucial tool for easing the transition tech startups face when moving into different cultural markets.

A final example considers computer vision. When tech startups need to develop computer vision tools in house, a significant bottleneck (in terms of time and cost) is obtaining labeled images with which to train new algorithms. Emerging data markets have the potential to eliminate this bottleneck too. For example, the emerging Visipedia project [126] (while free for now) provides an example of the potential of such a data market.

Thus, like in the case of cloud computing, ease of access and scaling, combined with the cost efficiency that comes with size, implies that cloud data markets have the potential to eliminate one of the major bottlenecks for tech startups today – data acquisition.

4.1.2 Operational challenges for data markets

While data pricing within cloud data markets has received increasing attention, the engineering of the system itself has been ignored. The engineering of such a geo-distributed "data cloud" is complex. In particular, the system must jointly make both data purchasing decisions and data placement, replication and delivery decisions, as described in the introduction.

Even considered independently, the task of optimizing data placement/replication within a geo-distributed data analytics system is challenging. Such systems aim to allow queries on databases that are stored across data centers, as opposed to traditional databases that are stored within a single data center. Examples include Google Spanner [127], Mesa [128], JetStream [129], Geode [28], and Iridium [26]. The aim in designing a geo-distributed data analytics system is to distribute the computation needed to answer queries across data centers; thus avoiding the need to transfer all the data to a single data center to respond to queries. This distribution of computation is crucial for minimizing bandwidth and latency costs, but leads to considerable engineering challenges, e.g., handling replication constraints due for fault tolerance and regulatory constraints on data placement due to data privacy. See [26, 28] for a longer discussion of these challenges and for examples illustrating the benefit of distributed query computation in geo-distributed data analytics systems.

Importantly, all previous work on geo-distributed analytics systems assumes that the system already owns the data. Thus, on top of the complexity in geo-distributed analytics systems, a geo-distributed cloud data market must balance the cost of data purchasing with the impact on data placement/replication costs as well as the decisions for data delivery. For example, if clients who are interested in some data are located close to data center A, while the data provider is located close to data center B (far from data center A), it may be worth it to place that data in data center A rather than data center B. In practice, the problem is more complex since clients are usually geographically distributed rather than centralized and one client may require data from several different data providers.

Additional complexity is created by versioning the data, i.e., the fact that clients have differing quality requirements for the data requested. For example, if some clients are interested in high quality data and others are interested in low quality data, then it may be worth it to provide high quality level data to some clients that only need low quality data (thus incurring a higher price) because of the savings in

bandwidth and replication costs that result from being able to serve multiple clients with the same data.

4.1.3 Related work

Our work focuses on the joint design of data purchasing and data placement in a geo-distributed cloud data market. As such, it is related to recent work on data pricing, content distribution networks, and geo-distributed data analytics systems. Further, the algorithmic problem at the core of our design is the facility location problem, and so our work builds on that literature. We discuss related work in these areas.

Data pricing: The design of data markets has attracted increasing interest in recent years, especially in the database community, see [130] for an overview. The current literature mainly focuses on query-based pricing mechanism designs [29, 30, 32] and seldom considers the operating cost of the market service providers (i.e., the data cloud). There is also a growing body of work related to data pricing with differentiated qualities [31, 32, 131], often motivated by privacy.

Similarly, in [132, 133] a data pricing scheme is proposed for XML trees in which data prices vary with data quality. Quality is measured in different ways, in the form of accuracy [132], or completeness [133]. In these works, data providers offer prices, but clients may propose a price less than that of the data provider, after which the quality of the data will be decreased. Their framework is based on uniform sampling of rooted subtrees in weighted XML documents. Additionally, [134–136] consider issues related to pricing strategies and data quality, and allow customers to suggest prices rather than imposing fixed prices. They consider the *Name Your Own Price* mechanism [135], which is also used in earlier work [133]. We however allow data providers to set the price, and also allow for clients to view available quality levels and prices before making a query. This work relates to data pricing on the data provider side and is orthogonal to our discussion in this chapter. We assume that prices are known, and are instead concerned with data purchasing and placement choices.

Geo-distributed data analytics systems: As cloud servers are increasingly located in geo-distributed systems, analysis and optimization of data stored in geographically distributed data centers has received increasing attention [26–28, 137, 138]. Bandwidth constraints [27, 28] as well as latency [26] are the two main challenges for system design, and a number of system designs have been proposed, e.g., see Sec-

tion 4.1.2 for more discussion. Our work builds on the model of geo-distributed data analytics systems in [26, 28], but is distinct from this literature because none of the work on geo-distributed data analytics systems considers the costs associated with purchasing data.

Content Distribution Networks: Content Distribution Networks (CDNs) [139–142] were originally introduced to improve quality of Internet services to meet the challenge of rapid growth in the intensity of the content. CDNs replicate content and data from source servers to many other servers that are located closer to the end customers; therefore making it possible to process requests for content locally, saving bandwidth and reducing latency [143–152]. As such, the algorithms governing CDNs are similar in spirit to those underlying data markets. However, the novelty of data cloud design stems from the fact that it is not enough to replicate content on a network of data centers so as to minimize the cost of delivering the data from sources to customers, data clouds must also consider the fact that there is no free access to the data – it must be *purchased* from providers. This cost significantly changes the algorithmic challenge and leads us to study a framework to *simultaneously* decide (i) what data to *purchase* from which providers and (ii) where to *place* the data on the network and how to route the data to the clients in order to minimize its operating costs.

Algorithms for facility location: Our data cloud cost minimization problem can be viewed as a variant of the uncapacitated facility location problem. Though such problems have been widely studied, most of the results, especially algorithms with constant approximation ratios, require the assumption of metric cost parameters [113–115], which is not the case in our problem. In contrast, for the non-metric facility location problem the best known algorithm is the greedy algorithm proposed in [112]. Beyond this algorithm, a variety of heuristics for solving UPFL have been proposed. However, (i) the usual UPFL heuristics do not take into account the added structure of our problem, and (ii) the new heuristic that we propose, Datum, shows that you can in fact simplify the intermediary's problem by separating purchasing and placement decisions and still provide near-optimal solutions in practical settings. The usual UPFL heuristics do not exhibit such a decomposition. Datum may also be valuable more broadly for facility location problems.

4.2 A Geo-Distributed Data Cloud

This chapter presents a design for a geo-distributed cloud data market, which we refer to as a “data cloud." This data cloud serves as an intermediary between *data providers*, which gather data and offer it for sale, and *clients*, which interact with the data cloud through queries for particular subsets/qualities of data. More concretely, the data cloud purchases data from multiple data providers, aggregates it, cleans it, stores it (across multiple geographically distributed data centers), and delivers it (with low-latency) to clients in response to queries, while aiming at minimizing the *operational cost* which constitutes of both bandwidth and data purchasing costs.

Our design builds on and extends the contributions of recent papers – specifically [26, 28] – that have focused on building geo-distributed data analytic systems but assume the data is already owned by the system and focus solely on the interaction between a data cloud and its clients. Unfortunately, as we highlight in Section 4.3, the inclusion of data providers means that the data cloud’s goal of cost minimization can be viewed as a non-metric uncapacitated facility location problem, which is NP-hard. For ease of exposition, we keep our model simple. We note that, however, it can be easily adapted to include various pricing mechanisms on the data provider side, different query structures and execution plans in the data cloud, as well as different types of contracts and payment methods between data cloud and clients.

For reference, Figure 4.1 provides an overview of the interaction between these three parties as well as some basic notations.

4.2.1 Modeling Data Providers

The interaction between the data cloud and data providers is a key distinction between the setting we consider and previous work on geo-distributed data analytics systems such as [26, 28]. We assume that each data provider offers distinct data to the data cloud, and that the data cloud is a price-taker, i.e., cannot impact the prices offered by data providers. Thus, we can summarize the interaction of a data provider with the data cloud through an exogenous menu of data qualities and corresponding prices.

We interpret the quality of data as a general concept that can be instantiated in multiple ways. For categorical data, quality may represent the resolution of the information provided, e.g., for geographical attributes the resolution may be {street address, zip code, city, county, state}. For numerical data, quality could take many

forms, e.g., the numerical precision, the statistical precision (e.g., the confidence of an estimator), or the level of noise added to the data.[2]

Concretely, we consider a setting where there are P data providers selling different data, *provider* $p \in \mathcal{P} = \{1, 2, \ldots, \mathcal{P}\}$.[3] Each data provider offers a set of quality levels, indexed by *level* $l \in \mathcal{L} = \{1, 2, \ldots, L_p\}$, where L_p is the number of levels that data provider p offers. We use $q(l, p)$ to denote the data quality level l, offered by data provider p. Similarly, we use $f(l, p)$ to denote the fee charged by data provider p for data of quality level l. Importantly, the prices vary across providers p since different providers have different procurement costs for different qualities and different data.

The data purchasing contract between data providers and the data cloud may have a variety of different types. For example, a data cloud may pay a data provider based on usage, i.e., per query, or a data cloud may buy the data in bulk in advance. In this chapter, we discuss both per-query data contracting and bulk data contracting. See Section 4.2.3.1 for details.

4.2.2 Modeling Clients

Clients interact with the data cloud through queries, which may require data (with varying quality levels) from multiple data providers.

Concretely, we consider a setting where there are C clients, *client* $c \in C = \{1, 2, \ldots, C\}$. A client c sends a *query* to the data center, requesting particular data from multiple data providers.[4] Denote the set of data providers required by the request from client query c by $G(c)$. The client query also specifies a minimum desired quality level, $w_c(p)$, for each data provider p it requests, i.e., $\forall p \in G(c)$. We assume that the client is satisfied with data at a quality level higher than or equal to the level requested.

More general models of queries are possible, e.g., by including a DAG modeling the structure of the query and query execution planning (see [28] for details). For ease of exposition, we do not include such detailed structure here, but it can be added at the expense of more complicated notation.

[2] A common suggestion for guaranteeing privacy is to add Laplace noise to data provided to data markets, see e.g., [32, 153].

[3] We distinguish data providers based on data, i.e., one data provider that sells multiple data is treated as multiple data providers.

[4] We distinguish clients based on queries, i.e., one client that sends multiple queries is treated as multiple clients.

Depending on the situation, the client may or may not be expected to pay the data cloud for access. If the clients are internal to the company running the data cloud, client payments are unnecessary. However, in many situations the client is expected to pay the data cloud for access to the data. There are many different types of payment structures that could be considered. Broadly, these fall into two categories: (i) subscription-based (e.g., Azure DataMarket [154]) or (ii) per-query-based (e.g. Infochimps [104]).

In this chapter, we do not focus on (or model) the design of payment structure between the clients and the data cloud. Instead, we focus on the operational task of minimizing the cost of the data cloud operation (i.e., bandwidth and data purchasing costs). This focus is motivated by the fact that minimizing the operation costs improves the profit of the data cloud regardless of how clients are charged. Interested readers can find analyses of the design of client pricing strategies in [29, 30, 32].

4.2.3 Modeling a Geo-Distributed Data Cloud

The role of the data cloud in this marketplace is as an aggregator and intermediary. We model the data cloud as a geographically distributed cloud consisting of D data centers, *data center* $d \in \mathcal{D} = \{1, 2, \ldots, D\}$. Each data center aggregates data from geographically separate local data providers, and data from data providers may be (and often is) replicated across multiple data centers within the data cloud.

Note that, even for the same data with the same quality, data transfer from the data providers to the data cloud is not a one time event due to the need of the data providers to update the data over time. Here we consider a model where clients are enterprises and clients and the data cloud sign on contracts in advance to decide on which data and what data quality is to be transferred to the clients. Thus we target the modeling and optimization of the data cloud within a fixed time horizon, given the assumption that queries from clients are known beforehand or can be predicted accurately. This assumption is consistent with previous work [26, 28] and reports from other organizations [155, 156]. Online versions of the problem are also of interest, but are not the focus of this chapter.

4.2.3.1 Modeling costs

Our goal is to provide a design that minimizes the operational costs of a data cloud. These costs include both data purchasing and bandwidth costs. In order to describe

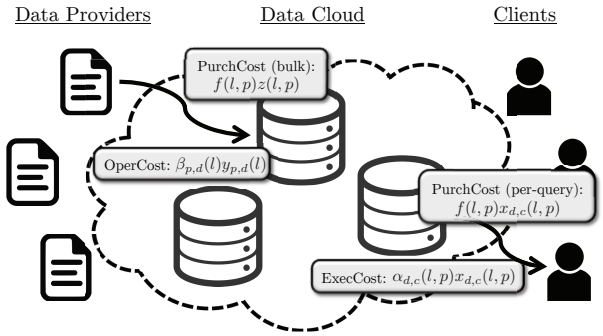


Figure 4.1: An overview of the interaction between data providers, the data cloud, and clients. The dotted line encircling the data centers (DC) represents the geo-distributed data cloud. Data providers and clients interact only with the cloud. Data provider p sends data of quality $q(l,p)$ to data center d, and the corresponding operation cost is $\beta_{p,d}(l)y_{p,d}(l)$. Similarly, data center d sends data of quality $q(l,p)$ to client c, and the corresponding execution cost is $\alpha_{d,c}(l,p)x_{d,c}(l,p)$. In bulk data contracting, the corresponding purchasing cost is $f(l,p)z(l,p)$. In per-query data contracting, the corresponding purchasing cost is $f(l,p)x_{d,c}(l,p)$.

these costs, we use the following notation, which is summarized in Figure 4.1.[5]

$x_{d,c}(l,p) \in \{0,1\}$: $x_{d,c}(l,p) = 1$ if and only if data of quality $q(l,p)$, originating from data provider p, is transferred from data center d to client c.

$\alpha_{d,c}(l,p)$: cost (including bandwidth and/or latency) to transfer data of quality $q(l,p)$, originating from data provider p, from data center d to client c

$y_{p,d}(l) \in \{0,1\}$: $y_{p,d}(l) = 1$ if and only if data of quality $q(l,p)$ is transferred from data provider p to data center d.

$\beta_{p,d}(l)$: cost (including bandwidth and/or latency) to transfer data of quality $q(l,p)$ from data provider p to data center d.

$z(l,p) \in \{0,1\}$: $z(l,p) = 1$ if and only if data of quality $q(l,p)$, originating from data provider p, is transferred to the data cloud.

$f(l,p)$: purchasing cost of data with quality $q(l,p)$, originating from data provider p.

[5]Throughout, subscript indices refer to data transfer "from, to" a location, and parenthesized indices refer to data characteristics (e.g., quality, from which data provider).

Given the above notations, the costs of the data cloud can be broken into three categories:

(i) The *operation cost* due to transferring data of all quality levels from data providers to data centers is

$$\text{OperCost} = \sum_{p=1}^{P} \sum_{l=1}^{L_p} \sum_{d=1}^{D} \beta_{p,d}(l) y_{p,d}(l). \tag{4.1}$$

(ii) The *execution cost* due to transferring data of all quality levels from data centers to clients is

$$\text{ExecCost} = \sum_{c=1}^{C} \sum_{p \in G(c)} \sum_{l=1}^{L_p} \sum_{d=1}^{D} \alpha_{d,c}(l,p) x_{d,c}(l,p). \tag{4.2}$$

(iii) The *purchasing cost* (PurchCost) due to buying data from the data provider could result from a variety of differing contract styles. In this chapter we consider two extreme options: *per-query* and *bulk* data contracting. These are the most commonly adopted strategies for data purchasing today.

In *per-query* data contracting, the data provider charges the data cloud a fixed rate for each query that uses the data provided by the data provider. So, if the same data is used for two different queries, then the data cloud pays the data provider twice. Given a per-query fee $f(l,p)$ for data $q(l,p)$, the total purchasing cost is

$$\text{PurchCost(query)} = \sum_{c=1}^{C} \sum_{p \in G(c)} \sum_{l=1}^{L_p} \sum_{d=1}^{D} f(l,p) x_{d,c}(l,p). \tag{4.3}$$

In *bulk* data contracting, the data cloud purchases the data in bulk and then can distribute it without owing future payments to the data provider. Given a one-time fee $f(l,p)$ for data $q(l,p)$, the total purchasing cost is

$$\text{PurchCost(bulk)} = \sum_{p=1}^{P} \sum_{l=1}^{L_p} f(l,p) z(l,p). \tag{4.4}$$

To keep the presentation of this chapter simple, we focus on the per-query data contracting model throughout the body of the chapter and discuss the bulk data contracting model (which is simpler) in Appendix 4.A.

4.2.3.2 Cost Optimization

Given the cost models described above, we can now represent the goal of the data cloud via the following integer linear program (ILP), where OperCost, ExecCost, and PurchCost are as described in equations (4.1), (4.2) and (4.3), respectively.

$$\min_{x,y} \quad \text{OperCost} + \text{ExecCost} + \text{PurchCost} \tag{4.5}$$

$$\text{subject to} \quad x_{d,c}(l,p) \leq y_{p,d}(l) \ \forall c,p,l,d \tag{4.5a}$$

$$\sum_{l=1}^{L_p} \sum_{d=1}^{D} x_{d,c}(l,p) = 1, \ \forall c, p \in G(c) \tag{4.5b}$$

$$\sum_{l=1}^{L_p} \sum_{d=1}^{D} x_{d,c}(l,p) q(l,p) \geq w_c(p), \ \forall c, p \in G(c) \tag{4.5c}$$

$$x_{d,c}(l,p) \geq 0, \forall c,p,l,d \tag{4.5d}$$

$$y_{p,d}(l) \geq 0, \ \forall p,l,d \tag{4.5e}$$

$$x_{d,c}(l,p), y_{p,d}(l) \in \{0,1\}, \forall c,p,l,d \tag{4.5f}$$

The constraints in this formulation warrant some discussion. Constraint (4.5a) states that any data transferred to some client must already have been transferred from its data provider to the data cloud.[6] Constraint (4.5b) ensures that each client must get the data it requested, and constraint (4.5c) ensures that the minimum quality requirement of each client must be satisfied. The remaining constraints state that the decision variables are binary and nonnegative.

An important observation about the formulation above is that data purchasing/placement decisions are decoupled across data providers, i.e., the data purchasing/placement decision for data from one data provider does not impact the data purchasing/placement decision for any other data providers. Thus, we frequently drop the index p.

Note that there is a variety of practical issues that we have not incorporated into the formulation in (4.5) in order to minimize notational complexity, but which can be included without affecting the results described in the following. A first example is that a minimal level of data replication is often desired for fault tolerance and disaster recovery reasons. This can be added to (4.5) by additionally considering constraints of the form $\sum_{d=1}^{D} y_{p,d}(l) \geq k z(l,p)$, where k denotes the minimum required number of copies. Similarly, privacy concerns often lead to regulatory constraints on data

[6] For bulk data contracting model, one more constraint $y_{p,d}(l) \leq z(l,p), \ \forall c,l,p,d$ is required. This constraint states that any data placed in the data cloud must be purchased by the data cloud.

movement. As a result, regulatory restrictions may prohibit some data from being copied to certain data centers, thus constraining data placement and replication. This can be included by adding constraints of the form $y_{p,d}(l) = 0$ to (4.5) where p and d denote the corresponding data provider and data center, respectively. Finally, in some cases it is desirable to enforce SLA constraints on the latency of delivery to clients. Such constraints can be added by including constraints of the form $\sum_{p \in G(c)} \sum_{l=1}^{L_p} \sum_{d=1}^{D} \alpha_{d,c}(l,p) x_{d,c}(l,p) \leq r_c$, where r_c denotes the SLA requirement of client c.

We refer the reader to [26–28] for more discussions of these additional practical constraints. Each paper includes a subset of these factors in the design of geo-distributed data analytics systems, but does not model data purchasing decisions.

4.3 Optimal Data Purchasing and Data Placement

Given the model of a geo-distributed data cloud described in the previous section, the design task is now to provide an algorithm for computing the optimal data purchasing and data placement/replication decisions, i.e., to solve data cloud cost minimization problem in (4.5). Unfortunately, this cost minimization problem is an ILP, which are computationally difficult in general.[7]

A classic NP-hard ILP is the uncapacitated facility location problem (UFLP) [112]. In the uncapacitated facility location problem, there is a set of I clients and J potential facilities. Facility $j \in J$ costs f_j to open and can serve clients $i \in I$ with cost $c_{i,j}$. The task is to determine the set of facilities that serves the clients with minimal cost.

Our first result, stated below, highlights that cost minimization for a geo-distributed data cloud can be reduced to the uncapacitated facility location problem, and vice-versa. Thus, the task of operating a data cloud can then be viewed as a facility location problem, where opening a facility parallels purchasing a specific quality level from a data provider and placing it in a particular data center in the data cloud.

Theorem 5. *The cost minimization problem for a geo-distributed data cloud given in* (4.5) *is NP-hard.*

[7] Note that previous work on geo-distributed data analytics where data providers and data purchasing were not considered already leads to an ILP with limited structure. For example, [28] suggest only heuristic algorithms with no analytic guarantees.

Proof. To prove Theorem 5, we show a connection between the data cloud cost minimization problem in (4.5) and the uncapacitated facility location problem. In particular, we show both that the facility location problem can be reduced to a data cloud optimization problem and vice versa.

First, we show that every instance of the uncapacitated facility location problem can be viewed as an instance of (4.5).

Take any instance of the uncapacitated facility location problem (UFLP). Let I be the set of customers, J the set of locations, α_{ij} the cost of assigning customer i to location j, and β_j the cost of opening facility j. Binary variables $y_j = 1$ if and only if facility is open at site j, and $x_{j,i} = 1$ if and only if customer i is assigned to location j. Then the UFLP can be formulated as following.

$$\min_{x,y} \quad \sum_{j \in F} \beta_j y_j + \sum_{i \in I, j \in F} \alpha_{ij} x_{j,i} \tag{4.6}$$

subject to

$$x_{j,i} \leq y_j, \ \forall i, j$$
$$\sum_{j \in F} x_{j,i} = 1, \ \forall c$$
$$x_{j,i}, y_j \in \{0, 1\}, \forall i, j$$

Mapping j to d and i to c yields an instance of (4.5) with $|P| = |L| = 1$, $f(l) = 0$ and $w_c(l) = 0$, in which case constraint (4.5c) becomes trivial.

Next, we show that every instance of (4.5) can be written as an instance of UFLP.

We start by remarking that (4.5) (with p dropped,)[8]

$$\min_{x,y} \quad \sum_{d,l=1}^{D,L} \beta_d(l) y_d(l) + \sum_{d,l,c=1}^{D,L,C} (f(l) + \alpha_{d,c}(l)) x_{d,c}(l) \tag{4.7}$$

subject to

$$x_{d,c}(l) \leq y_d(l), \ \forall c, l, d$$
$$\sum_{d=1}^{D} \sum_{l=1}^{L} x_{d,c}(l) = 1, \ \forall c$$

[8]Recall from the discussion in Section 4.2.3.2 that the program is separable per provider, hence we can without loss of generality consider the problem for a single provider, and drop the p index in the notation.

$$x_{d,c}(l), y_d(l) \in \{0,1\}, \forall c, l, d$$

with $\alpha_{d,c}(l) = M$, for M big enough, whenever $l < w_c$. Indeed, in any feasible solution of (4.5), we necessarily have $x_{d,c}(l) = 0$ whenever $l < w_c$, as each client purchases exactly one quality level and this quality level has to be higher than the minimum required level w_c; by setting $\alpha_{d,c}(l)$ big enough, we ensure that any optimal solution must have $x_{d,c}(l) = 0$ thus must be feasible for (4.5), and has the same cost as in (4.5). Now, take $J = [D] \times [L]$ and $I = [C]$, and the problem can be rewritten as

$$\begin{aligned} \min_{x,y} \quad & \sum_{(d,l)\in J} \beta_d(l) y_d(l) + \sum_{(d,l)\in J, c\in I} (f(l) + \alpha_{d,c}(l)) x_{d,c}(l) \qquad (4.8) \\ \text{subject to} \quad & x_{d,c}(l) \le y_d(l), \ \forall (d,l) \in J, c \in I \\ & \sum_{(d,l)\in J} x_{d,c}(l) = 1, \ \forall c \in I \\ & x_{d,c}(l), y_d(l) \in \{0,1\}, \forall c \in I, (d,l) \in J \end{aligned}$$

which is an UFLP. □

The proof of Theorem 5 provides a reduction both to and from the uncapacitated facility location problem. Importantly, the proof of Theorem 5 serves a dual purpose: it both characterizes the hardness of the data cloud cost minimization problem and highlights that algorithms for the facility location problem can be applied in this context. Given the extensive literature on facility location, this is important.

More specifically, the reduction leading to Theorem 5 highlights that the data cloud optimization problem is equivalent to the *non-metric* uncapacitated facility location problem – every instance of any of the two problems can be written as an instance of the other. While constant-factor polynomial running time approximation algorithms are given for the *metric* uncapacitated facility location problem in [113–115], in the more general *non-metric* case the best known polynomial running time algorithm achieves a $\log(C)$-approximation via a greedy algorithm with polynomial running time, where C is the number of clients [116]. This is the best worst-case guarantee possible (unless NP has slightly superpolynomial time algorithms, as proven in [118]); however some promising heuristics have been proposed for the non-metric case, e.g., [119–124].

Nevertheless, even though our problem can, in general, be viewed as the non-metric uncapacitated facility location, it does have a structure in real-world situations that we can exploit to develop practical algorithms.

In particular, in this section we begin with the case of a data cloud made up of a single data center. We show that, in this case, there is a structure that allows us to design an algorithm with polynomial running time that gives an exact solution (Section 4.3.1). Then, we move to the case of a data cloud made up of geo-distributed data centers and highlight how to build on the algorithm for the single data center case to provide an algorithm, Datum, for the general case (Section 4.3.2). Importantly, Datum allows decomposition of the management of data purchasing (operations outside of the data cloud) and data placement (operations inside the data cloud). This feature of Datum is crucial in practice because it means that the algorithm allows a data cloud to manage internal operations without factoring in data purchasing costs, mimicking operations today. While we do not provide analytic guarantees for Datum (as expected given the reduction to/from the non-metric facility location problem), we show that the heuristic performs well in practical settings using a case study in Section 4.4.

4.3.1 An exact solution for a single data center

We begin our analysis by focusing on the case of a single data center, which interacts with multiple data providers and multiple clients. The key observation is that, if the execution costs associated with transferring different quality levels of the same data are the same, i.e., $\forall l, \alpha_c(l) = \alpha_c$, then the execution cost becomes a constant which is independent of the data purchasing and data placement decisions as shown in (4.9).

$$\text{ExecCost} = \sum_{c=1}^{C} \sum_{l=1}^{L} \alpha_c x_c(l) = \sum_{c=1}^{C} \alpha_c \left(\sum_{l=1}^{L} x_c(l) \right) = \sum_{c=1}^{C} \alpha_c \tag{4.9}$$

The assumption that the execution costs are the same across quality levels is natural in many cases. For example, if quality levels correspond to the level of noise added to numerical data, then the size of the data sets will be the same. We adopt this assumption in Section 4.3.1 and extend to the general case in Section 4.3.2.

This assumption allows the elimination of the execution cost term from the objective. Additionally, we can simplify notation by removing the index d for the data center. Thus, in per-query data contracting, the data cloud optimization problem can be simplified to (4.10). (We discuss the case of bulk data contracting in Appendix 4.A.)

$$\text{minimize} \quad \sum_{l=1}^{L} \beta(l) y(l) + \sum_{c=1}^{C} \sum_{l=1}^{L} f(l) x_c(l) \tag{4.10}$$

$$
\begin{aligned}
\text{subject to} \quad & x_c(l) \leq y(l), \ \forall c, l \\
& \sum_{l=w_c}^{L} x_c(l) = 1, \ \forall c \qquad (4.10a) \\
& x_c(l) \geq 0, \forall c, l \\
& y(l) \geq 0, \ \forall l \\
& x_c(l), y(l) \in \{0, 1\}, \forall c, l
\end{aligned}
$$

Note that constraint (4.10a) is a contraction of (4.5b) and (4.5c), and simply means that any client c must be given exactly one quality level above w_c, the minimum required quality level.[9] The remaining constraints follow directly from (4.5) by dropping d since we only consider one data center case in (4.10). While this problem is still an ILP, in this case there is a structure that can be exploited to provide a polynomial time algorithm that can find an exact solution.

Theorem 6. *There exists a binary optimal solution to the linear relaxation program in* (4.11) *which is an optimal solution of the integer program in* (4.10) *and can be found in polynomial time.*

Proof. Assume without loss of generality that all clients can be satisfied by the highest quality level, i.e., $w_c \leq q(L), \forall c$. Define $C_i = \{c : q(i-1) < w_c \leq q(i)\}$ (q(0) = 0 by default). Given these assumptions, clients can be grouped into L categories $\{C_1, C_2, \ldots, C_L\}$ based on their minimum quality level. Note that $C_i \cap C_j = \emptyset, \forall i, j$ and $\cup_{i=1}^{L} C_i = C$. Without loss of generality, assume $C_i \neq \emptyset, \ \forall i$.

As the clients in the same group C_i all face exactly the same choice of quality levels and minimum quality requirements, there must always be an optimal solution in which the data purchasing decisions of any clients within one category are the same.

Let us denote the number of clients in category C_i by S_i. Denote the purchasing decision of category C_i by χ_i, e.g., $\chi_i(l) = x_c(l), \ \forall l, c \in C_i$, similar to the argument in the proof of Theorem 5, and we can reformulate (4.10) as follows. Note the slight abuse of notation, as clients and their associated required quality level are represented by the same letter, i, due to clients in category C_i having minimum

[9] While the two constraints are equivalent for an ILP, they lead to different feasible sets when considering its LP-relaxation; in particular, facility location algorithms based on LP-relaxations such as randomized rounding algorithms need to use the contracted version of the constraints to preserve the $O(\log C)$-approximation ratio for non-metric facility location. It is equivalent to the reformulation given in (4.8) and does not introduce infinite costs that may lead to numerical errors.

quality level i by definition.

$$\text{minimize} \quad \sum_{l=1}^{L} \beta(l)y(l) + \sum_{i=1}^{L}\sum_{l=i}^{L} S_i f(l)\chi_i(l) \tag{4.11}$$

$$\text{subject to} \quad \chi_i(l) \le y(l), \ \forall i, l \tag{4.11a}$$

$$\sum_{l=i}^{L} \chi_i(l) = 1, \ \forall i \tag{4.11b}$$

$$\chi_i(l) \ge 0, \forall i, l \tag{4.11c}$$

$$y(l) \ge 0, \ \forall l \tag{4.11d}$$

$$\chi_i(l), y(l) \in \{0, 1\}, \forall i, l \tag{4.11e}$$

Consider the linear relaxation of (4.11), which drops the $0-1$ integer constraint (4.11e). For any optimal solution $\{\chi_i^r(l), y^r(l)\}$ of the linear relaxation we have the following observations.

1. $\chi_L^r(L) = 1$.

   Proof. From (4.11b), let $i = L$, then $\chi_L^r(L) = 1$. The intuition behind this is that, since $C_L \neq \emptyset$, highest quality data always has to be purchased to provide service for clients in $C(L)$. □

2. $y^r(l) = \max_i\{\chi_i^r(l)\} \in [0, 1]$ and $y^r(L) = 1$.

   Proof. From (4.11a), the non-negativity of $\{\beta(l)\}$, and the optimality of $\{y^r(l)\}$, $y^r(l) = \max_i\{\chi_i^r(l)\}$. From the non-negativity of $\{\chi_i^r(l)\}$, $y^r(l) = \max_i\{\chi_i^r(l)\} \le \sum_{l=i}^{L} \chi_i^r(l) = 1$, and $y^r(L) = \chi_L^r(L) = 1$ □

3. $\forall l \ge i$, if $\sum_{l=i}^{L} y^r(l) \le 1$, $\chi_i^r(l) = y^r(l)$; otherwise, $\chi_i(l) = \max\{1-\sum_{k=i}^{l-1} y^r(k), 0\}$.

   Proof. For some fixed i, $\{S_i f(l)\}$ is a positive, strictly increasing sequence as l increases. From constraint (4.11a) and (4.11b), $\chi_i^r(l) \le y^r(l)$, and $\sum_{l=i}^{L} \chi_i^r(l) = 1$. Since $\{\chi_i^r(l), y^r(l)\}$ is optimal, $\forall l \ge i$, if $\sum_{k=i}^{l} y^r(k) \le 1$, $\chi_i^r(l) = y^r(l)$; otherwise, $\chi_i^r(l) = \max\{1 - \sum_{k=i}^{l-1} y^r(k), 0\}$. □

Next, define $m_i \in \{i, \ldots, n\}$ such that $\sum_{l=i}^{m_i-1} y^r(l) < 1$, and $\sum_{l=i}^{m_i} y^r(l) \geq 1$. Such an m_i must exist since $y^r(l) \geq 0$ for all l and $y^r(L) = 1$. Recall $\chi_L^r(L) = y^r(L) = 1$. For for any $i = 1, 2, \ldots, L-1$, if the values of $\{y^r(l)\}$ are given, the optimal $\{\chi_i^r(l)\}$ satisfy the following closed form expression:

$$\chi_i^r(l) = \begin{cases} y^r(l), & i \leq l < m_i \\ 1 - \sum_{k=i}^{m_i-1} y^r(k), & l = m_i \\ 0, & m_i < l \leq n. \end{cases} \tag{4.12}$$

Note that, if y^r are binary, then χ^r are binary. Suppose there exists an optimal solution $\{\chi^r, y^r\}$ with $y^r \notin \{0,1\}^L$, in the following we show that there exists a feasible binary solution $\{\chi^*, y^*\}$ of (4.11) such that the objective value generated by $\{\chi^*, y^*\}$ is better than or equal to that of $\{\chi^r, y^r\}$.

Suppose fractional solution y^r is an optimal solution of the linear relaxation and calculate m_i as in (4.12). Write χ as a function of y, $\forall i, l$.

$$\chi_i(l) = \begin{cases} y(l), & i \leq l < m_i \\ 1 - \sum_{k=i}^{m_i-1} y(k), & l = m_i \\ 0, & m_i < l \leq n \end{cases} \tag{4.13}$$

Substituting (4.13) in the objective function (4.11), the objective function becomes a linear combination of $\{y(l)\}$ that we denote $L(y)$.

Consider the optimization problem in which $\{\chi_i(l)\}$ is expressed as a function of $\{y(l)\}$ in the linear relaxation:

$$\begin{aligned} \text{minimize} \quad & L(y) \\ \text{subject to} \quad & \sum_{l=i}^{m_i'-1} y(l) \leq 1, \ \forall i = 1, \ldots, L-1 \\ & \sum_{l=i}^{m_i'} y(l) \geq 1, \ \forall i = 1, \ldots, L-1 \\ & y(l) \geq 0, \ \forall l = 1, \ldots, L \\ & y(L) = 1 \end{aligned} \tag{4.14}$$

The following claims hold:

1. (4.14) is feasible and bounded, and always has an optimal solution at an extreme point.

Proof. Clearly, $\forall l, y(l) \in [0, 1]$. And starting from $y(L)$, it is easy to construct a feasible solution of (4.14). Thus, (4.14) is feasible and bounded, and always has an optimal solution at an extreme point. □

2. $\{y^r(l)\}$ is a feasible solution of (4.14).

Proof. Since $\{y^r(l)\}$ is feasible for (4.11), $y^r(l) \geq 0, \ \forall l$, and $y^r(L) = 1$. By definition of m_i, $\sum_{l=i}^{m_i-1} y^r(l) \leq 1$, $\sum_{l=i}^{m_i} y^r(l) \geq 1$. □

3. Any extreme point $\{y(l)\}$ of (4.14) is binary.

Proof. Since $y(L) = 1$, we can drop $y(L)$, and write (4.14) in the following standard linear programming form:

$$\begin{aligned} \min_{y} \quad & L(y) \\ \text{s.t.} \quad & Ay \leq b \\ & y \geq 0 \end{aligned} \tag{4.15}$$

Note that all entries of A are $0, \pm 1$, and all rows of A have either consecutive 1s or consecutive -1s. Thus, from [157], A is a totally unimodular matrix thus the extreme points of (4.15) are all integral. In particular, since all $y(l) \in [0, 1]$, the extreme points of (4.15) are all binary. □

4. The $\{\chi_i^*(l)\}$ obtained through (4.13) corresponding to an optimal binary solution $\{y^*\}$ is also binary.

Proof. Follows immediately from (4.13) and integrality of $\{y^*(l)\}$. □

5. $\{\chi_i^*(l), y^*(l)\}$ is a feasible solution of the linear relaxation of (4.11).

Proof. Follows from (4.13) and $\sum_{l=i}^{L} \chi_i^*(l) = 1$. □

$\{\chi_i^r(l), y^r(l)\}$ and any optimal extreme point $\{\chi_i^*(l), y^*(l)\}$ see their corresponding objective values unchanged between (4.14) and the relaxation of (4.11) by construction of the $\chi_i(l)$'s. And any such extremal and optimal $\{\chi_i^*(l), y^*(l)\}$ has a better or equal objective value compared to $\{\chi_i^r(l), y^r(l)\}$ in relaxed (4.11). Since $\{\chi_i^r(l), y^r(l)\}$ is optimal for (4.14), it implies any optimal extreme point of

relaxed (4.11) yields a binary and optimal solution for (4.14). This provides a polynomial time algorithm to find such a binary optimal solution, which can be summarized as in Section 4.3.2.

□

In summary, the following gives a polynomial time algorithm which yields the optimal solution of (4.10).

Step 1: Rewrite (4.10) in the form given by (4.11).

Step 2: Solve the linear relaxation of (4.11). If it gives an integral solution, this solution is an optimal solution of (4.10), and the algorithm finishes. Otherwise, denote the fractional solution of the previous step by $\{\chi^r(l), y^r(l)\}$ and continue to the next step.

Step 3: Find $m_i \in \{i, \ldots, n\}$ such that $\sum_{l=i}^{m_i-1} y^r(l) < 1$, and $\sum_{l=i}^{m_i} y^r(l) \geq 1$. And express $\{\chi_i(l)\}$ as a function of $\{y(l)\}$ based on (4.13). Substitute the expressions of $\{\chi_i(l)\}$ with $\{y(l)\}$ in the linear relaxation of (4.11) to obtain an instance of (4.14). Solve the linear programming problem (4.14) and find an optimal solution that is also an extreme point of (4.14).[10] This yields a binary optimal solution of (4.14). Use transformation (4.13) to get a binary optimal solution of the linear relaxation of (4.11), which can be reformulated as an optimal solution of (4.10) from the definition of $\{\chi_i(l)\}$.

4.3.2 The Design of Datum

Unlike the data cloud cost minimization problem for a single data center, the general data cloud cost minimization is NP-hard. In this section, we build on the exact algorithm for cost minimization in a data cloud made up of a single data center (Section 4.3.1) to provide an algorithm, Datum, for cost minimization in a geo-distributed data cloud.

The idea underlying Datum is to, first, optimize data purchasing decisions as if the data market was made up of a single data center (given carefully designed "transformed" costs), which can be done tractably as a result of Theorem 6. Then, second, Datum optimizes data placement/replication decisions given the data purchasing decisions.

[10] This step can be finished in polynomial time [158].

Before presenting Datum, we need to reformulate the general cost minimization ILP in (4.5). Recall that (4.5) is separable across providers, thus we can consider independent optimizations for each provider, and drop the index p throughout. Second, we denote the set of all possible subsets of data centers, e.g., $\{\{d_1\}, \{d_2\}, \ldots, \{d_1, d_2\}, \{d_1, d_3\}, \ldots\}$ by V.[11] Further, define $\beta_v(l) = \sum_{d \in v} \beta_d(l)$, and $\alpha_{v,c}(l) = \min_{d \in v}\{\alpha_{d,c}(l)\}$, where $v \in V$. Given this change, we define $y_v(l) = 1$ if and only if data with quality level l is placed in (and only in) data centers $d \in v$ and $x_{v,c}(l) = 1$ if and only if data with quality level l is transferred to client c from some data center $d \in v$. These reformulations allow us to convert (4.5) to (4.16) as following.

$$\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L}\sum_{v=1}^{V} \beta_v(l) y_v(l) + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V} \alpha_{v,c}(l) x_{v,c}(l) \\
& + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V} f(l) x_{v,c}(l) && (4.16)\\
\text{subject to} \quad & x_{v,c}(l) \leq y_v(l), \ \forall c, l && (4.16a)\\
& \sum_{l=w_c}^{L}\sum_{v=1}^{V} x_{v,c}(l) = 1, \ \forall c && (4.16b)\\
& \sum_{v=1}^{V} y_v(l) \leq 1, \ \forall l && (4.16c)\\
& \sum_{v=1}^{V} x_{v,c}(l) \leq 1, \ \forall c, l && (4.16d)\\
& x_{v,c}(l) \geq 0, \forall v, c, l && (4.16e)\\
& y_v(l) \geq 0, \ \forall v, l && (4.16f)\\
& x_{v,c}(l), y_v(l) \in \{0, 1\}, \forall v, c, l && (4.16g)
\end{aligned}$$

Compared to (4.5), the main difference is that (4.16) has two extra constraints (4.16c) and (4.16d). Constraint (4.16c) ensures that data can only be placed in at most one subset of data centers across V. And constraint (4.16d) follows from constraint (4.16b). Using this reformulation Datum can now be explained in two steps.

[11] Note that, in practice, the number of data centers is usually small, e.g., 10 – 20 world-wide. Further, to avoid exponential explosion of V, the subsets included in V can be limited to only have a constant number of data centers, where the constant is determined by the maximal number of replicas to be stored.

Step 1: Solve (4.17) while treating the geo-distributed data cloud as a single data center. Specifically, define $Y(l) = \sum_{v=1}^{V} y_v(l)$ and $X_c(l) = \sum_{v=1}^{V} x_{v,c}(l)$. Note that, $Y(l)$ and $X_c(l)$ are $0-1$ variables from Constraint (4.16c) and (4.16d). Further, ignore the middle term in the objective, i.e., the ExecCost. Finally, for each quality level l, consider a "transformed" cost $\beta^*(l)$. We discuss how to define $\beta^*(l)$ below. This leaves the "single data center" problem (4.17). Crucially, this formulation can be solved optimally in polynomial time using the results for the case of a data cloud made up of a single data center (Section 4.3.1).

$$
\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L} \beta^*(l) Y(l) + \sum_{c=1}^{C} \sum_{l=1}^{L} f(l) X_c(l) && (4.17)\\
\text{subject to} \quad & X_c(l) \le Y(l), \ \forall c, l \\
& \sum_{l=w_c}^{L} X_c(l) = 1, \ \forall c \\
& X_c(l) \ge 0, \forall c, l \\
& Y(l) \ge 0, \ \forall l \\
& X_c(l), Y(l) \in \{0, 1\}, \forall c, l
\end{aligned}
$$

The remaining issue is to define $\beta^*(l)$. Note that the reason for using transformed costs $\beta^*(l)$ instead of $\beta_v(l)$ is that the optimal $\beta_v(l)$ cannot be known precisely without also optimizing the data placement. Thus, in defining $\beta^*(l)$ we need to anticipate the execution costs that result from data placement and replication given the purchase of data with quality level l. This anticipation then allows a decomposition of data purchasing and data placement decisions. Note that the only inaccuracy in the heuristic comes from the mismatch between $\beta^*(l)$ and $\min\{\beta_v(l) + \sum_{c \in C^*(l)} \alpha_{v,c}(l)\}$ where $C^*(l)$ is the set of customers who buy at quality level l in an optimal solution – if these match for the minimizer of (4.5) then the heuristic is exact. Indeed, in order to minimize the cost of locating quality levels to data centers, and allocating clients to data centers and quality levels, the set of data centers v where an optimal solution chooses to put quality level l has to minimize the cost of data transfer in the set v and allocating all clients who get data at quality level l, i.e. $C^*(l)$, to this set of data centers v.

Many choices are possible for the transformed costs $\beta^*(l)$. A conservative choice is $\beta^*(l) = \min_v \beta_v(l)$, which results in a solution (with Step 2) whose OperCost + PurchCost is a lower bound to the corresponding costs in the optimal solution

of (4.5).[12] However, it is natural to think that more aggressive estimates may be valuable. To evaluate this, we have performed experiments in the setting of the case study (see Section 4.4) using the following parametric form $\beta^*(l) = \min_v\{\beta_v(l) + \mu_1 \sum_{l' \le l} \sum_{w_c = l'} \alpha_{v,c}(l') e^{-\mu_2(l-l')}\}$, where μ_1 and μ_2 are parameters. This form generalizes the conservative choice by providing a weighting of $\alpha_{v,c}(l')$ based on the "distance" of the quality deviation between l' and the target quality level l. The idea behind this is that a client is more likely to be served data with quality level close to the requested minimum quality level of the client. Here we use the exponential decay term $e^{-\mu_2(l-l')}$ to capture the possibility of serving the data with quality level l to a client with minimum quality level $l' \le l$. Interestingly, in the setting of our case study, the best design is $\mu_1 = \mu_2 = 0$, i.e., the conservative estimate $\beta^*(l) = \min_v \beta_v(l)$, and so we adopt this $\beta^*(l)$ in Datum.

Step 2: At the completion of Step 1 the solution (X, Y) to (4.17) determines which quality levels should be purchased and which quality level should be delivered to each client. What remains is to determine data placement and data replication levels. To accomplish this, we substitute (X, Y) into (4.16), which yields (4.18).

$$\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L}\sum_{v=1}^{V} \beta_v(l) y_v(l) + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V} \alpha_{v,c}(l) x_{v,c}(l) \\
& + \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{v=1}^{V} f(l) x_{v,c}(l) && (4.18)\\
\text{subject to} \quad & x_{v,c}(l) \le y_v(l), \ \forall c, l && (4.18a)\\
& \sum_{l=w_c}^{L}\sum_{v=1}^{V} x_{v,c}(l) = 1, \ \forall c && (4.18b)\\
& \sum_{v=1}^{V} y_v(l) = Y(l) && (4.18c)\\
& \sum_{v=1}^{V} x_{v,c}(l) = X_v(l) && (4.18d)\\
& x_{v,c}(l) \ge 0, \forall v, c, l && (4.18e)\\
& y_v(l) \ge 0, \ \forall v, l && (4.18f)\\
& x_{v,c}(l), y_v(l) \in \{0, 1\}, \forall v, c, l && (4.18g)
\end{aligned}$$

[12] However the ExecCost cannot be bounded, thus we cannot obtain a bound for the total cost. The proof of this is simple and is not included in the chapter.

The key observation is that this is no longer a computationally hard ILP. In fact, the inclusion of (X, Y) means that it can be solved in closed form. Specifically, let $C(l)$ denote the set of clients that purchase data with quality level l, i.e., $C(l) = \{c : X_c(l) = 1\}$. Then (4.19) gives the optimal solution of (4.18).

$$y_v(l) = \begin{cases} 1, & \text{if } Y(l) = 1 \text{ and } v = \operatorname{argmin}\{\beta_v(l) + \sum_{c \in C(l)} \alpha_{v,c}(l)\}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.19a}$$

$$x_{v,c}(l) = \begin{cases} y_v(l), & \text{if } c \in C(l), \\ 0, & \text{otherwise.} \end{cases} \tag{4.19b}$$

Proof. We start by discussing the form of $x_{v,c}(l)$. Consider the following two cases based on the value of $Y(l)$.

1. For any quality level l', if $Y(l') = 0$, then $\forall v$, $\sum_{v=1}^{V} y_v(l') = Y(l') = 0$. From the non-negativity of $y_v(l')$, $\forall v$, $y_v(l') = 0$. Further, $\forall v, c$, $x_{v,c}(l') = 0$ from (4.18a).

2. For any quality level l', if $Y(l') = 1$, then from the definition of $y_v(l)$ and $Y(l)$, $\exists! v' \in V$, such that $y_{v'}(l') = Y(l') = 1$. Recall that $C(l') = \{c : X_c(l') = 1\}$ represents the set of clients that are assigned data with quality level l' by Step 1 in Section 4.3.2.

   a) For client $c' \in C(l')$, $X_{c'}(l') = 1$. Since v' is the unique data center set across V such that $y_{v'}(l') = 1$, from (4.18a) and (4.18b), $x_{v',c'}(l') = 1$ and $x_{v,c'}(l) = 0$, $\forall v \neq v'$ or $l \neq l'$. In other words, $x_{v,c'}(l') = y_v(l')$, $\forall v \in V, c \in C(l')$.

   b) For client $c \notin C(l')$, $X_c(l') = 0$. From the definition of $X_c(l')$, $x_{v,c}(l') = 0$, $\forall v$.

In all above cases, the optimal solution $\{x_{v,c}(l), y_v(l)\}$ of (4.18) satisfies the following:

$$x_{v,c}(l) = \begin{cases} y_v(l), & \text{if } c \in C(l), \\ 0, & \text{otherwise.} \end{cases} \tag{4.20}$$

Next, we use this form for $x_{v,c}(l)$ to derive $y_v(l)$. After substituting (4.20) into (4.18), most constraints become trivial due to the form of (4.20) and the optimality of $X_c(l)$

and $Y(l)$. And we only need to optimize the objective function with the constraints stating that $y_v(l)$ is binary, and $\sum_v y_v(l) = Y(l)$. Thus, we only need to optimize the following problem.

$$\begin{aligned}
\text{minimize} \quad & \sum_{l:Y(l)=1} \sum_{v=1}^{V} \beta_v(l) y_v(l) \\
& + \sum_{l:Y(l)=1} \sum_{c \in C(l)} \sum_{v=1}^{V} (\alpha_{v,c}(l) + f(l)) y_v(l) \\
\text{subject to} \quad & \sum_{v=1}^{V} y_v(l) = Y(l), \forall v, c, l \\
& y_v(l) \in \{0, 1\}, \forall v, c, l
\end{aligned}$$

The above optimization can be decoupled by l and optimized across v, yielding the closed form solution in (4.19).

□

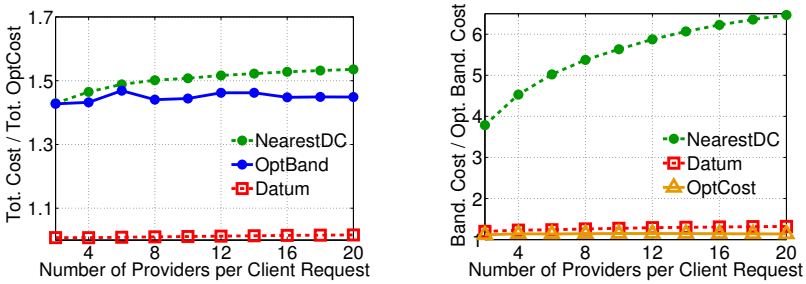
4.4 Case Study

We now illustrate the performance of Datum using a case study of a geo-distributed data cloud running in North America. While the setting we use is synthetic, we attempt to faithfully model realistic geography for data centers in the data cloud, data providers, and clients. Our focus is on quantifying the overall cost (including data purchasing and bandwidth/latency costs) of Datum compared to two existing designs for geo-distributed data analytics systems and the optimal. To summarize, the highlights of our analysis are

1. Datum provides consistently lower cost (> 45% lower) than existing designs for geo-distributed data analytics systems.

2. Datum achieves near optimal total cost (within 1.6%) of optimal.

3. Datum achieves reduction in total cost by significantly lowering purchasing costs without sacrificing bandwidth/latency costs, which stay typically within 20-25% of the minimal bandwidth/latency costs necessary for delivery of the data to clients.



(a) Total Cost (b) Bandwidth Cost

Figure 4.2: Illustration of the near-optimality of Datum as a function of the complexity of client requests (i.e., the average number of providers data must be procured from in order to complete a client request).

4.4.1 Experimental setup

The following outlines the setting in which we demonstrate the empirical performance of Datum.

Geo-distributed data cloud. We consider a geographically distributed data cloud with 10 data centers located in California, Washington, Oregon, Illinois, Georgia, Virginia, Texas, Florida, North Carolina, and South Carolina. The locations of the data centers in our experiments mimic those in [159] and include the locations of Google's data centers in the United States.

Clients. Client locations are picked randomly among US cities, weighted proportionally to city populations. We consider 100 clients. Each client requests data from a subset of data providers, chosen *i.i.d.* from a Uniform distribution. Unless otherwise specified, the average number of providers per client request is $P/2$. There are 8 quality levels. The quality level requested from each chosen provider follows a Zipf distribution with mean $L_p/2$ and shape parameter 30. P and L_p are defined as in Section 4.2.1 and Section 4.2.2. We choose a Zipf distribution motivated by the fact that popularity typically follows a heavy-tailed distribution [160]. Results are averaged over 20 random instances. We observe that the results of the 20 instances for the same plot are very close (within 5%), and thus do not show the confidence intervals on the plots.

Data providers. We consider 20 data providers. We place data providers in the second and third largest cities within a state containing a data center. This ensures that the data providers are nearby, but not right on top of, data center and client locations.

Operation and execution costs. To set operation and execution costs, we compute the geographical distances between data centers, clients and providers. The operation and execution costs are proportional to the geographical distances, such that the costs are effectively one dollar per gigameter. This captures both the form of bandwidth costs adopted in [27] and the form of latency costs adopted in [26].

Data purchasing costs. The per-query purchasing costs are drawn *i.i.d.* from a Pareto distribution with mean 10 and shape parameter 2 unless otherwise specified. We choose a Pareto distribution motivated by the fact that incomes and prices often follow heavy-tailed distributions [160]. Results were averaged over 20 random instances. To study the sensitivity of Datum to the relative size of purchasing and bandwidth costs, we vary the ratio of them between $(0.01, 100)$.

Baselines. We compare the performance of Datum to the following baselines.

- *OptCost* computes the optimal solution to the data cloud cost minimization problem by solving the integer linear programming (4.5). Note that this requires solving an NP-hard problem, and so is not feasible in practice. We include it in order to benchmark the performance of Datum.
- *OptBand* computes the optimal solution to the bandwidth cost minimization problem. It is obtained by minimizing only the operation cost and execution cost in the objective of (4.5). Bandwidth cost minimization is commonly considered as a primary goal for cost minimization in geo-distributed data analytics systems [28]. Due to computational complexity, heuristics are usually applied to minimize the bandwidth cost. Here, instead of implementing a heuristic algorithms, we optimistically use OptBand in order to lower bound the achievable performance. Note that this also requires solving an NP-hard problem and thus is not feasible in practice.
- *NearestDC* is a greedy heuristic for the total cost minimization problem that is often applied in practice. It serves the clients exactly what they ask for by purchasing the data and storing it at the data center closest to the data provider.

4.4.2 Experimental results

Quantifying cost reductions from Datum. Figure 4.2(a) illustrates the costs savings Datum provides. Across levels of query complexity (number of providers involved),

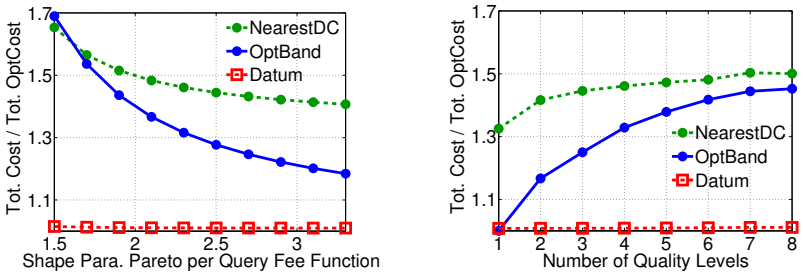


Figure 4.3: Illustration of Datum's sensitivity to query parameters. (a) varies the heaviness of the tail in the distribution of purchasing fees. (b) varies the number of quality levels available. Note that Figure 4.2 sets the shape parameter of the Pareto governing purchasing fees to 2 and includes 8 quality levels.

Datum consistently provides $>$ 45% savings over OptBand and $>$ 51% savings compared to NearestDC. Further, Datum is within 1.6% of the optimal cost in all these cases. The improvement of Datum compared to OptBand comes as a result of optimizing purchasing decisions at the expense of increased bandwidth. Importantly, Figure 4.2(b) shows that the extra bandwidth cost incurred is small, $20-25\%$. Thus, joint optimization of data purchasing and data placement decisions leads to significant reductions in total cost without adversely impacting bandwidth costs.

The form of client queries. To understand the sensitivity of the cost reductions provided by Datum, we next consider the impact of parameters related to client queries. Figure 4.2 shows that the complexity of queries has little impact on the cost reductions of Datum. Figure 4.3 studies two other parameters: the heaviness of the tail of the per-query purchasing fee and the number of quality levels offered.

Across all settings, Datum is within 1.6% of optimal; however both of these parameters have a considerable impact on the cost savings Datum provides over our baselines. In particular, the lighter the tail of the prices of different quality levels is, the less improvement can be achieved. This is a result of greater concentration of prices across quality levels leaving less room for optimization. Similarly, fewer quality levels provides less opportunity to optimize data purchasing decisions. At the extreme, with only one quality level available, the opportunity to optimize data purchasing goes away and OptBand and OptCost are equivalent.

Data purchasing vs. bandwidth costs. The most important determinant of the magnitude of Datum's cost savings is the relative importance of data purchasing

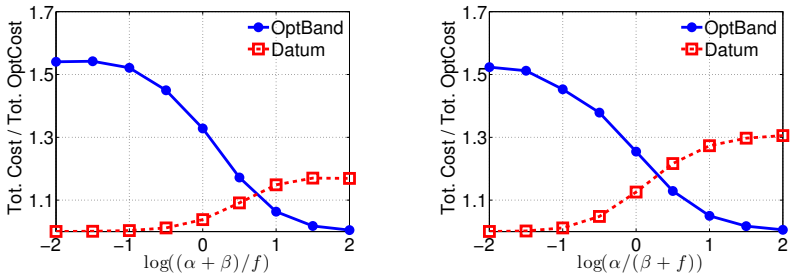


Figure 4.4: Illustration of the impact of bandwidth and purchasing fees on Datum's performance. NearestDC is excluded because its costs are off-scale. (a) varies the ratio of bandwidth costs (summarized by $\alpha + \beta$) to purchasing costs (summarized by f). (b) varies the ratio of costs internal to the data cloud (α) to costs external to the data cloud ($\beta + f$). Note that in Figure 4.2 the ratios are set to $\log(\frac{\alpha+\beta}{f}) = -0.5$ and $\log(\frac{\alpha}{\beta+f}) = -1$.

costs. In one extreme, if data is free, then the data purchasing decisions disappear and the problem is simply to do data placement in a manner that minimizes bandwidth costs. In the other extreme, if data purchasing costs dominate, then data placement is unimportant. In Figure 4.4 we only compare total costs among OptCost, OptBand, and Datum. NearestDC is far worse (more than 5 times worse than OptCost in some cases) and thus is dropped from the plots. Figure 4.4(a) studies the impact of the relative size of data purchasing and bandwidth costs. When the x-axis is 0, the data purchasing and bandwidth costs of the data center are balanced. Positive values mean that bandwidth costs dominate and negative values mean that data purchasing costs dominate. As expected, Datum's cost savings are most dramatic in regimes where data purchasing costs dominate. Cost savings can be 54% in extreme settings. Data purchasing costs are expected to dominate in the future – for some systems this is already true today. However, it is worth noting that, in settings where bandwidth costs dominates, Datum can deviate from the optimal cost by 10 – 20% in extreme circumstances, and can be outperformed by the MinBand benchmark. Of course, Datum is not designed for such settings given its prioritization of the minimization of data purchasing costs.

Internal vs. external costs. An important aspect of the design of Datum is the decomposition of data purchasing decisions from data placement decisions. This provides a separation between the internal and external operations (and costs) of Datum. Given this separation, it is important to evaluate the sensitivity of Datum's design to the relative size of internal and external costs.

Since Datum prioritizes the optimization of external costs (optimizing them in Step 1, see Section 4.3.2), it is natural to expect that Datum performs best when these costs dominate. This is indeed the case, as illustrated in Figure 4.4(b). Like in Figure 4.4(a), when the x-axis is 0, the internal and external costs are balanced. Positive values indicate the internal costs dominate and negative values indicate the external costs dominate. In settings where external costs dominate, Datum can provide 50% cost savings and be within a few percent of the optimal. However, in cases when internal costs dominate, Datum can deviate from the optimal cost by $10-30\%$ in extreme circumstances, and can be outperformed by the MinBand benchmark. Note that, as data purchasing costs grow in importance, external costs will dominate, and so we can expect that Datum will provide near optimal performance in practical settings.

Scalability Computing the optimal solution to the ILP as our benchmark is a NP-hard problem and can quickly become computationally intractable as the problem size grows. Thus, we limit the number of clients to 100 in our evaluation. In contrast, Datum scales well as the problem size grows, since it only requires solving a linear program with a size that scales linearly with the problem size.

4.5 Concluding Remarks

This work sits at the intersection of two recent trends: the emergence of online data marketplaces and the emergence of geo-distributed data analytics systems. Both have received significant attention in recent years across academia and industry, changing the way data is bought and sold and changing how companies like Facebook run queries across geo-distributed databases [27, 28]. In this chapter we study the engineering challenges that come when online data marketplaces are run on top of a geo-distributed data analytics infrastructure. Such cloud data markets have the potential to be a significant disruptor (as we highlight in Section 4.1). However, there are many unanswered economic and engineering questions about their design. While there has been significant prior work on economic questions [29, 30, 32, 125, 130, 161], the engineering questions have received much less attention.

In this chapter, we have presented the design of a geo-distributed cloud data market: Datum. Datum jointly optimizes data purchasing decisions with data placement decisions in order to minimize the overall cost. While the overall cost minimization problem is NP-hard (via a reduction to/from the facility location problem),

Datum provides near-optimal performance (within 1.6% of optimal) in realistic settings via a polynomial-time algorithm that is provably optimal in the case of a data cloud running on a single data center. Additionally, Datum provides > 45% improvement over current design proposals for geo-distributed data analytics systems. Datum works by decomposing the total cost minimization problem into subproblems that allow optimization of data purchasing and data placement separately, which provides a practical route for implementation in real systems. Further, Datum provides a unified solution across systems using per-query pricing or bulk pricing, systems with data replication constraints and/or regulatory constraints on data placement, and systems with SLA constraints on delivery.

This chapter is meant to initiate the study of data purchasing and data placement for data markets; thus, there many directions are left for future exploration. For example, Datum assumes clients are single entities with fixed locations and which data they need is known a priori as a result of pre-signed contracts with data market providers. In practice, clients can also be geo-distributed large companies with different data requirements in different locations. Exploring the optimal data purchasing and data placement with a mixed types of clients is interesting and challenging. Further, in Datum, computing resources used to process/clean raw data are assumed to be negligible. As data markets provide services on top of the raw data, e.g., analytics or learning services, the amount of computational power needed will grow and joint optimization of computational power and data purchasing and placement will become a crucial challenge.

4.A Appendix: Bulk Data Contracting

In bulk data contracting, the data cloud only has to pay a one-time fee $f(l, p)$ for data $q(l, p)$, no matter how many times the data is replicated on the cloud and transferred to clients. Compared to per-query contracting, the main difference lies in the purchasing fees modeling. Defining $z(l, p) \in \{0, 1\}$ to be equal to 1 if and only if data of quality $q(l, p)$ from data provider p is transferred to the data cloud, the whole optimization problem can still be formulated in a form similar to (4.5), with the purchasing costs now given by (4.4) and with the addition of the following constraint:

$$y_{p,d}(l) \leq z(l, p), \ \forall c, l, p, d \tag{4.22}$$

This constraint states that any data placed in the data cloud must have been purchased by the data cloud. As in the per-query contracting case, the data purchasing/placement decision for data from one data provider does not impact the data purchasing/placement decision for any other data providers. Thus, we drop the index p in the following.

In general, the cost minimization problem for bulk contracting is NP-hard. To be specific, the 1-level UFLP can reduce to the cost minimization problem for a geo-distributed data cloud, and the cost minimization problem can reduce to the 2-level UFLP in the bulk case. In the 2-level UFLP, facilities are organizing on 2 levels, $J_1 \times J_2$; each customer $i \in I$ has to be assigned to a valid path $p \in J_1 \times J_2$. A pass is valid if and only if both facilities are open along the path. More details on the 2-level UFLP can be found in [162].

The first reduction follows directly from the first part of the proof for Theorem 5. It can be easily proved by defining facilities in J_1 to be the quality levels, and using the same reformulation as the second part of the proof for Theorem 5 for the facilities in J_2, i.e. define facilities in J_2 to be pairs of quality levels and data centers. In the reduction, a facility $j_1 \in J_1$ is open if and only if the corresponding quality level l is purchased, and a facility $j_2 \in J_2$ is opened if and only if data of quality level l is placed in data center d.

While the cost minimization in bulk contracting is generally hard, it can be solved optimally in both the single data center and the geo-distributed data cloud settings under certain assumptions.

For the single data center case, we always have $z(l) = y(l)$ for all quality level l - this follows immediately from dropping the dependence of $y_d(l)$ in d, implying that $z(l)$ is only lower-bounded by $y(l)$ in the constraints. Furthermore, if the execution costs are the same across quality levels, the cost minimization problem can be formulated as follows:

$$\text{minimize} \quad \sum_{l=1}^{L} (\beta(l) + f(l))\, y(l) \tag{4.23}$$

$$\text{subject to} \quad x_c(l) \leq y(l), \ \forall c, l$$

$$\sum_{l=w_c}^{L} x_c(l) = 1, \ \forall c$$

$$x_c(l) \geq 0, y(l) \geq 0, x_c(l), y(l) \in \{0, 1\}, \forall c, l \tag{4.23a}$$

Since the decisions for variables $\{x_c(l)\}$ do not affect the objective value, (4.23) can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{l=1}^{L} (\beta(l) + f(l))\, y(l) \\ \text{subject to} \quad & \sum_{l=w_c}^{L} y(l) \geq 1, \ \forall l, c \\ & y(l) \in \{0, 1\}, \forall l \end{aligned} \tag{4.24}$$

Since there are customers buying the highest quality level, the highest level quality L is always purchased by the data cloud and $y(L) = 1$ in any feasible solution. Since all customers are satisfied and all costs are non-negative, an optimal solution for (4.24) is $y(L) = z(L) = 1$, $x_c(L) = 1$ with all other variables are set to 0. The result implies the data cloud will only purchase the highest quality level of data and serve that data to every customers.

For a geo-distributed data cloud, the cost minimization problem is generally hard. However, if we assume the operation cost and execution cost are independent of l, i.e., $\beta_d(l) = \beta_d$ and $\alpha_{d,c}(l) = \alpha_{d,c}$, it is easy to show that the optimal solution will only purchase the highest quality data as in the single data center case. We can then use Step 2 in Section 4.3.2 to give an optimal solution to the data placement problem.

CHAPTER 5

POWER CAPPING IN COLOCATION DATA CENTERS

The emergence of Internet and cloud services has significantly fueled demand for data centers worldwide, resulting in an aggregate power demand of 38GW as of 2012 (a growth of 63% compared to 2011) [163]. Accommodating the accelerated demand, however, is costly. It can be a multi-million or even multi-billion dollar project to construct a new data center or expand an existing data center's capacity (typically measured in IT critical power). For example, power infrastructure, including back-up generation and uninterrupted power supplies (UPS), is sized based on the critical power budget and estimated at U.S.$10-25 per watt [164]. The capital expense (CapEx) in power and cooling infrastructure even exceeds 1.5 times the total energy cost of operating a data center over a 15-year lifespan [164–166]. Moreover, other limitations, such as space and grid capacity, may also prohibit the expansion of data center capacity.

In view of the high CapEx and practical constraints for building new capacity, data center operators aggressively oversubscribe the existing infrastructure throughout the power hierarchy (e.g., UPS level and PDU level) by deploying more servers than the power budget/capacity allows. This is equivalent to under-provisioning the capacity to reduce CapEx for new data center construction: to deploy the same number of servers, the data center capacity can be downsized to save CapEx. The rationale underlying oversubscription is that, in most cases, not all servers simultaneously run at their peak powers and thus, the servers' aggregate power usage remains well below the power budget with a very high probability, as illustrated by measurements in [34, 165].

A dangerous consequence of oversubscription is the emergence of power *emergencies* that bring significant challenges for data center uptime. Although uncommon, when loads on many servers peak simultaneously, the aggregate power demand will exceed the capacity (e.g., overloading UPS), thus compromising the desired power availability and even leading to unplanned downtime incidents [33, 166]. Such

power emergencies have become a major cause of unplanned data center outages, which may take several hours or even days to fully recover and incur significant economic losses (estimated average of $901,560 per incident) [167, 168].

Naive techniques to handle emergencies, e.g., arbitrarily putting involved servers into low power states or switching them off, are not appealing [33, 165, 169], because they may result in significant performance degradation and even business disruption. Instead, a graceful power capping solution is required to coordinate servers' power usage at a minimum performance loss. Towards this end, prior research has studied various techniques, e.g., judiciously scaling down CPU frequency [33], admission control, and workload migration (to public clouds and/or other servers not subject to power emergency) [34–36]. These studies, although promising, are only applicable for owner-operated data centers (e.g., Google), where data center operators have control over the physical servers and hence can easily coordinate the servers to minimize performance impact.

In sharp contrast, we study power capping in multi-tenant data centers, an under-explored but even more common type of data center. In a multi-tenant data center, multiple individual tenants house and manage their own physical servers, while the data center operator is responsible for power and cooling infrastructure support. Like owner-operated data centers, multi-tenant data center operators also aggressively oversubscribe capacity to gain more revenue and/or save CapEx by selling the capacity to more tenants than it allows.

To handle a power emergency due to oversubscription, however, multi-tenant data center operators cannot directly apply existing power capping techniques (e.g., through server and workload management [33, 34]), because of lack of control over tenants' servers. Thus, a common practice today for a multi-tenant data center operator is to simply take the risk of capacity overloading when many tenants' power demands peak simultaneously. In other words, whether or not an outage will occur depends largely on the robustness of infrastructure. Consequently, according to a 2014 Uptime Institute survey, 25% of the tenants have experienced at least one power outage (for which capacity overloading is a major cause) over the past year [167, 168, 170].

To address the lack of coordination among tenants to shed power during a power emergency, we propose a novel COOrdinated Power management solution, called COOP, that leverages a market mechanism called supply function bidding [171] commonly used in electricity markets in order to incentivize and coordinate in-

dividual tenants' power demand reduction. The challenge in designing such a mechanism is that the mechanism must not bring much overhead and the overall impact of power reduction on tenants' application performance needs to be as little as possible (similar to the design objective of power capping techniques, e.g., [33, 34], for owner-operated data centers). COOP achieves both. It only solicits one bidding parameter from each participating tenant which, when plugged into the supply function, specifies the amount of power reduction and corresponding reward the tenant is willing to accept. More importantly, the overall performance impact across all the participating tenants is small: the total performance cost incurred by the tenants is very close to the ideal case where the data center operator is assumed to have full control over tenants' servers.

The novelty of this study is that COOP is the first market-based solution for handling an emergency caused by capacity oversubscription in a multi-tenant data center, an important yet rarely-studied type of data center.

Concretely, this chapter makes the following contributions. First, we introduce and formulate the problem of multi-level power capping in a multi-tenant data center. Second, we propose a supply function bidding based mechanism, motivated by the literature on electricity markets [171, 172], to incentivize and coordinate tenants' power reduction during a power emergency, capping the aggregate power demand while minimizing the total performance cost. Third, we validate COOP using realistic settings on a testbed. Our results show that COOP is efficient in terms of minimizing total performance cost and that COOP is "win-win", increasing the operator's profit and reducing tenants' cost (through financial compensation).

5.1 Opportunities and Challenges

Multi-tenant data centers are common in practice. There are over 1,400 multi-tenant data centers in the U.S. alone [173]. As a quickly growing data center segment, it consumes as much as five times the energy of Google-type owner-operated data centers combined together (37.3% v.s. 7.8%, in percentage relative to all data center energy usage, excluding tiny server closets) [174]. It provides a cost-effective and scalable data center solution to many industry sectors, including major websites (e.g., Twitter), banking, content delivery provider (e.g., Akamai) [175], and even IT giants (e.g., Microsoft) that leverage third-party data centers to complement their own facilities [176].

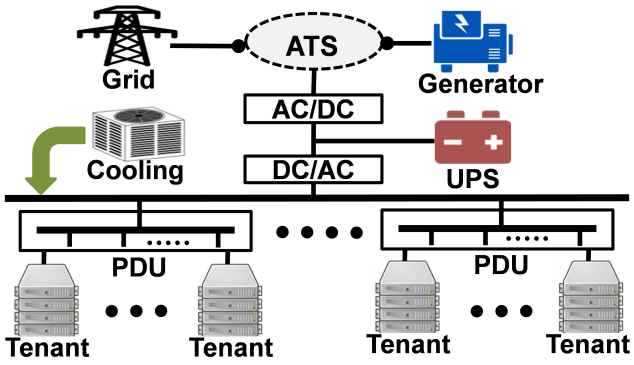


Figure 5.1: Data center infrastructure.

Despite their importance, multi-tenant data centers have been less investigated than the more visible owner-operated data centers (like Google). They present new challenges due to the operator's lack of central control over servers. This means that standard approaches for handling power emergencies do not apply to multi-tenant data centers [33, 34, 36].

5.1.1 Power in Multi-Tenant Data Centers

While different designs (e.g., using fuel cell as the main power source [177]) are emerging, most data centers, including new constructions, still heavily rely on diesel generators, uninterrupted power supplies (UPS), and power distribution units (PDU) for achieving high power availability. Fig. 5.1 illustrates the infrastructure commonly found in today's multi-tenant data centers: electricity first enters data center through a utility substation; next, through AC/DC and DC/AC double conversions, power goes to PDUs, which then distribute power to individual tenants' server racks. By default, the automatic transfer switch (ATS) takes power from the utility and, in the event of a grid failure, switches to the back-up generator. As the generator cannot be instantly turned on, a UPS will be discharged to supply continuous power until the diesel generator is fully activated.

The power hierarchy. In a multi-tenant data center, the power hierarchy often has a tree-type structure. At the top level sits the centralized UPS, which supports multiple cluster-level PDUs at a lower level. Each cluster-level PDU typically has a capacity of 200-300kW, supporting around 50 racks which then distributes power to servers at the lowest level. Individual tenants may have highly diverse power demands, ranging from a few kW (often in a retail multi-tenant data center) to hundreds of kW or even larger, depending on their needs. Each PDU or even rack may also

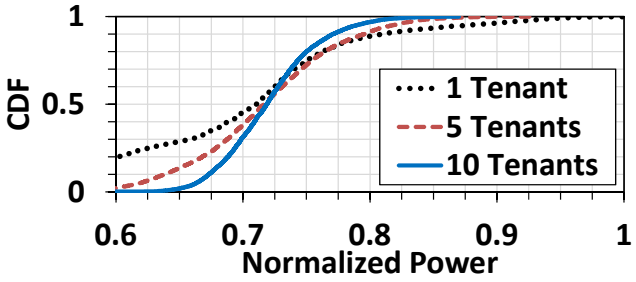


Figure 5.2: CDF of measured power usage.

have its own dedicated UPS (e.g., lead-acid battery, not shown in Fig. 5.1), which complements or even fully substitutes the centralized UPS while enhancing power availability at a lower cost [166].

The data center operator also provides reliable cooling. Among various designs, a multi-tenant data center usually uses mechanical chiller or direct-expansion air conditioning as the cooling mechanism, depending on the data center size.

Tenants' power usage. A crucial motivation for power oversubscription is the heterogeneity of tenant power usage. We present in Fig. 5.2 the temporal analysis of power measurement in a commercial multi-tenant data center collected from May to July, 2015. The data includes the server power usage of 10 tenants, subscribing approximately 500kW in total and ranging from sectors of utility, education, media, content distribution and public clouds. Fig. 5.2 shows the cumulative density function (CDF) of the power consumption by different groups of tenants, from 1 tenant to 10 tenants. The x-axis is normalized with respect to the sum of the maximum power usage of all the servers within that group of tenants.

We see that with more tenants (i.e., power hierarchy moves up from rack to clusters), statistical multiplexing effects of power demands become more significant, and it is even rarer for all tenants' servers to peak simultaneously. For example, for one tenant, the probability that the normalized power exceeds 80% of its peak is roughly 13%, whereas this number reduces to less than 3% for 10 tenants. Such observations have also been reported for owner-operated data centers like Google [169], whose server clusters are equivalent to tenants'. While the specific CDF of power usage varies with different data centers, the qualitative insights hold widely: power oversubscription is safe in *most* cases.

5.1.2 Opportunities for Oversubscription

Leasing data center capacity with power and cooling (typically $150/kW/month in the U.S. [178]) is the most significant revenue source for a multi-tenant data center operator. Naturally, through oversubscription, the operator can earn extra revenue by serving more tenants without upgrading the power/cooling infrastructure.[1] Except for increased risk of downtime (due to capacity overloading), there is almost no additional operating expense resulting from the operator's oversubscription, because in many large (especially wholesale) multi-tenant data centers, the energy cost will be split across tenants depending on their actual usage.

In Table 5.1, we show the potential economic benefit of oversubscription for multi-tenant operators, based on a leasing cost of $150/kW/month [178]. For each kW capacity, with x% oversubscription, the operator earns an extra revenue of $\$150 \times 12 \times x\%$ per year. Overloading probability is obtained based on measured power usage of the 10-tenant cluster shown in Figure 5.2: with x% oversubscription, overloading occurs if the aggregate power demand exceeds $100/(100 + x)$ of its maximum.

We see from Table 5.1 that there is a great economic opportunity for oversubscription. The last row in Table 5.1 shows the maximum reward rate for power reduction that can be offered to tenants without decreasing the operator's profit (assuming that during each power emergency, the tenants' server power demand reaches the peak, i.e., rated capacity plus the oversubscribed amount). If the operator is not too aggressive and oversubscribes its capacity by less than 20%, it can offer a reward rate at more than 200 times of the market electricity price without losing profit.

5.1.3 Challenges for Oversubscription

The economic benefit of oversubscription is significant, but the danger of creating power emergencies cannot be ignored. While a power emergency may not necessarily lead to a downtime given infrastructure redundancy (e.g., "2N" duplicating all power/cooling units), ignoring it without proper attention is not a good practice, as IT critical loads exceeding the design capacity will lose the desired redundancy protection and increase outage risk [34, 166]. In fact, according to a recent survey [170], despite redundancy, 25% of tenants have experienced at least one unplanned

[1] A tenant may also oversubscribe its reserved capacity to reduce leasing cost, but it must handle resulting emergencies by itself. Thus, this is addressed by prior research [33, 34].

Oversubscription	10%	15%	20%	25%
Extra Revenue ($/kW/year)	180	270	360	450
Probability of Overloading (%)	1%	1%	2%	3%
Est. Overloading Time (hours/year)	88	88	175	262
Max. Reward for Power Reduction ($/kW/hour)	22.60	23.63	12.32	8.56

Table 5.1: Analysis of Capacity Oversubscription.

power-related downtime over the past year (for which IT loads exceeding the design capacity is a major cause [167, 168]). Therefore, regardless of redundancy protection, it is very critical to handle power emergencies by gracefully capping the servers' power demand below the design capacity.

One approach for handling power emergencies is to temporarily "boost" power supply by discharging an energy storage device (diesel generation, e.g., battery in UPS) [166, 179]. However, a potential risk when leveraging diesel generation is that the cooling capacity (typically sized based on the IT critical load due to high CapEx) may still be exceeded [180, 181], because the servers' actual aggregate power consumption (i.e., cooling load) is not reduced to the designed level. As a result, discharging diesel generation can safely handle power capacity overloading, but *not* necessarily cooling capacity overloading, which can quickly lead to server overheating and is another major cause for unplanned outages [168]. Moreover, inappropriate/frequent discharging may drain the diesel generation sooner and compromise data center reliability (e.g., recent Google power failure incident, for which Google cited "extended or repeated battery drain" as a root cause [182]).

In this chapter we propose to handle power emergencies via IT power reduction from the tenants. In practice, however, the operator lacks control over tenants' servers and hence cannot enforce tenants' power reduction during an emergency, which is due to the operator's *fault* of oversubscription. Even assuming that the operator can somehow force tenants to cut power, which tenants should reduce power and by how much still needs to be decided so as to minimize tenants' performance degradation. This requires the knowledge of tenants' workloads and business values, which is private information and unknown to the operator. Thus, despite the huge economic

benefit of power oversubscription, gracefully capping tenants' power to handle the resulting emergencies with a minimum performance impact on tenants presents significant challenges for multi-tenant data center operators.

A data center typically oversubscribes capacity at multiple interdependent power hierarchies (e.g., data center UPS-level, cluster PDU-level, and even rack-level), each having its own capacity below which the involved tenants' aggregate power demand should be capped at all times [34, 183]. In the following sections, we first start from a simple case when there is only data center UPS-level capacity constraint. We provide an analysis of the efficiency of our mechanism, the supply function bidding, proposed in COOP. Our analysis precisely characterizes the equilibrium outcome, both when tenants are price-taking and when they are price-anticipating. In both cases, our results highlight that COOP suffers little performance loss compared to the socially optimal outcome, both from the operator's and the tenants' perspectives. Then we generalize our design to solve power capping at multiple power hierarchies and build a prototype to show that COOP is efficient in terms of minimizing the total performance cost.

5.2 COOP with a Single Data Center Level Power Constraint

A data center typically oversubscribes capacity at multiple interdependent power hierarchies (e.g., data center UPS-level, cluster PDU-level, and even rack-level), each with its own capacity below which the involved tenants' aggregate power demand should be capped at all times [34, 183]. COOP is not restricted to any particular levels. In this section, our focus is the design of a mechanism for a colo operator in response to data center level capacity constraint only. We begin by describing a model, then propose an overview of the mechanism in this section, and finally, we provide efficiency analysis for our mechanism in Section 5.3 and Section 5.3.2. We generalize our design to a multiple level power oversubscription scenario in Section 5.4.

Recall that the colo operator is responsible for non-IT facility support (e.g., high-availability power, cooling). We capture the non-IT energy consumption using Power Usage Effectiveness (PUE) γ, which is the ratio of the total data center energy consumption to the IT energy consumption. Typically, γ ranges from 1.1 to 2.0, depending on factors such as outside temperature.

When the operator receives an power capping signal on data center level, it has two

options for satisfying the load reduction. First, without involving the tenants, the colo operator can use its on-site backup diesel generator.[2] We denote the amount of energy reduction by diesel generation by y and the cost per kWh of diesel generation (e.g., for fuels) by α.

Alternatively, the colo operator could try to extract IT energy reductions from the tenants. We consider a setting where there are N tenants, $i \in \mathcal{N} = \{1, 2, \cdots, N\}$. When shedding energy consumption, a tenant i will incur some costs and we denote the cost from shedding s_i by a function $c_i(s_i)$. These costs could be due to wear-and-tear, performance degradation, workload shifting, etc. For the purposes of our model, we do not specify which technique reduces the IT energy, only its cost. For details on how one might model such costs, see [169, 185–187]. A standard, natural assumption on the costs is the following.

Assumption 1. *For each n, the cost function $c_n(s_n)$ is continuous, with $c_n(s_n) = 0$ if $s_n \leq 0$. Over the domain $s_n \geq 0$, the cost function c_n is convex and strictly increasing.*

Intuitively, convexity follows from the conventional assumption that the unit cost increases as tenants reduce more energy (e.g., utilization becomes higher when servers are off, leading to a faster increase in response time of tenants' workloads).

5.2.1 An overview of COOP

The operation of COOP with data center level constraint only is summarized below, and then discussed in detail in the text that follows.

1. The colo operator receives an reduction target δ and broadcasts the supply function $S(\cdot, p)$ specified by(5.1) to tenants;
2. Participating tenants respond by placing their bids b_n;
3. The colo operator decides the amount of on-site generation y and market clearing price p to minimize its cost, using equations (5.2) to set the market clearing price p and (5.3) to set y in order to minimize the cost of power capping;

[2] Other alternatives, e.g., battery [35], usually only last for < 5 minutes. So, diesel generation is the typical method [184].

4. Power capping is exercised. $\forall n \in \mathcal{N}$, tenant n sheds $S(b_n, p)$, and receives $pS(b_n, p)$ reward.

Given the overview above, we now discuss each step in more detail.

Step 1. Upon receiving an power capping notification of an energy reduction target δ, the colo operator broadcasts a parameterized supply function $S(b, p)$ to tenants (by, e.g., signalling to the tenants' server control interfaces, which are widely in use today [188]). The form of $S(b, p)$ is the following parameterized family[3]:

$$S(b_n, p) = \delta - \frac{b_n}{p}, \tag{5.1}$$

where p is an offered reward for each kWh of energy reduction and b_n is the bidding values that can be chosen by tenant n. This form is inspired by [171], where it is shown that by restricting the supply function to this parameterized family, the mechanism can guide the firms in the market to reach an equilibrium with desirable properties.[4] Note that, to be consistent with the supply function literature, we exchangeably use "price" and "reward rate" wherever applicable.

Step 2. Next, according to the supply function, each participating tenant submits its bid b_n to the colo operator. This bid specifies that, at each price p, it is willing to reduce $S(b_n, p)$ units of energy. The bid is chosen by tenants individually to maximize their own utility and can be interpreted as, e.g., the amount of IT service revenue that tenant n is willing to forgo. Note that b_n can be chosen to ensure that tenant n will not be required to reduce more energy than its capacity. To see this, note that since the operator is cost-minimizing, $p(\mathbf{b}, y) \leq \alpha$ always holds, i.e., the market clearing price is lower than the unit cost of diesel generation. Hence, if δ_n is the capacity of reduction for tenant n, as long as $b_n \geq \alpha(\delta - \delta_n)$, then

$$S(b_n, p) = \delta - \frac{b_n}{p} \leq \delta - \frac{b_n}{\alpha} \leq \delta_n.$$

An important note about the tenant bids is that the supply function is likely of a different form than the true cost function c_n, and so it is unlikely for the tenants to reveal their cost functions truthfully. This is necessary in order to provide a simple

[3]The supply function allows tenants to have negative supply, i.e., tenants consume more energy intentionally, which is neither profit maximizing nor practical. We show in Section 5.3 that energy reduction of each tenant is always nonnegative in both equilibrium and social optimal outcomes.

[4][171] studies the case where firms bid to supply an inelastic demand, which is equivalent to fixing the diesel generation $y = 0$ in our case. Allowing the operator to choose y in a cost-minimizing manner leads to significantly different results, as will be shown in Section 5.3.1 and Section 5.3.2.

form for tenant bids. Bidding their true cost functions is too complex and intrusive. However, a consequence of this is that one must carefully analyze the emergent equilibrium to understand the efficiency of the pricing mechanism. We study both the cases of price-taking and price-anticipating equilibrium in Section 5.3.

Step 3. After tenants have submitted their bids, the colo operator decides the amount of energy y to produce via on-site generation and the clearing price p. Given y, the market clearing price has to satisfy $\Sigma_n S(p(\mathbf{b}), b_n) + y = \delta$, thus

$$p(\mathbf{b}, y) = \frac{\sum_n b_n}{(N-1)\delta + y}. \tag{5.2}$$

To determine the amount of local generation y, the operator minimizes the cost of the two load-reduction options, i.e.,

$$y = \underset{0 \le y \le \delta}{\arg\min} (\delta - y) \cdot p(\mathbf{b}, y) + \alpha y. \tag{5.3}$$

Step 4. Finally, power capping is exercised and tenants receive financial compensation from the colo operator via the realized price in (5.2), shed load $S(p, b_n)$, and on-site generation produces the energy in (5.3).

5.3 Efficiency Analysis of COOP

Given the COOPmechanism described above, our task now is to characterize its efficiency. There are two potential causes of inefficiency in the mechanism: the cost minimizing behavior of the operator and the strategic behavior (bidding) of the tenants. In particular, since the forms of the tenant's cost functions are likely more complex than the supply function bids, tenants cannot bid their true cost function even if they wanted to. This means that evaluating the equilibrium outcome is crucial to understanding the efficiency of the mechanism.

Further, the equilibrium outcome that emerges depends highly on the behavior of the tenants – whether they are *price-taking*, i.e., they passively accept the offered market price p as given when deciding their own bids; or *price-anticipating*, i.e., they anticipate how the price p will be impacted by their own bids. We investigate both models, in Section 5.3.1 and Section 5.3.2, respectively.

In both cases, the goal of our analysis is to assess the efficiency of COOP. To this end, we adopt a notion of a (socially) optimal outcome, and focus on the following

social cost minimization (SCM) problem.

$$\text{SCM}: \quad \min \quad \alpha y + \sum_{i \in \mathcal{N}} c_i(s_i) \tag{5.4a}$$

$$\text{s.t.} \quad y + \gamma \cdot \sum_{i \in \mathcal{N}} s_i = \delta \tag{5.4b}$$

$$s_i \geq 0, \ \forall i \in \mathcal{N} \tag{5.4c}$$

$$y \geq 0, \tag{5.4d}$$

where s_i and c_i are tenant i's energy reduction and corresponding cost, respectively.

The objective in SCM can be interpreted as the tenants' cost plus the colo operator's cost. Note that the internal payment transfer between the colo operator and tenants cancels, and does not impact the social cost. Also, note that payment from the LSE to the colo operator is not included in the social cost objective, since it is independent of how the operator obtains the amount δ of load reduction. Additionally, we do not include the option of ignoring the power capping signal and taking the penalty, since the non-compliance penalties are typically extreme [189]. Finally, the Lagrange multiplier of (5.4b) can be interpreted as the social optimal price p^*, i.e., given this price as reward for energy reduction, each tenant will individually reduce their energy by an s_n that corresponds to the social cost minimization solution in (5.4).

Before moving to the analysis, in order to simplify notation, we suppress the PUE γ by, without loss of generality, setting $\gamma = 1$. To obtain results for $\gamma \neq 1$, simply take the results assuming $\gamma = 1$ and modify them in the following way: let y', δ' and α' be the diesel generation, EDR target and diesel price that appear in the results for $\gamma = 1$, replace them by $y' = y/\gamma$, $\delta' = \delta/\gamma$, and $\alpha' = \alpha\gamma$ where y, δ, α are the respective quantities when $\gamma \neq 1$.

5.3.1 Price-Taking Tenants

When tenants are price-taking, they maximize their net utility, which is the difference between the payment they receive and the cost of energy reduction, given the assumption that they consider their action does not impact the price. A price-taking tenant n will try to maximize the following payoff $P_n(b_n, p)$:

$$P_n(b_n, p) \ = pS_n(b_n, p) - c_n(S_n(b_n, p)) \tag{5.5a}$$

$$= p\delta - b_n - c_n\left(\delta - \tfrac{b_n}{p}\right). \tag{5.5b}$$

Here, the price-taking assumption implies that the variable p is considered to be as is. The price-taking assumption normally holds when the market consists of many

players of similar sizes who have little power to impact the market clearing price. The other market model, when tenants are price-anticipating, is analyzed in Section 5.3.2.The market equilibrium for price-taking tenants is thus defined as follows:

Definition 1. *A triple* $(\mathbf{b}, p, y)$ *is a (price-taking) market equilibrium if each tenant maximizes its payoff defined in* (5.5)*, market is cleared by setting price* p *according to* (5.2)*, and the amount of on-site generation is decided by* (5.3)*, i.e.,*

$$P_n(b_n; p) \geq P_n(\bar{b}_n; p) \quad \forall \bar{b}_n \geq 0, \quad n = 1, \ldots, N. \tag{5.6}$$

$$p = \frac{\sum_{i \in \mathcal{N}} b_i}{(N-1)\delta + y}. \tag{5.7}$$

$$y = \underset{0 \leq y \leq \delta}{\arg\min} (\delta - y) \cdot p(\mathbf{b}, y) + \alpha y. \tag{5.8}$$

5.3.1.1 Market Equilibrium Characterization

The key to our analysis is the observation that the equilibrium can be characterized by an optimization problem. Once we have this optimization, we can use it to characterize the efficiency of the equilibrium outcome. This approach parallels that used in [171]; however, the optimization obtained has a different structure due to local diesel generation. Note that, though we use an optimization to characterize the equilibrium, the game is not a potential game since the objective (5.9a) below is not a potential function.

Our first result highlights that, given any choice for on-site generation, a unique market equilibrium exists for the tenants, and can be characterized via a simple optimization.

Proposition 7. *Under Assumption 1, when tenants are price-taking, for any on-site generation level* $0 \leq y < \delta$*, there exists a market equilibrium, i.e., a vector* $\mathbf{b}^t = (b_1^t, \ldots, b_N^t) \geq 0$ *and a scalar* $p > 0$ *that satisfies* (5.2)*, and the resulting allocation* $s_n = S(b_n, p)$ *is the optimal solution of the following:*

$$\min_{\mathbf{s}} \quad \sum_{i \in \mathcal{N}} c_i(s_i) \tag{5.9a}$$

$$s.t. \quad \sum_{i \in \mathcal{N}} s_i = (\delta - y), \tag{5.9b}$$

$$s_i \geq 0, \ \forall i \in \mathcal{N}. \tag{5.9c}$$

Proof. When tenants are price takers, they maximize the payout $P_n(b_n, p) = pS_n(b_n, p) - c_n(s_n)$ over the bid b_n. Note that $b_n \in [0, p\delta]$ as no tenant will bid beyond $p\delta$ otherwise the payout $P_n < 0$. Hence $\mathrm{b} = (b_1, \ldots, b_n)$ is an equilibrium if and only if the following condition is satisfied

$$\frac{\partial^- c_n(s_n)}{\partial s_n} \leq p, \quad 0 \leq b_n < p\delta, \tag{5.10a}$$

$$\frac{\partial^+ c_n(s_n)}{\partial s_n} \geq p, \quad 0 < b_n \leq p\delta. \tag{5.10b}$$

At least one feasible solution to (5.9) exists because it is minimizing a continuous function over a compact set. Furthermore, (5.9b) - (5.9c) satisfy the standard constraint qualification, hence for the Lagrangian

$$L(\mathrm{s}, \mu) = \sum_n c_n(s_n) + \mu((\delta - y) - \sum_n s_n),$$

there exists optimal primal dual pair (s, μ), such that (5.9b) and (5.9c) are satisfied, and

$$\frac{\partial^- c_n(s_n)}{\partial s_n} \leq \mu, \quad s_n > 0, \tag{5.11a}$$

$$\frac{\partial^+ c_n(s_n)}{\partial s_n} \geq \mu, \quad s_n \geq 0. \tag{5.11b}$$

Given the optimal (s, μ), let $p = \mu$, and $b_n = p(\delta - s_n)$, then (5.9b) implies p satisfies (5.2), and (5.11a)-(5.11b) implies (5.10a) - (5.10b), hence an equilibrium exists.

Conversely, if (b, p) is an equilibrium and p satisfies (5.2), the resulting allocation s is optimal to (5.9). To see this, if $0 \leq s_n < \delta - y$ for all n, (5.10a)-(5.10b) is equivalent to (5.11a)-(5.11b) if we set $\mu = p$, hence (s, μ) is primal dual optimal pair for (5.9). If $s_n = (\delta - y)$, then $s_m = 0, \forall m \neq n$. In this case, we set $\bar{\mu} = \min\{p, \partial^+ c_n(s_n)/\partial s_n\}$, and we can check that $(\mathrm{s}, \bar{\mu})$ is the primal dual optimal solution for (5.9). □

This result is a key tool for understanding the overall market outcome. Intuitively, the operator running COOP is more likely (than the social optimal) to use on-site generation, since this reduces the price paid to tenants. The following proposition quantifies this statement.

Proposition 8. *Under Assumption 1, it is optimal for price-taking tenants to use on-site generation if and only if*

$$\alpha < \frac{(\Sigma_n b_n)}{(N-1)\delta}.^{5} \tag{5.12}$$

[5] We adopt the convention that $\frac{0}{0} = 0$ and $\frac{x}{0} = +\infty$ when $x > 0$. Therefore, when $N = 1$, unless the bid is 0, the condition is always satisfied.

However, when the operator is profit maximizing, it will turn on on-site generation if and only if

$$\alpha < \frac{N}{N-1}\frac{(\Sigma_n b_n)}{(N-1)\delta}. \tag{5.13}$$

This proposition is an important building block because the most interesting case to consider is when it is optimal to use some on-site generation and some tenant load shedding, i.e., $\delta > y^* > 0$. Otherwise the power capping requirement should be entirely fulfilled by tenants, and the analysis reduces to the case of an inelastic demand, as studied in [171]. Thus, subsequently, we make the following assumption, which ensures that on-site generation is valuable.

Assumption 2. *The unit cost of on-site generation is cheap enough that the optimal on-site generation is non-zero, i.e., α satisfies* (5.12)*.*

Note that, when Assumption 2 holds, by first-order optimality condition of (5.3) we have

$$y = \sqrt{\frac{(\Sigma_{i\in\mathcal{N}} b_i)N\delta}{\alpha}} - (N-1)\delta, \tag{5.14}$$

and so the market clearing price for the tenants given on-site generation is

$$p = \frac{\sum_{i\in\mathcal{N}} b_i}{(N-1)\delta + y} = \sqrt{\frac{(\Sigma_{i\in\mathcal{N}} b_i)\alpha}{N\delta}}. \tag{5.15}$$

Using these allows us to prove a complete characterization of the market equilibrium under price-taking tenants. This theorem is the key to our analysis of market efficiency.

Theorem 9. *When Assumptions 1 and 2 hold there is a unique market equilibrium, i.e., a vector $\mathbf{b}^t = (b_1^t, \ldots, b_N^t) \geq 0$, $y^t > 0$ and a scalar $p^t > 0$ that satisfies* (5.6)-(5.8)*, and the resulting allocation $(\mathbf{s}^t, y^t)$ where $s_n^t = S(b_n^t, p^t)$ is the optimal solution of the following problem:*

$$\min_{\mathbf{s},y} \quad \sum_n c_n(s_n) + \frac{\alpha}{2N\delta}(y + (N-1)\delta)^2 \tag{5.16a}$$

$$s.t. \quad \sum_n s_n = \delta - y, \tag{5.16b}$$

$$s_n \geq 0, \ \forall n, \quad y \geq 0. \tag{5.16c}$$

Proof. By Proposition 7, when tenants are price-taking, for any y, the there is always an equilibrium, and the resulting $\mathbf{s}$ is always the optimal allocation to provide $(\delta - y)$ energy reduction.

Hence we only need to verify that the on-site generation level y is the solution to (5.16a)-(5.16c). Similar to the proof of Proposition 7, by Assumption 2, the first order optimality condition for the y in (5.16a)-(5.16c) is $\frac{\alpha}{N\delta}(y + (N-1)\delta) = p$. By Proposition 7, p satisfies the relation (5.2), substitute the left-hand-side into (5.2) and solve for y, we have $y = \sqrt{\frac{\Sigma_n b_n N\delta}{\alpha}} - (N-1)\delta$. This is exactly the on-site generation y that minimizes $\text{cost}_o(\mathbf{b}, y)$ given in (5.14). Hence the data center will always pick y that is optimal for (5.16a)-(5.16c). Together with Proposition 7, an equilibrium exists, and the resulting allocation $(\mathbf{s}, y)$ is optimal for (5.16a)-(5.16c).

□

5.3.1.2 Bounding Efficiency Loss

We now use Theorem 9 to bound the efficiency loss due to strategic behavior in the market. Denote the socially optimal on-site generation by y^*, the optimal price that leads to the optimal allocation $s_i, \forall i \in \mathcal{N}$ by p^*, and let y^t and p^t be the allocation under the price-taking assumption.

Our first result highlights that, due to the cost-minimizing behavior of the operator, the equilibrium outcome uses more on-site generation and pays a lower price to the tenants than the social optimal.

Proposition 10. *Suppose that Assumptions 1 and 2 hold. When tenants are price-taking, the operator running* COOP *uses more on-site generation and pays a lower price for power reduction to its tenants than the social optimal. Specifically,* $y^t \geq y^*$ *and* $\frac{N-1}{N}p^* \leq p^t \leq p^*$.

Proof. Since $y \geq 0$, it suffices to prove that whenever the optimal on-site generation is non-zero, $y^* > 0$, $y^t \geq y^*$. From (5.4), the Lagrangian of SCM is

$$L(\mathrm{s}, y, \mu^*, \lambda^*) = \sum_n c_n(s_n) + \alpha y + \mu^*((\delta - y) - \sum_n s_n) - \lambda^* y.$$

By constraint qualification and the KKT conditions, assuming $y^* > 0$, then $\lambda = 0$, $\mu^* = \alpha$, hence the market clearing price in the optimal allocation should be $p^* = \alpha$.

Next, consider the market price for price taking tenants. From (5.15),

$$p^t = \frac{\sum_{i\in\mathcal{N}} b_i^t}{(N-1)\delta + y^t} = \sqrt{\frac{(\Sigma_{i\in\mathcal{N}} b_i^t)\alpha}{N\delta}}. \tag{5.17}$$

The second equality yields $\sum_{i\in\mathcal{N}} b_i^t = \frac{((N-1)\delta+y^t)^2}{N\delta}\alpha$. Substitute this back to (5.17),

$$p^t = \frac{\sum_{i\in\mathcal{N}} b_i^t}{(N-1)\delta + y^t} = \frac{(N-1)\delta + y^t}{N\delta}\alpha. \tag{5.18}$$

And note that $y_t \in [0, \delta]$ and $p^* = \alpha$, thus (5.18) yields $\frac{N-1}{N}p^* \le p^t \le p^*$.

Finally, from (5.16), the Lagrangian of the price-taking characterization optimization is,

$$L(\mathrm{s}, y, \mu^t, \lambda^t) = \sum_n c_n(s_n) + \frac{\alpha}{2N\delta}(y + (N-1)\delta)^2 + \mu^t((\delta - y) - \sum_n s_n) - \lambda^t y.$$

By examining the KKT condition and using a similar argument to the proof of Proposition 7, we have $p^t = \mu^t$, also, $\frac{\partial^- c_n(s_n^t)}{\partial s_n^t} \le p^t \le p^* \le \frac{\partial^+ c_n(s_n^*)}{\partial s_n^*}$. Thus, $\forall n, s_n^t \le s_n^*$. Since $y = \delta - \sum s_n$, $y^t \ge y^*$. □

Now, we move to more detailed comparisons. There are three components of market efficiency that we consider: social welfare, operator cost, and tenant cost.

First, let us consider the social cost.

Theorem 11. *Suppose that Assumptions 1 and 2 hold. Let* $(\mathbf{s}^t, y^t)$ *be the allocation when tenants are price-taking, and* $(\mathbf{s}^*, y^*)$ *be the optimal allocation. Then the welfare loss is bounded by:* $\sum_n c_n(s_n^t) + \alpha y^t \le \sum_n c_n(s_n^*) + \alpha y^* + \alpha\delta/2N$.

Proof. Note that $(\mathbf{s}^*, y^*)$ is a feasible solution to (5.16). By Theorem 9, we have $\sum_n c_n(s_n^t) + \frac{\alpha}{2N\delta}(y^t + (N-1)\delta)^2 \le \sum_n c_n(s_n^*) + \frac{\alpha}{2N\delta}(y^* + (N-1)\delta)^2$. Rearranging, we have

$$\begin{aligned}
&\sum_n c_n(s_n^t) + \alpha y^t - \left(\sum_n c_n(s^*) + \alpha y^*\right) \le \frac{\alpha}{2N\delta}(y^t - y^*)\left(2\delta - (y^t + y^*)\right) \\
=&\frac{\alpha}{2N\delta}[-(y^t - y^*)^2 + 2(\delta - y^*)(y^t - y^*)] \le \frac{\alpha}{2N\delta}[-(y^t - y^* - (\delta - y^*))^2 + (\delta - y^*)^2] \\
=&\frac{\alpha}{2N\delta}(\delta - y^*)^2 \le \frac{\alpha\delta}{2N}.
\end{aligned}$$

□

Importantly, this theorem highlights that the market equilibrium is quite efficient, especially if the number of tenants is large (the efficiency loss decays to zero as $O(1/N)$). However, the market could maintain good overall social welfare at the expense of either the operator or the tenants. The following results show this is not true.

Let $\text{cost}_o(p, y)$ be the operator's cost, i.e.,

$$\text{cost}_o(p, y) = p(\delta - y) + \alpha y. \tag{5.19}$$

Then, we have the following results.

Theorem 12. *Suppose that Assumptions 1 and 2 are satisfied. The cost of colo operator with price-taking tenants is smaller than the cost in the socially optimal case. Further, we have* $\text{cost}_o(p^*, y^*) - \alpha\delta/N \leq \text{cost}_o(p^t, y^t) \leq \text{cost}_o(p^*, y^*)$.

Proof. From Proposition 10, we have $\frac{N-1}{N}\alpha \leq p^t \leq p^* = \alpha$, and $0 \leq y^t \leq \delta$, which yields:

$$\text{cost}_o^*(p^*, y^*) - \text{cost}_o(p^t, y^t) = p^*(\delta - y^*) + \alpha y^* - \left(p^t(\delta - y^t) + \alpha y^t\right) = (\alpha - p^t)(\delta - y^t).$$

Substituting the above bounds for p^t and y^t gives $0 \leq \text{cost}_o^*(p^*, y^*) - \text{cost}_o(p^t, y^t) \leq \frac{\alpha\delta}{N}$. □

5.3.2 Price-Anticipating Tenants

In contrast to the price-taking model, price-anticipating tenants realize that they can change the market price by their bids, i.e., that p is set according to (5.15), and adjust their bids accordingly. The price-anticipating model is suitable when the market consists of a few dominant players, who have significant power to impact the market price through their bids, i.e., the oligopoly setting. Clearly, this additional strategic behavior can lead to larger efficiency loss. However, in this section, we show that the extra loss is surprisingly small, especially when a large number of tenants participate in COOP.

Given bids from the other tenants, each price-anticipating tenant n optimizes the following cost over bidding value b_n:

$$Q_n(b_n, \mathbf{b}_{-n}) = p(\mathbf{b})S_n(b_n, p) - c_n(S_n(b_n, p)),$$

where we use $\mathbf{b}_{-n}$ to denote the vector of bids of tenants other than n; i.e., $\mathbf{b}_{-n} = (b_1, \ldots, b_{n-1}, b_{n+1}, \ldots, b_N)$. Thus, substituting (5.1) and (5.15), we have

$$Q_n(b_n; \mathbf{b}_{-n}) = \sqrt{\frac{(\Sigma_n b_n)\alpha\delta}{N}} - b_n - c_n\left(\delta - \frac{b_n}{\sqrt{\Sigma_m b_m}}\sqrt{\frac{N\delta}{\alpha}}\right). \tag{5.20}$$

Note that the payoff function Q_n is similar to the payoff function P_n in the price-taking case, except that the tenants anticipate that the colo operator will set the price p according to $p = p(\mathbf{b}, y)$ from (5.15).

Definition 2. *A triple* $(\mathbf{b}, p, y)$ *is a (price-anticipating) market equilibrium if each tenant maximizes its payoff defined in* (5.20)*, the market is cleared by setting the price p according to* (5.2) *and the amount of on-site generation is decided by* (5.3)*, i.e.,*

$$Q_n(b_n; \mathbf{b}_n) \geq Q_n(\bar{b}_n; \mathbf{b}_n) \quad \forall \bar{b}_n \geq 0, \quad n = 1, \ldots, N \tag{5.21}$$

$$p = \frac{\sum_n b_n}{(N-1)\delta + y} \tag{5.22}$$

$$y = \arg\min_{0 \leq y \leq \delta} (\delta - y) \cdot p(\mathbf{b}, y) + \alpha y. \tag{5.23}$$

Note that our analysis in this section requires one additional technical assumption about the tenant cost functions.

Assumption 3. *For all tenants, the marginal cost of energy reduction at 0 is greater than* $\frac{\alpha}{2N}$*, i.e.,* $\frac{\partial^+ c_n(0)}{\partial s_n} \geq \frac{\alpha}{2N}$, $\forall n$.

This assumption is quite mild, especially if the number of tenants N is large. Intuitively, it says that the unit cost of on-site generation is competitive with the cost of tenants reducing their server energy.

5.3.2.1 Market Equilibrium Characterization

Our analysis of market equilibria proceeds along parallel lines to the price-taking case. We again show that there exists a unique equilibrium and, furthermore, that the tenants and operator behave in equilibrium as if they were solving an optimization problem of the same form as the aggregate cost minimization (5.4), but with "modified" cost functions.

Theorem 13. *Suppose that Assumption 1-3 are satisfied, then there exists a unique equilibrium of the game defined by*
$(Q_1, \ldots, Q_n)$ *satisfying* (5.21)-(5.23). *For such an equilibrium, the vector* $\mathbf{s}^a$ *defined by* $s_n^a = S(p(\mathbf{b}^a), b_n^a)$ *is the unique optimal solution to the following optimization:*

$$\min \quad \sum_n \hat{c}_n(s_n) + \frac{\alpha}{2N\delta}(y + (N-1)\delta)^2 \tag{5.24a}$$

$$s.t. \quad \sum_n s_n = \delta - y \tag{5.24b}$$

$$y \geq 0, \; s_n \geq 0, \quad n = 1, \ldots, N, \tag{5.24c}$$

where, for $s_n \geq 0$,

$$\hat{c}_n(s_n) = \frac{1}{2}\left(c_n(s_n) + s_n\frac{\alpha}{2N}\right) + \frac{1}{2}\int_0^{s_n}\sqrt{\left(\frac{\partial^+ c_n(z)}{\partial z} - \frac{\alpha}{2N}\right)^2 + 2\frac{\partial^+ c_n(z)}{\partial z}\frac{z\alpha}{N\delta}}dz, \tag{5.25}$$

and for $s_n < 0, \quad \hat{c}_n(s_n) = 0$.

Proof. The proof proceeds in a number of steps. We first show that the payoff function Q_n is a concave and continuous function for each firm n. We then establish necessary and sufficient conditions for $\mathbf{b}$ to be an equilibrium; these conditions look similar to the optimality conditions (5.10a)-(5.10b) in the proof of Proposition 7, but for a "modified" cost function defined according to (5.25). We then show the correspondence between these conditions and the optimality conditions for the problem (5.24a)-(5.24c). This correspondence establishes existence of an equilibrium, and uniqueness of the resulting allocation.

Step 1: *If* $\mathbf{b}$ *is an equilibrium, and Assumption 2 is satisfied, at least one coordinate of* $\mathbf{b}$ *is positive.*

By Assumption 2, $0 < \alpha < \frac{\Sigma_n b_n}{(N-1)\delta}$, hence at least one coordinate of $\mathbf{b}$ must be positive.

Step 2: *The function* $Q_n(\bar{b}_n; \mathbf{b}_{-n})$ *is concave and continuous in* $\bar{b}_n$, *for* $\bar{b}_n \geq 0$. From (5.20) and by plugging $p(\mathbf{b})$ into s_n in (5.1), we have

$$Q_n(\bar{b}_n; \mathbf{b}_{-n}) = \sqrt{\frac{(\Sigma_{m\neq n} b_m + \bar{b}_n)\alpha\delta}{N}} - \bar{b}_n - c_n\left(\delta - \frac{\bar{b}_n}{\sqrt{\Sigma_{m\neq n} b_m + \bar{b}_n}}\sqrt{\frac{N\delta}{\alpha}}\right).$$

When $\Sigma_{m\neq n} b_m + \bar{b}_n > 0$, the function $\bar{b}_n / \sqrt{\Sigma_{m\neq n} b_m + \bar{b}_n}$ is a strictly concave function of $\bar{b}_n$ (for $\bar{b}_n \geq 0$). Since c_n is assumed to be convex and nondecreasing (and hence continuous), it follows that $Q_n(\bar{b}_n, \mathbf{b}_{-n})$ is concave and continuous in $\bar{b}_n$, for $\bar{b}_n \geq 0$. It is easy to show that for s_n to be positive, we need $b_n \leq \overline{b_n}$ where $\overline{b_n} = \frac{1}{2}\left(\frac{\alpha\delta}{N} + \sqrt{\frac{\alpha\delta}{N}(\frac{\alpha\delta}{N} + 4\Sigma_{m\neq n} b_m)}\right)$.

Step 3: *In an equilibrium,* $0 \leq b_n \leq \overline{b_n}, \forall n$.

Tenant n would never bid more than $\bar{b}_n$ given $\mathbf{b}_{-n}$. If $b_n > \overline{b_n}$, then $S(p(\mathbf{b}), b_n) = \delta - \frac{b_n}{\sqrt{b_n + \Sigma_{m\neq n} b_m}} \frac{N\delta}{\alpha} < 0$, so the payoff $Q_n(b_n; \mathbf{b}_{-n})$ becomes negative; on the other hand, $Q_n(\overline{b_n}; \mathbf{b}_{-n}) = 0$.

We specify the following condition when marginal cost of production is not less than the price:

$$\forall n, \quad \frac{\partial^- c_n(s_n)}{\partial s_n} \leq p(\mathbf{b}), \quad s_n > 0. \tag{5.26}$$

This condition is satisfied when tenants are price-taking; in the next step, we show that (5.26) also holds in an equilibrium outcome when tenants are price-anticipating.

Step 4: *The vector b is an equilibrium if and only if* (5.26) *is satisfied, at least one component of* $\mathbf{b}$ *is positive, and for each n,* $b_n \in [0, \overline{b_n}]$, *and the following conditions hold:*

if $0 < b_n \leq \overline{b_n}$,

$$\frac{1}{2}\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) + \frac{1}{2}\sqrt{\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N}\right)^2 + \frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{2s_n\alpha}{N\delta}} \geq p(\mathbf{b}), \tag{5.27a}$$

if $0 \leq b_n < \overline{b_n}$,

$$\frac{1}{2}\left(\frac{\partial^- c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) + \frac{1}{2}\sqrt{\left(\frac{\partial^- c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N}\right)^2 + \frac{\partial^- c_n(s_n)}{\partial s_n}\frac{2s_n\alpha}{N\delta}} \leq p(\mathbf{b}). \tag{5.27b}$$

By Step 2, $Q_n(b_n; \mathbf{b}_{-n})$ is concave and continuous for $b_n \geq 0$. By Step 3, $b_n \in [0, \overline{b_n}]$. b_n must maximize $Q_n(b_n; \mathbf{b}_{-n})$ over $0 \leq b_n \leq \overline{b_n}$, and satisfy the following first order optimality conditions:

$$\frac{\partial^+ Q_n(b_n; \mathbf{b}_{-n})}{\partial b_n} \leq 0, \quad \text{if } 0 < b_n \leq \overline{b_n},$$
$$\frac{\partial^- Q_n(b_n; \mathbf{b}_{-n})}{\partial b_n} \geq 0, \quad \text{if } 0 \leq b_n < \overline{b_n}.$$

Recalling the expression for $p(\mathbf{b})$ given in (5.15), and noting that by (5.15) and (5.1), we have : $\frac{1}{\sqrt{\Sigma_m b_m}} = \frac{1}{p(\mathbf{b})}\sqrt{\frac{\alpha}{N\delta}}$, and $\frac{b_n}{\sqrt{\Sigma_m b_m}} = (\delta - s_n)\sqrt{\frac{\alpha}{N\delta}}$. Expanding the first order optimality conditions with (5.15) and simplifying with the two equations into the above, we have

$$\frac{1}{2p(\mathbf{b})}\frac{\alpha}{N} - 1 + \frac{\partial^- c_n(s_n)}{\partial s_n}\frac{1}{p(\mathbf{b})}\left(1 - \frac{1}{2p(\mathbf{b})}\frac{\alpha}{N}\frac{\delta - s_n}{\delta}\right) \leq 0. \tag{5.28a}$$

$$\frac{1}{2p(\mathbf{b})}\frac{\alpha}{N} - 1 + \frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{1}{p(\mathbf{b})}\left(1 - \frac{1}{2p(\mathbf{b})}\frac{\alpha}{N}\frac{\delta - s_n}{\delta}\right) \geq 0. \tag{5.28b}$$

To show (5.26) holds, we divide into two cases, when $N \geq 2$, by rearranging (5.28a), we have

$$\frac{\partial^- c_n(s_n)}{\partial s_n}\frac{1}{p(\mathbf{b})} \leq \frac{2Np(\mathbf{b}) - \alpha}{2Np(\mathbf{b}) - \alpha\frac{\delta - s_n}{\delta}} \leq 1.$$

This is because by Assumption 2, $2Np(\mathbf{b}) - \alpha > 0$ when $N \geq 2$. Also, we have $2Np(\mathbf{b}) - \alpha\frac{\delta - s_n}{\delta} \geq 2Np(\mathbf{b}) - \alpha$. Hence (5.26) holds for $N \geq 2$.

When $N = 1$, we can simplify (5.28a) further to

$$\frac{1}{2p(\mathbf{b})}\alpha - 1 + \frac{\partial^- c_n(s_n)}{\partial s_n}\frac{1}{2p(\mathbf{b})} \leq 0, \Rightarrow p(\mathbf{b}) \geq \frac{1}{2}\left(\alpha + \frac{\partial^- c_n(s_n)}{\partial s_n}\right) \geq \frac{\partial^- c_n(s_n)}{\partial s_n}.$$

The last inequality is because $\alpha \geq \frac{\partial^- c_n(s_n)}{\partial s_n}$, otherwise $p(\mathbf{b}) > \alpha$, but the profit maximizing operator will not pay for price more than α, contradiction. Hence (5.26) must hold for all N. After multiplying through (5.28a)-(5.28b) by $p(\mathbf{b})$ and rearranging, we have two quadratic inequalities in terms of $p(\mathbf{b})$. Solving the inequalities leads to two sets of conditions of $p(\mathbf{b})$ that satisfy the first order optimality conditions, they are:

$$\text{if } 0 \leq b_n < \overline{b_n}, \quad \frac{1}{2}\left(\frac{\partial^- c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) \pm \frac{1}{2}\sqrt{\left(\frac{\partial^- c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N}\right)^2 + 4\frac{\partial^- c_n(s_n)}{\partial s_n}\frac{s_n\alpha}{2N\delta}} \leq p(\mathbf{b}), \tag{5.29a}$$

$$\text{if } 0 < b_n \leq \overline{b_n}, \quad \frac{1}{2}\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) \pm \frac{1}{2}\sqrt{\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N}\right)^2 + 4\frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{s_n\alpha}{2N\delta}} \geq p(\mathbf{b}). \tag{5.29b}$$

However, only the conditions with plus signs satisfy (5.26); the conditions with minus signs violate (5.26) because since

$$\forall s_n > 0, \quad p(\mathbf{b}) \leq \frac{\alpha}{2N} \leq \frac{\partial^+ c_n(0)}{\partial s_n} < \frac{\partial^- c_n(s_n)}{\partial s_n}.$$

Hence we discard the conditions with minus signs and note that (5.29b)-(5.29a) corresponds to (5.27a)-(5.27b).

Conversely, suppose that $\mathbf{b}$ has at least one strictly positive component, that $0 \leq b_n \leq \overline{b_n}$, and that $\mathbf{b}$ satisfies (5.26) and (5.27a)-(5.27b). Then we may simply reverse the argument: by Step 2, $Q_n(b_n; \mathbf{b}_{-n})$ is concave and continuous in $b_n \geq 0$, and in this case the conditions (5.27a)-(5.27b) imply that b_n maximizes $Q_n(b_n; \mathbf{b}_{-n})$ over $0 \leq b_n \leq \overline{b_n}$. Since we have already shown that choosing $b_n > \overline{b_n}$ is never optimal for firm n, we conclude that $\mathbf{b}$ is an equilibrium, and it is easy to check that in this case condition (5.26) is satisfied.

Step 5: *If Assumption 2 holds, then the function $\hat{c}_n(s_n)$ defined in* (5.25) *is continuous, and strictly convex and strictly increasing over $s_n \geq 0$, with $\hat{c}(s_n) = 0$ for $s_n \leq 0$.*

$\hat{c}_n(s_n)$ is continuous on $s_n > 0$ by continuity of c_n and on $s_n < 0$ by definition. We only need to show that $\hat{c}_n(0) = 0$, this is because when $s_n = 0$, $c_n(s_n) = 0$, $s_n \frac{\alpha}{2N} = 0$, and integrating from 0 to s_n is 0. Hence $\hat{c}_n(s_n) = 0$ for $s_n \leq 0$.

For $s_n \geq 0$, we simply compute the directional derivatives of $\hat{c}_n$:

$$\frac{\partial^+ \hat{c}_n(s_n)}{\partial s_n} = \frac{1}{2}\left(\frac{\alpha}{2N} + \frac{\partial^+ c_n(s_n)}{\partial s_n}\right) + \frac{1}{2}\sqrt{\left(\frac{\alpha}{2N} - \frac{\partial^+ c_n(s_n)}{\partial s_n}\right)^2 + 2\frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{s_n \alpha}{N\delta}},$$

$$\frac{\partial^- \hat{c}_n(s_n)}{\partial s_n} = \frac{1}{2}\left(\frac{\alpha}{2N} + \frac{\partial^- c_n(s_n)}{\partial s_n}\right) + \frac{1}{2}\sqrt{\left(\frac{\alpha}{2N} - \frac{\partial^+ c_n(s_n)}{\partial s_n}\right)^2 + 2\frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{s_n \alpha}{N\delta}}.$$

Since c_n is strictly increasing and convex, for $0 \leq s_n < \bar{s}_n$, we will have

$$0 \leq \frac{\partial^+ \hat{c}(s_n)}{\partial s_n} < \frac{\partial^- \hat{c}(\bar{s}_n)}{\partial s_n} \leq \frac{\partial^+ \hat{c}(\bar{s}_n)}{\partial s_n}.$$

This guarantees that $\hat{c}_n$ is strictly increasing and strictly convex over $s_n \geq 0$.

Step 6: *There exists a unique vector $\mathbf{s} \geq 0$, $y \geq 0$ and at least one scalar $\rho > 0$ such that:*

$$\frac{1}{2}\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) + \frac{1}{2}\sqrt{\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N}\right)^2 + \frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{2s_n \alpha}{N\delta}} \geq \rho, \quad \text{if } s_n \geq 0; \tag{5.30a}$$

$$\frac{1}{2}\left(\frac{\partial^- c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) + \frac{1}{2}\sqrt{\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N}\right)^2 + \frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{2s_n \alpha}{N\delta}} \leq \rho, \quad \text{if } s_n > 0; \tag{5.30b}$$

$$\frac{\alpha}{N\delta}\left(y + (N-1)\delta\right) = \rho; \tag{5.30c}$$

$$\sum_n s_n = (\delta - y). \tag{5.30d}$$

The vector $\mathbf{s}$ *and* y *is then the unique optimal solution to* (5.24a)-(5.24c).

By Step 5, since $\hat{c}_n$ is continuous and strictly increasing over the convex, compact feasible region for each n, we know that (5.24a)-(5.24c) have a unique optimal solution $\mathbf{s}, y$. As in the proof of Proposition 7, form the Lagrangian

$$L(\mathbf{s}, y; \rho) = \sum_n \hat{c}_n(s_n) + \frac{\alpha}{2N\delta}(y + (N-1)\delta)^2 + \rho((\delta - y) - \sum_n s_n).$$

By assumption 2, $y > 0$, and by the fact that $\hat{c}_n(s_n) = 0$ for $s_n \leq 0$, $s_n \geq 0$, there exists a Lagrange multiplier ρ such that $(\mathbf{s}, y, \rho)$ satisfy the stationarity conditions which corresponds to (5.30a)-(5.30c) when we expand the definition of $\hat{c}_n(s_n)$, together with the constraint (5.30d). $\rho > 0$ follows by (5.30c) as $y > 0$.

Step 7: *If* $\mathbf{s} \geq 0, y \geq 0$ *and* $\rho > 0$ *satisfy* (5.30a)-(5.30d)*, then the triple* $(\mathbf{b}, \rho, y)$ *defined by* $b_n = (\delta - s_n)\rho$ *is an equilibrium as defined in* (5.21) *and* (5.22).

First observe that with this definition, together with (5.30d) and the fact that $s_n \geq 0$, we have $b_n \geq 0$ for all n. Furthermore, we can show $b_n \leq \overline{b_n}$, since $s_n \geq 0$, $b_n \leq \rho\delta$, but by (5.30c)-(5.30d), we have

$$\rho = \frac{\alpha}{N\delta}(y + (N-1)\delta) = \frac{\alpha}{N\delta}(N\delta - \sum_n s_n). \tag{5.31}$$

Substitute the definition $s_n = \delta - \frac{b_n}{\rho}$ into (5.31), we have

$$\rho = \frac{\alpha}{N\delta}\frac{\Sigma_n b_n}{\rho} \Rightarrow \rho = \sqrt{\frac{\Sigma_n b_n \alpha}{N\delta}}. \tag{5.32}$$

Substituting (5.32) into $b_n \leq \rho\delta$, we have $b_n \leq \sqrt{\frac{(\Sigma_{m\neq n} b_m + b_n)\alpha\delta}{N}}$. Solving this inequality we have $b_n \leq \overline{b_n}$.

Finally, at least one component of $\mathbf{b}$ is strictly positive, since otherwise we have $s_{n1} = s_{n2} = \delta$ for some $n1 \neq n2$, in which case $\Sigma_n s_n > \delta$, which contradicts (5.30d). (Or $s_n = \delta$, $y = 0$, contradicting our assumption that $y > 0$.)

By Step 4, to check that $\mathbf{b}$ is an equilibrium, we must only check the stationarity conditions (5.27a)-(5.27b). We simply note that under the identification $b_n = \rho(\delta - s_n)$, using (5.32) and (5.30c), we have

$$y = \sqrt{\frac{\Sigma_n b_n N\delta}{\alpha}} - (N-1)\delta; \quad \rho = \frac{\Sigma_n b_n}{(N-1)\delta + y} = p(\mathbf{b}).$$

Substituting $p(\mathbf{b})$ into (5.30a) will correspond to (5.27a), and (5.30b) implies (5.27b) and (5.26) because $\frac{\partial^- c_n(s_n)}{\partial s_n} \leq \frac{\partial^+ c_n(s_n)}{\partial s_n}$. Thus $(\mathbf{b}, \rho, y)$ is an equilibrium.

Step 8: *If (*$\mathbf{b}, p(\mathbf{b}), y$*) is an equilibrium, then there exists a scalar* $\rho \geq 0$ *such that the vector* $\mathbf{b}$ *defined by* $s_n = S(p(\mathbf{b}), b_n)$ *satisfies* (5.30a)-(5.30d).

We simply reverse the argument of Step 7. Since $\mathbf{b}$ is the bids in the equilibrium, by (5.22) and $s_n = S(p(\mathbf{b}), b_n)$, we have $\sum_n s_n = (\delta - y)$, i.e., (5.30d) is satisfied. By Step 4, $\mathbf{b}$ satisfies (5.27a)-(5.27b). Since $y > 0$ by Assumption 2, $0 \leq s_n < \delta$ for all n, let

$$\rho = \max\left\{p(\mathbf{b}), \frac{1}{2}\left(\frac{\partial^- c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}\right) + \frac{1}{2}\sqrt{(\frac{\partial^+ c_n(s_n)}{\partial s_n} - \frac{\alpha}{2N})^2 + \frac{\partial^+ c_n(s_n)}{\partial s_n}\frac{2s_n\alpha}{N\delta}}\right\}.$$

In this case $\rho > 0$ and $0 \leq b_n \leq \overline{b_n}$ for all n, so (5.27b) implies (5.30b) by definition of ρ, and (5.30a) holds by (5.27a) and the fact that $\partial^- c_n(s_n) \leq \partial^+ c_n(s_n)$ (by convexity).

Step 9: *There exists an equilibrium* $\mathbf{b}$*, and for any equilibrium that price is greater than marginal cost, the vector* $\mathbf{s}$ *defined by* $s_n = S(p(\mathbf{b}), b_n)$ *is the unique optimal solution of* (5.30a)-(5.30d).

The conclusion is now straightforward. Existence follows from Steps 6 and 7. Uniqueness of the resulting production vector $\mathbf{s}$, and the fact that $\mathbf{s}$ is an optimal solution to (5.24a)-(5.24c), follows by Steps 6 and 8.

□

Although the form of $\hat{c}_n(s_n)$ looks complicated, there is a simple linear approximation that gives useful intuition.

Lemma 14. *Suppose that Assumptions 1-3 are satisfied. For all modified cost* $\hat{c}_n, n \in 1, \ldots, N$*, for any* $0 \leq s_n \leq \delta$*,*

$$c_n(s_n) \leq \hat{c}_n(s_n) \leq c_n(s_n) + s_n\frac{\alpha}{2N}.$$

Furthermore, when the left or right derivatives of $\hat{c}(\cdot)$ *are defined, it can be bounded by*

$$\frac{\partial^- c_n(s_n)}{\partial s_n} \leq \frac{\partial^- \hat{c}(s_n)}{\partial s_n} \leq \frac{\partial^+ \hat{c}(s_n)}{\partial s_n} \leq \frac{\partial^+ c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}.$$

Proof. We exploit the structure of the modified cost $\hat{c}_n$ to prove the result. Note that, for all n, $s_n \geq 0$, if we define $G_n(s_n) = \int_0^{s_n} \sqrt{(\frac{\partial^+ c_n(z)}{\partial z} - \frac{\alpha}{2N})^2 + \frac{\partial^+ c_n(z)}{\partial z}\frac{2z\alpha}{N\delta}}dz$, then

$$G_n(s_n) \geq \int_0^{s_n} \sqrt{\left(\frac{\partial^+ c_n(z)}{\partial z} - \frac{\alpha}{2N}\right)^2} dz = c_n(s_n) - s_n\frac{\alpha}{2N}.$$

The first inequality is because $z \geq 0$, and the last equality is because by convexity and Assumption 3. We have $\frac{\partial^+ c_n(z)}{\partial z} \geq \frac{\partial^+ c_n(0)}{\partial s_n} \geq \frac{\alpha}{2N}$.

Hence we have $\hat{c}_n(s_n) = \frac{1}{2}\left(c_n(s_n) + s_n\frac{\alpha}{2N}\right) + \frac{1}{2}G_n(s_n) \geq c_n(s_n)$. On the other hand, notice that $s_n \leq \delta$, we have:

$$\begin{aligned} G_n(s_n) &\leq \int_0^{s_n} \sqrt{\left(\frac{\partial^+ c_n(z)}{\partial z} - \frac{\alpha}{2N}\right)^2 + \frac{\partial^+ c_n(z)}{\partial z}\frac{2\delta\alpha}{N\delta}}dz \\ &= \int_0^{s_n} \sqrt{\left(\frac{\partial^+ c_n(z)}{\partial z} + \frac{\alpha}{2N}\right)^2} dz = c_n(s_n) + s_n\frac{\alpha}{2N}. \end{aligned}$$

Hence we have $\hat{c}_n(s_n) = \frac{1}{2}\left(c_n(s_n) + s_n\frac{\alpha}{2N}\right) + \frac{1}{2}G_n(s_n) \leq c_n(s_n) + s_n\frac{\alpha}{2N}$. The bounds for the left and right derivatives can be obtained by taking the left (or right) derivatives at the bounds of $G_n(s_n)$. □

The form of Lemma 14 shows that the difference between the modified cost function in (5.25) and the true cost diminishes as N increases, and this is the key observation that underlies our subsequent results upper bounding the efficiency loss of COOP.

5.3.2.2 Bounding Efficiency Loss

We now use Theorem 13 to bound the efficiency loss due to strategic behavior. Note that, by comparing to both the socially optimal and the price-taking outcomes, we can understand the impact of both strategic behavior by the operator and by the tenants.

Our first result focuses on comparing the price-anticipating and price-taking equilibrium outcomes. It highlights that price-anticipating behavior leads to tenants receiving higher price while shedding less load.

Theorem 15. *Suppose Assumption 1-3 hold. Let (p^t, y^t) be the equilibrium price and on-site generation when tenants are price-taking, and (p^a, y^a) be those when tenants are price-anticipating, then we have, $y^t \leq y^a \leq y^t + \delta/2$ and $p^t \leq p^a \leq p^t + \alpha/2N$.*

Proof. Firstly we will prove one side of the inequality $p^t \leq p^a, y^t \leq y^a$. Recall that by examining the Lagrangians of the optimizations in Proposition 10 and Theorem 13, we have $p^t \geq \partial^- c_n(s_n^t)/\partial s_n$, $p^t \leq \partial^+ c_n(s_n^t)/\partial s_n$, $p^a \geq \partial^- \hat{c}_n(s_n^a)/\partial s_n$, $p^a \leq \partial^+ \hat{c}_n(s_n^a)/\partial s_n$, at the domain where the left or right derivative is defined, and $p^t = \frac{\alpha}{N\delta}(y^t + (N-1)\delta), p^a = \frac{\alpha}{N\delta}(y^a + (N-1)\delta)$. If $y^t > y^a$, then $p^t > p^a$. Also, because the total energy reduction δ is constant, we have $\sum_n s_n^t < \sum_n s_n^a$.

Hence there exist $s_r > 0$ such that $s_r^a > s_r^t$ for some $r \in \{1, \ldots, N\}$. Therefore, by strict convexity of c_n (Assumption 1):

$$p^t \leq \frac{\partial^+ c_r(s_r^t)}{\partial s_r} < \frac{\partial^- c_r(s_r^a)}{\partial s_r}. \tag{5.33}$$

However, by Lemma 14 we have $\frac{\partial^- \hat{c}_r(s_r)}{\partial s_r} \geq \frac{\partial^- c_r(s_r)}{\partial s_r}$. Hence, we have

$$p^a \geq \frac{\partial^- \hat{c}_r(s_r^a)}{\partial s_r} \geq \frac{\partial^- c_r(s_r^a)}{\partial s_r}. \tag{5.34}$$

Combining (5.33) and (5.34), we have $p^t < p^a$, contradiction. Hence we have $y^t \leq y^a$, and $p^t \leq p^a$.

Next we show the other side of the inequality $p^a \leq p^t + \frac{\alpha}{2N}, y^a \leq y^t + \frac{\delta}{2}$; by the previous part, we have $\sum_n s_n^a \leq \sum_n s_n^t$.

Let $n = \arg\max_m(s_m^t - s_m^a)$, clearly $s_n^t \geq s_n^a$, otherwise $\sum_n s_n^t < \sum_n s_n^a$, contradiction.

If $s_n^t = s_n^a$, then $\forall m, s_m^t = s_m^a$, and $y^t = y^a$, then $p^t = p^a$.

If $s_n^t > s_n^a$, then by strict convexity of c_n (assumption 1), and the fact that $s_n^a \geq 0, s_n^t > 0$, we have $\frac{\partial^+ \hat{c}_n(s_n^a)}{s_n} < \frac{\partial^- c_n(s_n^t)}{s_n} \leq p^t$. Also, by Lemma 14, we have $\frac{\partial^+ \hat{c}_n(s_n)}{\partial s_n} \leq \frac{\partial^+ c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}$, this gives us $p^a \leq \frac{\partial^+ \hat{c}_n(s_n^a)}{\partial s_n} \leq \frac{\partial^+ c_n(s_n^a)}{\partial s_n} + \frac{\alpha}{2N}$. Combining the two previous inequalities about p^t and p^a, we have $p^a < p^t + \frac{\alpha}{2N}$. Hence we have

$$\frac{\alpha}{N\delta}(y^a + (N-1)\delta) < \frac{\alpha}{N\delta}(y^t + (N-1)\delta) + \frac{\alpha}{2N} \Rightarrow y^a < y^t + \frac{\delta}{2}.$$

□

Next, combining Theorem 15 and Proposition 10 yields the following comparison between the price-anticipating and socially optimal outcomes.

Corollary 16. *Suppose Assumptions 1-3 hold. When tenants are price-anticipating, an operator running COOP uses more on-site generation and pays lower market price than in the socially optimal case, i.e., $y^a \geq y^*$ and $\frac{N-1}{N}p^* \leq p^a \leq p^*$.*

Now, we move to more detailed comparisons. There are three components of market efficiency that we consider: social cost, operator cost, and tenant cost.

First, let us consider the social cost.

Theorem 17. *Suppose that Assumptions 1-3 hold. Let* $(\mathbf{s}^a, y^a)$ *be the allocation when tenants are price-anticipating, and* $(\mathbf{s}^*, y^*)$ *be the optimal allocation. The welfare loss is bounded by:* $\sum_n c_n(s_n^a) + \alpha y^a \leq \sum_n c_n(s_n^*) + \alpha y^* + \alpha\delta/N$.

Proof. As $(\mathbf{s}^*, y^*)$ is a feasible solution to (5.24), by Theorem 13, we have

$$\sum_n \hat{c}_n(s_n^a) + \frac{\alpha}{2N\delta}(y^a + (N-1)\delta)^2 \leq \sum_n \hat{c}_n(s_n^*) + \frac{\alpha}{2N\delta}(y^* + (N-1)\delta)^2. \quad (5.35)$$

Rearranging, we have $\sum_n \hat{c}_n(s_n^a) + \alpha y^a - \left(\sum_n \hat{c}_n(s_n^*) + \alpha y^*\right) \leq \frac{\alpha}{N}\left((y^a - y^*)(1 - \frac{y^a + y^*}{2\delta})\right)$. By Corollary 16 and the fact that $y^* \leq \delta, y^a \leq \delta$, both terms in the brackets are positive, hence the right-hand-side expression is maximized when $y^* \to 0^+$ and $y^a = \delta$, hence

$$\left(\sum_n \hat{c}_n(s_n^a) + \alpha y^a\right) - \left(\sum_n \hat{c}_n(s_n^*) + \alpha y^*\right) \leq \frac{\alpha\delta}{2N}. \quad (5.36)$$

However, by Lemma 14, we have $\sum_n \hat{c}_n(s_n^*) \leq \sum_n c_n(s_n^*) + \frac{\alpha}{2N}(\sum_n s_n) \leq \sum_n c_n(s_n^*) + \frac{\alpha\delta}{2N}$; and $\sum_n \hat{c}_n(s_n^a) \geq \sum_n c_n(s_n^a)$. Substituting the above relations into (5.36) and rearranging, we have the desired result. □

Similarly to the price-taking case, the efficiency loss in the price-anticipating case decays to zero as $O(1/N)$, only with a larger constant. Also, as in the case of price-taking tenants, we again see that neither the tenants nor the operator suffers significant efficiency loss.

Theorem 18. *Suppose that Assumptions 1-3 hold. The cost of colo operator for price-anticipating tenants is smaller than the cost in the socially optimal case. Further, we have*

$$\text{cost}_o(p^*, y^*) - \frac{\alpha\delta}{N} \leq \text{cost}_o(p^a, y^a) \leq \text{cost}_o(p^*, y^*),$$
$$\text{cost}_o(p^a, y^a) - \frac{\alpha\delta}{N} \leq \text{cost}_o(p^t, y^t) \leq \text{cost}_o(p^a, y^a).$$

Proof. First, we compare the cost by operator between the price-taking and price anticipating cases, by definition (5.19) and rearranging, we have $\text{cost}_o(p^a, y^a)$ –

$\text{cost}_o(p^t, y^t) = (p^a - p^t)\left(\delta - y^t\right) + (\alpha - p^a)(y^a - y^t)$. By the fact that $p^a = \frac{\alpha}{N\delta}(y^a + (N-1)\delta)$ (shown in Theorem 15) and the fact that $0 \le y^a \le \delta$, we have

$$\alpha\left(\frac{N-1}{N}\right) \le p^a \le \alpha. \tag{5.37}$$

By the upper bound of p^a in (5.37) and the upper bounds of p^t, y^t in Theorem 15, we have

$$\text{cost}_o(p^a, y^a) - \text{cost}_o(p^t, y^t) \ge 0. \tag{5.38}$$

Similarly, using the lower bound of p^a in (5.37) and the upper bounds of p^a, y^a in Theorem 15, we have

$$\text{cost}_o(p^a, y^a) - \text{cost}_o(p^t, y^t) \le \left(\frac{\alpha}{2N}\right)\cdot(\delta) + \left(\alpha \cdot \frac{1}{N}\right)\left(\frac{\delta}{2}\right) = \frac{\alpha\delta}{N}.$$

Second, we compare the cost by the operator to the social optimal. Since the energy reduction goal δ is the same, by Proposition 10 and Corollary 16, we have $p^t \le p^*$ and $p^a \le p^*$. Hence we have $\text{cost}_o(p^t, y^t) \le \text{cost}_o(p^a, y^a) \le \text{cost}_o(p^*, y^*)$. Furthermore,

$$\begin{aligned}\text{cost}_o(p^*, y^*) - \text{cost}_o(p^t, y^t) &= \alpha\delta - (p^t(\delta - y^t) + \alpha y^t)\\ =(\alpha - p^t)(\delta - y^t) &= \alpha\left(\frac{\delta - y^t}{N\delta}\right)(\delta - y^t) \le \frac{\alpha\delta}{N}.\end{aligned} \tag{5.39}$$

Lastly by (5.38) and (5.39), we have $\text{cost}(p^*, y^*) - \text{cost}(p^a, y^a) \le \text{cost}(p^*, y^*) - \text{cost}(p^t, y^t) \le \frac{\alpha\delta}{N}$. □

Finally, let us end by considering the amount of on-site generation used in equilibrium. Here, in the worst-case, the on-site generation at equilibrium for price-anticipating tenants can be arbitrarily worse than the socially optimal, i.e., the socially optimal can use no on-site generation while the equilibrium outcome uses only on-site generation.

Theorem 19. *Suppose that Assumptions 1-3 hold. For any $\varepsilon > 0$, $N \ge 1$, there exist cost functions $c_1, \ldots, c_N$, such that the on-site generation in the market equilibrium compared to the optimal is given by $y^a - y^* \ge \delta - \varepsilon$.*

Proof. Given any $\varepsilon > 0$, let $\varepsilon' = \frac{1}{2}\varepsilon$. Consider the following set of cost functions:

$$c_1(s_1) = \begin{cases} \frac{\alpha}{2N}s_1, & \text{if } s_1 < \varepsilon';\\ \alpha(1 - \frac{3\varepsilon'}{2N\delta})s_1 + C_1, & \varepsilon' \le s_1 \le \delta - \varepsilon';\\ 2\alpha s_1 + C_2, & s_1 > \delta - \varepsilon', \end{cases}$$

where C_1, C_2 are constants that make c_1 continuous[6], then c_1 is piece-wise linear and convex. Also, $\forall m \neq 1, c_m(s_m) = 2\alpha s_m$. It is easy to see that $s_1^* = \delta - \varepsilon'$ and $y^* = \varepsilon'$ is the optimal allocation.

Let $s_1^a = \varepsilon', y^a = \delta - \varepsilon'$, and $\forall m \neq 1, s_m^a = 0$, we claim that $(\mathbf{s}^a, y^a)$ is the unique optimal solution to (5.24a)-(5.24c). To see this, let $\rho = \alpha(1 - \varepsilon/(N\delta))$, then,

$$\frac{\alpha}{N\delta}(y^a + (N-1)\delta) = \rho; \quad \sum_n s_n^a = \delta - y^a; \tag{5.40a}$$

$$\frac{\partial^- \hat{c}_1(s_1^a)}{\partial s_1} \leq \rho; \quad \frac{\partial^+ \hat{c}_1(s_1^a)}{\partial s_1} \geq \rho; \quad \frac{\partial^+ \hat{c}_m(0)}{\partial s_m} \geq \rho, \quad \forall m \neq 1, \tag{5.40b}$$

where the second inequality is because if we let H_n be the term under square root for $\frac{\partial^+ \hat{c}_n(s_n)}{\partial s_n}$, then

$$\begin{aligned} H_n &= \sqrt{\left(\frac{\partial^+ c_n(s_n)}{\partial s_n} - (\frac{\alpha}{2N} - \frac{\alpha}{N}\frac{s_n}{\delta})\right)^2 + (\frac{\alpha^2}{N^2}\frac{(\delta + s_n)(\delta - s_n)}{\delta^2})} \\ &\geq \frac{\partial^+ c_n(s_n)}{\partial s_n} - (\frac{\alpha}{2N} - \frac{\alpha}{N}\frac{s_n}{\delta}). \end{aligned}$$

Note that $\frac{\partial^+ \hat{c}_n(s_n)}{\partial s_n} = \frac{1}{2}(\frac{\partial^+ c_n(s_n)}{\partial s_n} + \frac{\alpha}{2N}) + \frac{1}{2}H_n$. Hence we have $\frac{\partial^+ \hat{c}_1(s_1^a)}{\partial s_1} \geq \frac{\partial^+ c_1(s_1^a)}{\partial s_1} + \frac{\alpha s_1}{2N\delta} = \rho$. These conditions correspond to (5.30a)-(5.30d), so we conclude that $(\mathbf{s^a}, y^a)$ is the unique optimal solution to (5.24a)-(5.24c). Hence $y^a - y^* = \delta - 2\varepsilon' = \delta - \varepsilon$. □

This is a particularly disappointing result since a key goal of the mechanism is to obtain load shedding from the tenants. However, the proof emphasizes that this is unlikely to occur in practice. In particular, the worst-case scenario is that there exists a dominant (monopoly) tenant, which is unlikely in a multi-tenant colo, that has a cost function asymptotically linear with unit cost roughly matching the on-site generation price α.

5.3.3 Discussion

The main results for the price-taking and price-anticipating analyses are summarized in Table 5.2. Note that simplified bounds are presented in the table, to ease interpretation, and the interested reader should refer to the theorems in Section 5.3.1 and Section 5.3.2 for the actual bounds. Also, note that the benchmark for social cost we consider is an ideal, but not achievable, mechanism.

[6] $C_1 = -\alpha\varepsilon'(\frac{(2N-1)\delta - 3\varepsilon'}{2N\delta})$, and $C_2 = -\frac{\alpha}{N\delta}(N\delta^2 + \delta\varepsilon' - 3\varepsilon')$

Tenants	Price Ratio	Colo Saving	Welfare Loss
Price-taking	$[\frac{N-1}{N},\ 1]$	$[0,\ \alpha\delta/N]$	$[0,\ \alpha\delta/2N]$
Price-anticipating	$[\frac{N-1}{N},\ 1]$	$[0,\ \alpha\delta/N]$	$[0,\ \alpha\delta/N]$

Table 5.2: Performance guarantee of COOP compared to the social optimal allocation.

To summarize the results in Table 5.2 briefly, note first that COOP always benefits the operator, since the price paid to tenants to reduce energy is always less than the socially optimal price, and the total cost incurred by operator for energy reduction is also less than that of the social optimal. Secondly, COOP also gives the tenants approximately the social optimal payment, since the operator's additional benefit is bounded above by $\alpha\delta/N$, and the welfare loss is bounded above by $\alpha\delta/N$. This naturally means that the loss in payment for tenants compared to the social optimal is at most $2\alpha\delta/N$, which approaches 0 as N grows. Third, regardless of tenants being price-taking or price-anticipating, COOP is approximately socially cost minimizing as the number of tenants grows.

However, while COOP is good in terms of operator, tenant, and social cost, it may not use the most environmentally friendly form of load reduction: in the worst case, the upper bound on the extra on-site generation that COOP uses is not decreasing with N. However, the analysis highlights that this worst-case occurs when there exists a dominant tenant with unit cost of energy reduction that is consistently just below the cost of diesel over a large range of energy reduction, which is unlikely to occur in practice. So, COOP can be expected to use an environmentally friendly mix in most realistic situations.

5.4 COOP with Multi-level Power Constraints

We describe the design of COOP and provide the efficiency bounds with respect to social optimal allocation for both price taking tenants and price anticipating tenants in Section 5.3 and Section 5.3.2. In this section, we extend our design to multi-level power capping in the data center. Adding multi-level power capping constraints increases the complexity of the analysis of the efficiency bound dramatically. Thus we do not provide a theoretical analysis. We use a case study in Section 5.6 to show the efficiency of our algorithm.

5.4.1 Problem Formulation

A data center typically oversubscribes capacity at multiple interdependent power hierarchies (e.g., data center UPS-level, cluster PDU-level, and even rack-level), each having its own capacity below which the involved tenants' aggregate power demand should be capped at all times [34, 183]. COOP is not restricted to any particular levels. Like prior research [34], we consider the most typical two-level power oversubscription, i.e., cluster PDU-level and data center UPS-level, referred to as low and high levels, respectively. For ease of presentation, we treat the presence of a diesel generator as a special tenant with linear cost function in the following discussion.

Model. Consider a power emergency that involves a centralized UPS supporting M cluster PDUs and a total of N tenants denoted by a set $\mathcal{N}_0 = \{1, 2, \cdots, N\}$. The i-th PDU supplies power to a subset of tenants $\mathcal{N}_i \subseteq \mathcal{N}_0$, and the tenants served by two different PDUs are non-overlapping (i.e., $\cup_{i=1}^{M} \mathcal{N}_i = \mathcal{N}_0$ and $\mathcal{N}_i \cap \mathcal{N}_j = \varnothing$ if $i \neq j$). The high-level UPS capacity is exceeded by $D_0 \geq 0$, while the i-th low-level PDU capacity is exceeded by $D_i \geq 0$. Suppose that tenant i cuts power by s_i and incurs a cost of $c_i(s_i)$ that is increasing in s_i. Cutting power may result in service quality or performance degradation, and the cost $c_i(s_i)$ can therefore be interpreted as the performance cost, which converts the performance degradation into a monetary value. The function $c_i(s_i)$ is decided at the tenant's sole discretion as its *private* information that is unknown to the operator.

Objective. Like power capping for owner-operated data centers [33, 34], we consider an *equivalent* objective: minimizing tenants' overall performance loss, formalized below,

$$\min_{s_i \geq 0, i=1,2,\cdots,N} \sum_{i=1}^{N} c_i(s_i) \tag{5.41}$$
$$s.t., \quad \sum_{i \in \mathcal{N}_j} s_i \geq D_j, \text{ for } j = 0, 1, 2, \cdots, M,$$

where the objective of (5.41) is a scalar measure of overall performance loss and impact on tenants, and the constraint specifies power capping requirements at the high (D_0) and low (D_j for $j = 1, \cdots, M$) levels, respectively.

Tenants typically test power-performance profiles before production deployment, since power is a major cost for tenants' leasing [34, 178]. Thus, given its own traffic load, a tenant knows how much power can be shed and at what cost. If they are

uncertain at runtime (due to, e.g., changes in power profiles), tenants can evaluate costs conservatively (see Section 5.6.6); hence, repeated profiling of $c_i(s_i)$ at runtime is not necessary, and the overhead for the participating tenant is small.

The ideal case is when the operator can directly minimize the cost in (5.41), with full control over tenants' servers as in an owner-operated data center. We refer to the outcome of this idealized, but not feasible in practice, case as OPT. The choice of objective in (5.41) may seem counterintuitive, so let us discuss it briefly. One might expect to have the objective be operator profit. However, the operator has a priority of minimizing the impact of power capping on tenants' operation during an emergency since it is the operator's *fault* (due to oversubscription) the emergency occurred. This objective is consistent with prior power capping research on owner-operated data centers [33, 34] and, further, in our context, if the operator still attempts to make profits during an emergency, power outage risk may increase, which is unacceptable since downtime incidents will significantly damage the operator's business image as well as its long-term profit. Additionally, note that the operator will not lose profit during emergency events since it can always set an upper bound (according to Table 5.1) to ensure that it will not lose profit due to oversubscription while minimizing tenant impact.

5.4.2 A Market-Based Solution

We extend our design of COOP in Section 5.2.1 to multi-level power capping with some simple changes. Instead of setting a uniform supply function across tenants, we differentiate tenants based on their maximum possible power reductions and consider a parameterized supply function $s_i(b_i, p) = \left[\delta_i - \frac{b_i}{p}\right]^+$, where δ_i with a unit of kW indicates tenant i's maximum possible power reduction, b_i is its bid (with a unit of \$) and p is the reward/price (\$ per kW) offered by operator to all tenants. The sign "+" in the supply function indicates that tenant cannot supply negative power (i.e., increase power).

The supply function $s_i(b_i, r) = \left[\delta_i - \frac{b_i}{p}\right]^+$ indicates tenant i's willingness to reduce its power by $s_i(b_i, p)$ if the operator offers p for each kW reduction. The actual power reduction is jointly determined by the following sequence.

Step 1: Operator decides δ_i. The data center operator decides δ_i and announces the form of supply function $s_i(b_i, p) = \left[\delta_i - \frac{b_i}{p}\right]^+$ to tenants by signalling to tenants' server control interfaces. Tenant i's current power usage can be set as its maximum

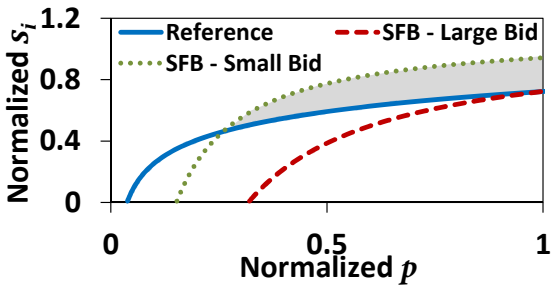


Figure 5.3: Illustration of tenant's bidding.

possible power reduction δ_i (i.e., power reduction if tenant i shuts down all its servers).

Step 2: Tenant decides b_i. With the price p as an unknown variable, tenant i individually chooses and submits a bid b_i to the operator. Essentially, tenant i reports to the operator its power reduction flexibility: *if* offered a price of p, then it will cut power by $s_i(b_i, p)$. In other words, given b_i, the actual power reduction is still a function of the variable p. (We will discuss how to choose b_i later.)

Step 3: Operator decides r. Once the operator receives tenants' bids, it needs to decide p (called market clearing price), which is plugged into $s_i(b_i, p) = \left[\delta_i - \frac{b_i}{p}\right]^+$ to determine tenant i's power reduction.

How to choose bid b_i? Tenants have the full discretion to decide their own bids. We first illustrate the impact of the bid on tenant's power reduction in Figure 5.3. As a *reference*, we also plot the tenant's maximum power reduction without losing profit: the maximum power reduction s_i such that tenant i's net profit, i.e., operator's payment minus tenant's private performance cost, is non-negative. Given a price o, reducing more power than this reference value will incur a profit loss for tenants. We see from both $s_i(b_i, p) = \left[\delta_i - \frac{b_i}{p}\right]^+$ and Figure 5.3 that a larger b_i means that tenant i is less willing to cut power given the same price p. We also notice that a too-small bid may result in a profit loss when tenants are offered higher prices (i.e., shaded area in Figure 5.3).

Similar to the data center level constraint only case, an expected outcome is the *equilibrium* point, at which each tenant i maximizes its net profit "$p \cdot s_i - c_i(s_i)$," thus having no incentives to choose arbitrarily high bids and representing a stabilized outcome.

Setting too large a bid deviates from an equilibrium point, since tenant will be *priced out* or only asked to reduce a small amount of power when other participating

tenants can reduce power at lower prices. For example, if $b_i \to \infty$, tenant i will be excluded from the mechanism without being asked to reduce any power, i.e., $s_i(b_i, p) = \left[\delta_i - \frac{b_i}{p}\right]^+ = 0$.

Tenants have the discretion to decide their bids, but the final price is set by the operator (which determines the actual power reduction for each tenant) and rational tenants will bid reasonably based on their private costs $c_i(s_i)$. One bidding strategy is that b_i is just large enough to avoid net profit loss over a price range (i.e., as illustrated in dashed line in Figure 5.3).

To guide tenants' bidding towards the equilibrium, the operator can tell the tenants its expected price range (e.g., market price p will only be within $[p_{\min}, p_{\max}]$), such that tenants can bid to avoid profit loss by considering this restricted price range rather than the entire range.

How to decide price p? Given tenants' bids, the operator's goal is to set price p as low as possible, while satisfying *all* the power capping constraints. It is clear that, to ensure $\sum_{i\in\mathcal{N}_j} s_i \geq D_j$, the price p needs to satisfy $\sum_{i\in\mathcal{N}_j} s_i(b_i, p) = \sum_{i\in\mathcal{N}_j} \left[\delta_i - \frac{b_i}{p}\right]^+ \geq D_j$. Thus, the market price p can be decided as

$$p = \min_{p'}\{p' \in [p_{\min}, p_{\max}] \mid \sum_{i\in\mathcal{N}_j} s_i(b_i, p') = \sum_{i\in\mathcal{N}_j} [\delta_i - \frac{b_i}{p'}]^+ \geq D_j, \text{ for } j = 0, 1, \cdots, M\},$$

i.e., the minimum price that satisfies all the power capping constraints and is within the range $[p_{\min}, p_{\max}]$. If no such price exists, the operator needs to activate the *failover* mode (see Section 5.4.4).

Scalability. COOP is highly scalable, as determination of the bid is performed by individual tenants in parallel and the market price is decided based on a simple rule $p = \min_{p'}\{p' \in [p_{\min}, p_{\max}] \mid \sum_{i\in\mathcal{N}_j} s_i(b_i, p') = \sum_{i\in\mathcal{N}_j} [\delta_i - \frac{b_i}{p'}]^+ \geq D_j, \text{ for } j = 0, 1, \cdots, M\}$. In practice, there are at most a few tens of tenants in wholesale data centers, and typically no more than a few hundreds of tenants in retail data centers. In either case, the complexity of COOP is reasonably low (further shown in Section 5.6).

5.4.3 Implementation

To implement COOP, we introduce a set of new APIs, for both the operator and tenants, as illustrated in Figure 5.4, where new APIs are inside shaded boxes. The system flow is also described in Algorithm 6, where we consider a general two-level

1: Input: UPS and PDU capacities P_i^{cap} for $i = 0, 1, \cdots, M$
2: Monitor UPS and PDU power $P_i(t)$ continuously.
3: **if** $P_i(t) > P_i^{cap}$ for any $i = 0, 1, \cdots, M$ **then**
4: Start waiting timer T_w
5: **while** T_w has not expired **do**
6: **if** $P_i(t) \leq P_i^{cap}$ for all $i = 0, 1, \cdots, M$ **then**
7: Go back to Line 2
8: ▷ *Entering "power capping" mode*
9: **if** $P_i(t) > P_i^{cap}$ for any $i = 0, 1, \cdots, M$ **then**
10: Set $D_i \leftarrow \left[P_i(t) - P_i^{cap}\right]^+$
11: Announce $s_i(b_i, r) = [\delta_i - \frac{b_i}{r}]^+$ to tenant i
12: Tenant i decides its bid b_i
13: Set price $r = \min_{r'}\{r' \in [r_{\min}, r_{\max}] \mid \sum_{i \in \mathcal{N}_j} s_i(b_i, r') \geq D_j, \text{ for } j = 0, 1, \cdots, M\}$
14: Each tenant i reduces $s_i(b_i, r)$ power
15: ▷ *Leaving "power capping" mode*
16: **wait** until $P_i(t) \leq P_i^{cap} - D_i$ for all $i = 0, 1, \cdots, M$
17: Start capping timer T_c and **wait** until T_c expires or $P_i(t) > P_i^{cap} - D_i$ for any $i = 0, 1, \cdots, M$
18: **if** $P_i(t) > P_i^{cap} - D_i$ for any $i = 0, 1, \cdots, M$ **then**
19: Go back to Line 16
20: **if** T_c expires **then**
21: Notify tenants to resume normal operation
22: Calculate the power capping duration T_o
23: Provide tenant i with a reward of $z_i = T_o \cdot r \cdot s_i$
24: Go back to Line 2

Pseudocode 6: COOP: Coordinated Power Management

capping. Note that, if only a few low-level PDUs are overloaded without exceeding the high-level UPS capacity shared with other non-overloaded PDUs, then the operator will only notify tenants served by these overloaded PDUs to participate in COOP.

• **Detecting power emergency.** The operator monitors power by accessing power meter API `Power( )` at runtime, which is already in place in multi-tenant data centers. While short-duration load spikes (e.g., a few seconds) can be tolerated by the infrastructure itself [33, 168], a *sustained* power emergency of capacity overloading should invoke the power capping mode and execute COOP. The time threshold, i.e., T_w in Algorithm 6, for deciding power emergency depends on how much the aggregate demand exceeds the capacity: if not too much, a larger T_w (e.g., a few tens of seconds) is used; and vice versa.

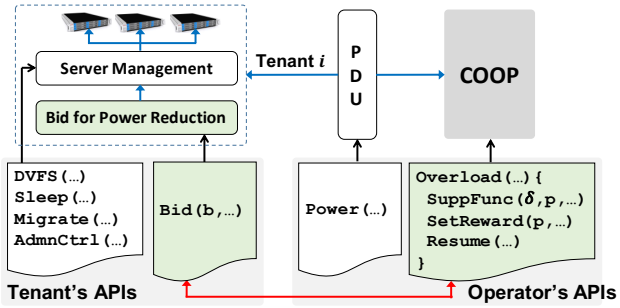


Figure 5.4: API diagram for COOP.

• **Executing market mechanism.** Upon a power emergency, the market mechanism is executed following the steps described in Section 5.4.2 using new APIs. Specifically, the operator communicates the supply function to tenants through `SuppFunc`$(\delta, p, \cdots)$ where the price p is a parameter to be decided, and the tenant decides its bid and submits it to the operator through `Bid`$(b, \cdots)$. Then, the operator sets the price p using `SetReward`$(p, \cdots)$ and announces it to tenants. Note that, to guide the outcome towards equilibrium, the operator can tell tenants its anticipated price range $[p_{\min}, p_{\max}]$, such that tenants can set bids to avoid a profit loss by only considering this restricted price range instead of all possible prices.

• **Reducing power demand.** After the execution of the market mechanism, each participating tenant i cuts its power by $s_i(b_i, p)$. It is at each tenant's discretion to decide the actual power reduction techniques, using a combination of resource management APIs illustrated in Figure 5.4 and/or its existing built-in power capping solutions [33, 34, 36]. Note that each knob has a different settling time for power reduction (e.g., DVFS is faster than load migration) and, depending on how much the power demand exceeds the capacity, the operator can also specify a timing constraint to guide tenants' selection of power reduction techniques.

• **Resuming normal operation.** When the tenants' aggregate power demand *without* power capping becomes lower than the capacity for a duration exceeding threshold T_c, the operator signals tenants to resume their normal operation using `Resume`$(\cdots)$. Tenants are compensated based on the power capping duration and price p.

5.4.4 Applicability of COOP

COOP applies to tenants who are interested in exchanging a temporary performance loss (due to power reduction) for financial compensation. It does not target tenants that have no tolerance on temporary performance loss (e.g., those running highly

mission-critical workloads). These tenants will be served as premium clients on separated infrastructure *without* oversubscription.

In practice, a large portion of the operator's revenue (over 50%) comes from tenants running non-mission-critical workloads (e.g., R&D, lab computing, internal services, and recently, Bitcoin) that exhibit a great scheduling flexibility for temporarily reducing power [190]. Tenants also typically provision their servers based on the peak need, thus often having a slackness for reducing server power [191, 192]. Further, increasingly mature power capping techniques [33] and emerging techniques (e.g., approximate computing [193] that trades service quality for resource/power saving), have been constantly lowering the barrier for using COOP.

As shown in Table 5.1, the operator can offer more than $20/hour for each kW reduction, which is nearly 200 times the market electricity price. If all tenants choose to neglect the operator's rewards and an unplanned downtime occurred, tenants would experience a costly business interruption but receive much less reward (around $3 per kW for each hour of downtime [178, 194]). Thus, it is also in the tenants' own interest to reduce power for handling emergencies.

In practice, tenants have no knowledge of whom they are sharing the PDU with. Further, if some tenants' power exceeds their own capacities, they will be penalized and may face an involuntary power cut. Thus, in practice, it is very difficult and risky for (some) tenants to collude and create an artificial power emergency for rewards.

Finally, whenever tenants' aggregate power demand is not capped below the capacity by using COOP for any reasons (e.g., communication failure, or insufficient financial compensation for incentiving enough power reduction), the operator may resort to other complementary power capping techniques, e.g., discharging diesel generation [166] that avoids power capacity overloading (albeit not applicable for handling cooling capacity overloading) [180]. In any event, using COOP will *not* increase the risk of outages compared to the case in which COOP is not used.

Combining all these factors, we have a good reason to believe that COOP is appealing for reducing risks of outages when power emergencies arise in a multi-tenant data center.

5.4.5 Comparison with Other Market Designs

Conceptually, our formulation in (5.41) can be viewed as a multi-resource allocation problem where the resources are "power reduction D_i for $i = 0, 1, \cdots, M$" [195, 196].

It is challenging because: first, "resources" in our context are interdependent (e.g., high-level power capacity overlaps with low-level capacity), whereas the resources to allocate are mostly orthogonal in prior research (e.g., CPU and memory in clusters [195]); and second, tenants have private cost information $c_i(s_i)$ and manage their own servers without being controlled by the operator.

While there are market-based studies (e.g., Nash bargaining) for multi-resource allocation [195, 196], their focus is on encouraging resource sharing (for improving utilization) and balancing efficiency versus fairness, whereas we aim at minimizing tenants' performance cost using a different mechanism — supply function bidding.

Market-based power management in (multi-tenant) data centers has recently received attention but differs from our work in problem formulation (due to our multiple *interdependent* power capping constraints) [43, 196–202]. Further, most of the prior studies have considered pricing-based or Vickrey-Clarke-Groves (VCG)-based mechanisms, which are not suitable for our problem due to the following limitations.

Pricing-based mechanisms. Under a pricing-based mechanism, the operator offers a reward (also called"price") to incentivize tenants' power reduction [198, 199, 201]. The challenge of such designs is the determination of the price. In order to properly set prices such that tenants reduce a desired amount of power, the operator needs to know *a priori* how much power tenants would reduce in response to the offered price, and prior literature [198] has shown that inaccurate prediction can lead to undesired outcomes (e.g., power capping violation in our context). Further, power emergencies often occur unexpectedly and thus, estimating tenants' responses is inherently highly noisy during such periods.

VCG-based mechanisms. Another commonly-studied approach to solving (5.41) is the VCG auction mechanism [203, 204], i.e., the data center operator treats the power reduction quota as a *resource* and auctions it to tenants. Such designs require that tenants submit complex bids disclosing their full cost functions $c_i(\cdot)$, which are private information. Further, under such designs the payments made to tenants may be unbounded and reward rates for different tenants' power reductions are significantly different (creating unfairness issues). Thus, VCG auction mechanisms are rarely used in real large-scale systems (see [171] for a longer discussion).

Supply function mechanisms. In contrast, COOP adapts a variant of supply function bidding widely used in power markets that, besides its cost efficiency [171], has compelling advantages. First, through a supply function, the operator *proactively*

Tenant	Type	No. of Servers	Tenant's Max. Power	Location	Cluster's Max. Power
#1	Web search	2	200 W	Cluster#A	740 W
#2	KVS	2	310 W		
#3	Hadoop	2	230 W		
#4	Web search	3	300 W	Cluster#B	530 W
#5	Hadoop	2	230 W		

Table 5.3: Testbed configuration.

solicits information from tenants as to how much power they would like to reduce if offered a certain price, while such information needs to be predicted by pricing-based mechanisms [198, 199]. Second, it uses the parameterized supply function as a proxy, thus avoiding tenants' disclosure of their private cost functions. Finally, it allows easy communication of the supply function through a single bidding parameter b_i from each tenant.

5.5 Evaluation Methodology

We now describe our methodology for evaluating the efficiency of COOP in realistic scenarios. We first describe our prototype for a multi-tenant data center, and then formalize tenants' cost and performance models.

Following prior power capping research [33, 34], we build a scale-down testbed with two clusters (labelled as #A and #B, with six and five Dell PowerEdge R720 servers, respectively) in view of the practical difficulty in accessing commercial systems. The servers each have one 6-core Intel Xeon E-2620 Processor and 32GB memory. They are virtualized to create multiple nodes. All servers are powered through CloudPOWER meters to measure power at runtime. Our testbed configuration is presented in Table 5.3, which has five tenants on the two clusters: two tenants (#1 and #4) process web search workloads, another two (#3 and #5) process Hadoop jobs and the remaining tenant (#2) processes key-value store (KVS) workloads.

According to tenants' maximum power, the total subscribed power at Cluster#A is 740W and at Cluster#B is 530W, which we use as a baseline to determine the cluster-level power oversubscription. For example, if the capacity of Cluster#A is 672W, then 740W power subscription represents a 10% oversubscription. As illustrated in Fig. 5.4, we implement the APIs for the operator on a separate desktop server, and APIs for tenants as a separate process on their own servers.

5.5.1 Workloads

In the following, we describe our implementation of the web search, key-value store (KVS) and Hadoop workloads. While COOP is not restricted to these workloads, we choose our setting for two reasons: (1) it resembles the common setting in commercial data centers serving a diverse set of tenants, including CDN, web services and data analytics; and (2) our choice of workload is consistent with prior studies (e.g., [34]) that investigate power capping for owner-operated clusters (which can be viewed as "tenants" in our context).

Web search: We use web search benchmark from CloudSuite [205]. It benchmarks the indexing process using the Nutch search engine. We implement it for tenants #1 and #4. Tenant #1 has one Nutch front end and five index serving nodes, while tenant #4 has one Nutch front end and eight index serving nodes.

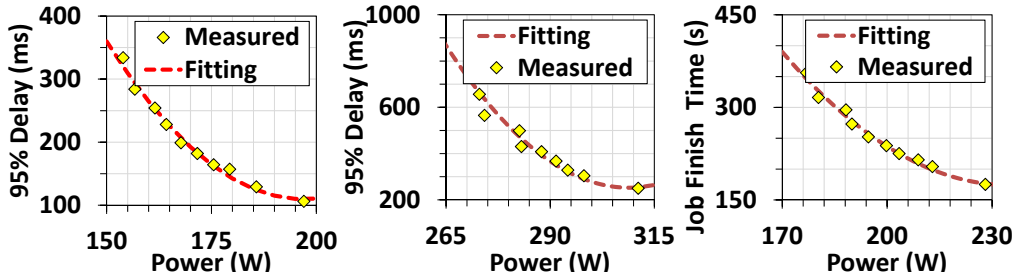
Key-value store (KVS): KVS resembles multi-tiered applications such as social networking. Tenant#2 has one load balancer VM, three Memcached VMs, three database VMs and nine application VMs.

Hadoop: Our Hadoop implementations for tenants #3 and #5 each consists of one master node and eleven worker nodes, using VMs hosted on two physical servers. We perform the *sort* benchmark on randomly generated files.

5.5.2 Performance and Cost Models

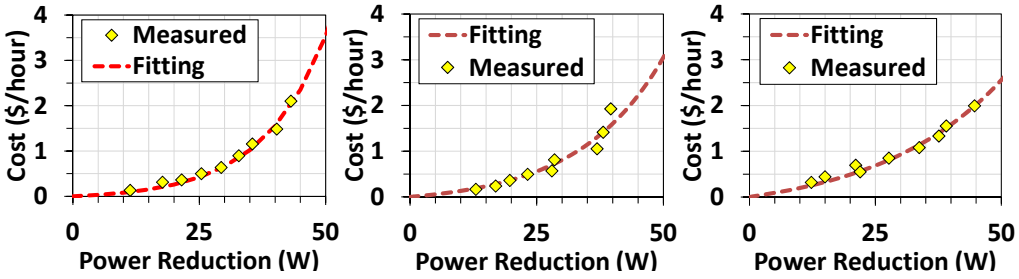
To participate in COOP, tenants need to employ power management (widely existing in today's systems [34, 191]) and evaluate their *costs* due to power reduction to decide bids.

Power and performance. Power reduction is normally accompanied by a performance degradation [33]. For the web search and KVS tenants, we use 95% response time as the performance metric (which is a key performance indicator for web services), while job completion time is used as the performance metric for the Hadoop tenants (due to the delay-tolerant nature). In our study, we consider that the tenants reduce their power using dynamic voltage frequency scaling (DVFS) supported by most modern CPUs [206]. As a commonly-used knob for power capping [33, 34], DVFS enables almost instantaneous power reduction. The Intel Xeon CPUs in our testbed servers have 10 discrete DVFS levels with processing speeds ranging from 1.2GHz to 2.0Ghz.



(a) T#1- Web search (b) T#2- KVS (c) T#3- Hadoop

Figure 5.5: Power and performance models.



(a) T#1- Web search (b) T#2- KVS (c) T#3- Hadoop

Figure 5.6: Cost models.

We model the tenants' performance and power at different DVFS levels, and show the results for the three different types of workloads in Fig. 5.5. For the convenience of clarity, we set the same speed for all servers of a tenant, and only show results under a certain traffic load: tenant #1's delay performance is measured for 80 simultaneous search sessions, tenant #2's performance is measured for 30 requests/second. These are their maximum processing capacities under their subscribed power. For tenant #3 serving Hadoop, the file size is 3GB. Fig. 5.5 shows the non-linear relation between delay performance and power consumption, indicating a natural result that tenants suffer from a greater performance loss when they run their servers in lower power modes. We do not show tenants #4 and #5, which have similar configurations to tenants #1 and #3, respectively.

Cost model. In principle, tenants have full discretion to decide their own cost models, considering one or more factors such as performance loss and risk attitude, among others. COOP applies to a large family of cost models in practice, although the theoretical efficiency guarantee only holds under a simplified setting with convex costs [171].

For evaluation purpose, we consider a cost model in terms of delay performance

and model the performance cost for web search and KVS tenants using a piece-wise cost function adopted by [207] as follows:

$$c_{tenant} = \begin{cases} a \cdot d, & \text{if } d \leq d_{th}, \\ a \cdot d + b \cdot (d - d_{th})^2, & \text{if } d > d_{th}, \end{cases} \tag{5.42}$$

where c_{tenant} is cost per job, a and b are tenants' own modeling parameters, d is 95% delay of interest, and d_{th} is the delay threshold below which the performance cost only increases linearly (since end users can barely perceive the delay increase if it is already small). When the delay exceeds the threshold, however, performance cost will increase quadratically to account for degradation in user experiences.

For the Hadoop tenants, we use a linear cost model that increases with job completion time $c_{tenant} = \rho \cdot T_{job}$, where ρ is a modeling parameter and T_{job} is the job completion time of the Hadoop system.

Using the above cost models, we determine tenants' costs corresponding to different levels of power reduction (by setting $d_{th} = 100ms$ for web search and $d_{th} = 300ms$ for KVS). Fig. 5.6 shows the cost of power reduction to the tenants, under the same traffic setting as in Fig. 5.5. We subtract the tenants' original costs (without power reduction) from their cost models to ensure "zero cost" for zero power reduction. Setting cost model parameters is the task of individual tenants.

For *evaluation* purpose, we set the cost parameter such that the tenants' cost for power reduction is comparable to the extra revenue the data center operator gets from oversubscribing the capacity. Cost function is tenant's private information, and COOP uses supply function as a proxy to avoid the disclosure of tenant's cost information.

While the cost values can be arbitrarily set by tenants, our choice in this evaluation is logical: if there are mission-critical tenants which have a very high cost of power reduction, the operator will offer these tenants a premium service and not oversubscribe the capacity serving them.

Importantly, our results are not particularly sensitive to the details of the cost model described above, provided that costs are not arbitrarily high (otherwise, those tenants are considered as "premium" and served without oversubscription). We highlight this in Section 5.6 by varying the cost models.

5.5.3 Capacity Overloading

We apply COOP to handle a two-level power emergency involving five tenants in two low-level clusters sharing one high-level UPS, for the following levels of oversubscriptions.

• *Aggressive.* Cluster#A capacity is 643W and Cluster#B capacity is 460W (15% oversubscription), while the high-level capacity is 1050W (5% oversubscription, i.e., $1050 * 1.05 = 643 + 460$).

• *Moderate.* Cluster#A capacity is 672W and Cluster#B capacity is 481W (10% oversubscription), while the high-level capacity is 1098W (5% oversubscription, i.e., $1098 * 1.05 = 672 + 481$).

• *Conservative.* Cluster#A capacity is 704W and Cluster#B capacity is 504W (5% oversubscription), while the high-level capacity is 1150W (5% oversubscription, i.e., $1150 * 1.05 = 672 + 481$).

Note that the three oversubscription cases described above are equivalent to a combined oversubscription at the high level of approximately 20%, 15% and 10%, respectively. We consider this combined 20% oversubscription as an "aggressive" strategy for two reasons. First, real-world data center power measurement demonstrates that the average power demand is roughly 70-80% of the peak [169, 208]: if the operator oversubscribes the capacity by more than 20% (equivalently, provisioning a capacity less than 83% of the peak demand), then the provisioned capacity may be quite close to or even below the servers' average power demand. Second, as shown in Table 5.1, if oversubscription is too large and exceeds 20%, the probability of overloading also increases and hence the reward rate that can be offered to tenants without decreasing the operator's profit actually decreases.

Power emergency. We create a power emergency by increasing tenants' traffic load simultaneously. The top envelope in Figure 5.8a illustrates the capacity overloading event: at around the 130th second, there is a spike in aggregate power demand, which begins to decrease by itself at around the 300th second when we decrease tenants' traffic (due to the completion of Hadoop jobs).

5.6 Evaluation Results

In this section, we evaluate COOP on the testbed described above. By assessing the efficiency of COOP in terms of total performance cost, we show that COOP is very

close to OPT. Moreover, we demonstrate that COOP provides economic benefits to both the data center operator (through extra profit) and tenants (by reducing leasing costs).

5.6.1 Baseline and Metric

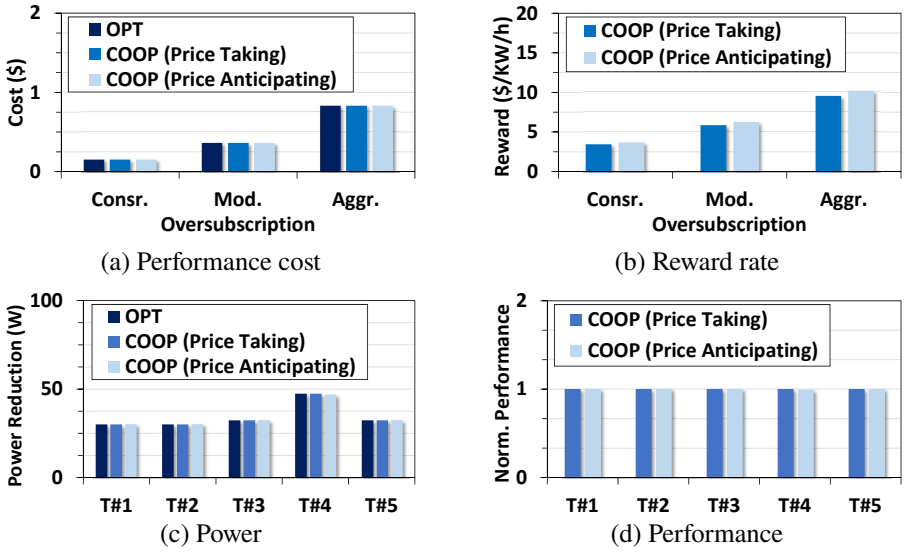
Baseline. We use OPT as the baseline, an ideal case where the operator minimizes the performance cost formulated in (5.41) and then dictates tenants' power reduction accordingly as if in an owner-operated data center.

Except for COOP, we are not aware of any alternative market mechanisms applied to handle a multi-level power capping in a multi-tenant data center. Furthermore, as shown later, COOP is very close to OPT in terms of the total performance cost (our key efficiency metric detailed below). Thus, we do not compare COOP with other market mechanisms which have yet to be introduced to multi-tenant data centers.

Metric. The key metric to assess COOP is *the total performance cost* of the tenants, which, as formulated in (5.41) and quantified in monetary value, is a scalar measure of overall performance impact on tenants. We also evaluate the tenants' performance: 95-percentile delay for web search (tenant #1 and #4) and KVS (tenant #2), and throughput (job processing rate) for Hadoop tenants (#3 and #5).

Normalized performance. Tenants' power reduction results in performance degradation during an emergency [33, 34]. Thus, we normalize tenants' performance under COOP with respect to that under OPT (our idealized baseline) to show how gracefully COOP can handle an emergency compared to OPT. Thus, the normalized performances are defined as: the ratio of OPT's 95% delay to COOP's 95% delay, and the ratio of COOP's throughput to OPT's throughput.

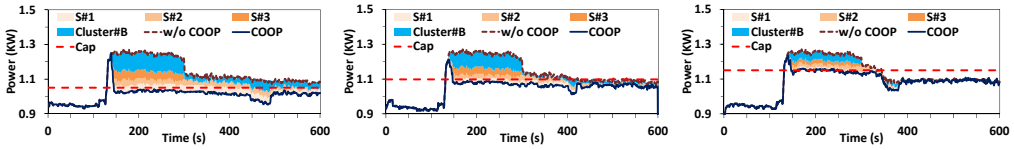
Tenants can be price-taking or price-anticipating. Price-taking means that tenants simply bid in a myopic way without predicting the impact of their bidding decisions on the market price. Price-anticipating means that tenants can predict how the operator sets price and more intelligently decide their bids to maximize their profits "$r \cdot s_i - c_i(s_i)$". See [171] for a detailed discussion of their different impacts on the equilibrium. For completeness, we show results for both cases under their respective equilibrium points, at which tenants maximize their own profits and have no incentives to deviate.



(a) Performance cost (b) Reward rate

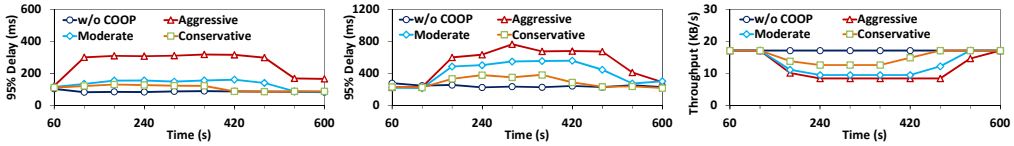
(c) Power (d) Performance

Figure 5.7: Comparison of different algorithms.



(a) Agressive oversubscription (b) Moderate oversubscription (c) Conservative oversubscription

Figure 5.8: Power traces under different oversubscription configurations.



(a) Tenant #1 (Web search) (b) Tenant #2 (Key-value store) (c) Tenant #3 (Hadoop)

Figure 5.9: Delay performance traces of the tenants under different oversubscription levels.

5.6.2 Efficiency

We first assess the efficiency of COOP in terms of the total performance cost. The results are shown in Figure 5.7a, where the absolute values are small due to the scale of our testbed. Under all the considered oversubscription levels, COOP is close to OPT, both when tenants are price-taking and when they are price-anticipating, which demonstrates that COOP is efficient in minimizing performance cost in practical settings that extend the theoretical study [171].

Figure 5.7b shows the price/reward ($/kW/Hour) paid to tenants. There is no price in OPT, because it assumes the operator's full control over tenants' servers as in an owner-operated data center. As expected, when tenants are more "clever", i.e., price-anticipating, they explicitly predict the way to set market price and then bid accordingly, thereby driving up the price.

Next, with a moderate oversubscription, we show in Figure 5.7c the breakdown of tenants' power reduction. Under both COOP and OPT, tenants' power reductions are almost identical, further confirming that COOP is close to OPT. Figure 5.7d shows COOP's performance normalized with respect to OPT's performance: COOP is almost identical to OPT in terms of the performance impact on tenants.

Settling time. There is a time lag, i.e., settling time, between the detection of power emergency and tenants' actual power reduction. First, the supply function bidding mechanism in COOP needs to be executed as described in Section 5.4.2. Each tenant needs to calculate its bidding parameter b_i based on its current traffic, which takes less than 50ms per computation and is performed in parallel; the operator clears the market price according to $r = \min_{r'}\{r' \in [r_{\min}, r_{\max}] \mid \sum_{i \in \mathcal{N}_j} s_i(b_i, r') \geq D_j, \text{ for } j = 0, 1, \cdots, M\}$, taking very little time. The messaging delay between the operator and tenants is in the order of tens of milliseconds, as only the supply function and bid/price parameters need to be communicated and the number of involved tenants is typically small (a few tens). Thus, the total time for executing COOP is less than 0.5 second.

The next step is for tenants to reduce power as decided by COOP. Here, we use DVFS as it is a widely-adopted technique and can switch between different speeds very quickly to cut enough power (even for 20% oversubscription).

The overall settling time for COOP in our study is less than one second, which is quickly enough to handle a power emergency and consistent with recent power capping studies for owner-operated data centers [34].

5.6.3 Execution

Figure 5.7 shows that the total performance cost and tenants' power reduction are very similar, under both COOP and OPT. Thus, we only show the results of COOP (with price-anticipating tenants) below.

5.6.3.1 Power demand

Aggressive oversubscription. Figure 5.8a shows the power trace for aggressive oversubscription. The top envelope represents the tenants' aggregate power demand without any power reduction, while the bottom envelope is the reduced power demand when applying COOP. The shaded areas represent the individual contributions in power reduction. We combine the contribution from tenants #4 and #5 connected to Cluster#B as a whole for better clarity. We set the timer for initiating power capping as T_w = 15s. After power capping is applied at around time 145s, the aggregate power demand goes below (but close to) the provisioned capacity. Then, at around time 490s, there is a change in the aggregate power demand (lower envelope), because tenant #5 finishes its job and COOP is re-applied to decide power reductions for participating tenants.

Moderate and conservative oversubscription. Figure 5.8b and Figure 5.8c show the power traces under moderate and conservative oversubscription, respectively. We make similar observations as in Figure 5.8a, except that tenants can resume normal operation sooner (since Hadoop tenants finish jobs sooner with a less aggressive oversubscription).

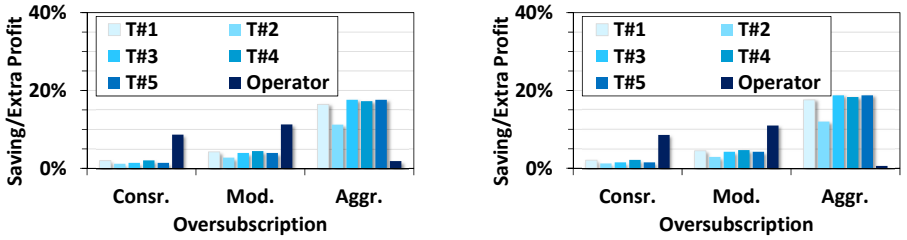
5.6.3.2 Performance

We show Cluster#A tenants' performance measured over a 60-second window in Figure 5.9. Tenants in Cluster#B have similar results. Figure 5.9a and 5.9b show the 95% delay performance of tenant #1 and tenant #2, respectively. Figure 5.9c shows the Hadoop tenant's performance (measured in the throughput, which is the inverse of job completion time given a fixed file size). As expected, we see the worst performance when the capacity is most aggressively oversubscribed (15% at the low level and 5% at the high level in our study).

While performance degradation is often unavoidable to handle power emergencies [33, 34], by using COOP, tenants' performance loss is minimum, as compared to OPT in terms of total performance cost and shown in Figure 5.7a.

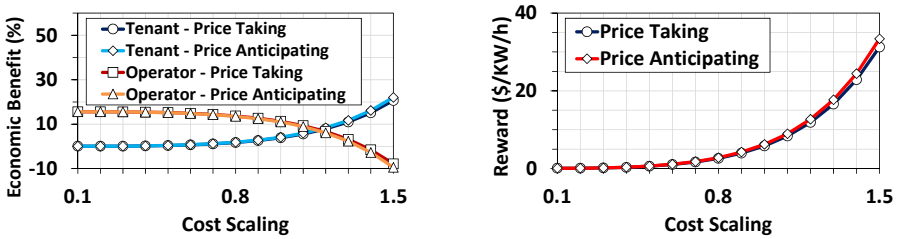
5.6.4 Economic Benefit

Figure 5.10 shows economic benefits under different oversubscription levels: tenants save leasing cost through financial compensation for temporary power reduction, while the operator earns extra profit through oversubscription. Tenants' total reward



(a) Price taking (b) Price anticipating

Figure 5.10: Economic benefit.



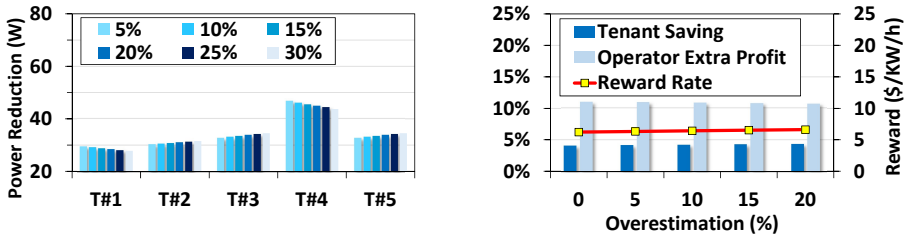
(a) Economic Benefit (b) Reward rate

Figure 5.11: Impact of tenants' cost.

is determined based on the reward rate and the probability of capacity overloading over a year (based on Figure 5.2). Tenants' cost saving is calculated as the ratio of their total rewards to their total leasing costs based on the average market price of 150$/kW/month. We exclude tenants' performance *cost*, which is a quantitative measure of tenants' performance consideration, and this is also the standard practice when assessing the cost saving benefit [34, 166]. The data center operator's extra profit is determined by subtracting the total payment to tenants from its additional revenue due to oversubscription. Figure 5.10a and Figure 5.10b show the economic benefits when tenants are price-taking and price-anticipating, respectively. In both cases, we see that tenants' cost saving goes up, as the level of oversubscription is increased. However, the operator has the highest extra profit under moderate oversubscription, because with aggressive oversubscription (20% combined oversubscription), the operator needs to pay a high price due to tenants' increasing reluctance to cut more power (Figure 5.6).

5.6.5 Tenant Costs

Tenant cost functions play a vital role in bidding decisions and hence the outcome of COOP. To illustrate the sensitivity of COOP to tenant cost functions we consider settings with costs scaled by a factor ranging from 0.1 to 1.5, and show the result



(a) Power Reduction

(b) Price Taking

Figure 5.12: Impact of tenant cost overestimation.

under moderate oversubscription in Figure 5.11. We see that regardless of price-taking and price-anticipating behaviors, tenants' saving, averaged over the three tenants, increases with their scaling of performance cost, while the operator's extra profit goes down and even becomes negative when the scaling factor is more than 1.3. Figure 5.11b shows the corresponding reward rates, which are going up as tenants' cost increases. This confirms that to earn extra profit through oversubscription, the operator should target those tenants that do *not* run highly mission-critical workloads and have a low cost for power reduction. We have also evaluated other cost models, and similar results hold.

5.6.6 Bidder Uncertainty

A main task for tenants in COOP is determination of the bidding strategy. One might expect that tenants have some uncertainty in this regard, and that this uncertainty, combined with risk aversion, may lead tenants to overestimate their costs when submitting bids. To illustrate the impact of this, we consider a setting where the web search tenants (#1 and #4) overestimate their costs by up to 30%. We see from Figure 5.12a that power reduction decreases for the two web-search tenants, while the other tenants' power reduction increases to meet power capping constraints. However, the impact is not significant. As shown in Figure 5.12b, cost overestimation slightly drives up the reward rate and has a very little impact on savings (for both operator and tenants). This is because the impact of tenants with overestimated costs is mitigated by the other tenants. Similar results hold for price-anticipating tenants. If tenants bid arbitrarily high for any reason, they will be excluded from COOP (equivalent to *premium* tenants served without oversubscription) and lose cost saving benefits provided by COOP. In fact, it is in tenants' interests to bid reasonably (as discussed in Section 5.4.2) to reach an equilibrium, at which all participating tenants maximize their own net profits.

We also run a larger-scale simulation to evaluate COOP with more tenants. Our simulation shows that COOP still applies and mutually benefits the data center operator and participating tenants. These results are omitted for brevity.

5.7 Related Work

There is a large and rich literature on power capping in owner-operated data centers. Various techniques have been proposed for minimizing performance loss, such as reducing CPU power [33, 209], admission control [210], virtualizing power allocation [34, 36], and load migration [34, 210]. These can be leveraged as power capping techniques by *individual tenants*, but they are not applicable for handling emergencies resulting from operator's oversubscription due to lack of control over tenants' servers. Recent studies [35, 166, 211] have explored discharging diesel generation (e.g., battery) to temporarily boost power supply for handling a emergency. These techniques can be viewed as "*supply*-side" solutions and are complementary to our "*demand*-side" power reduction. Further, discharging diesel generation might still overload the cooling capacity, which, typically sized based on the IT power, may increase overheating risk, which is a major reason for downtimes [168, 181]. Recent work [212] proposes to place phase changing materials inside servers to avoid cooling capacity overloading, but tenants' servers may not have such advanced materials. COOP still works if cooling capacity is over-provisioned and/or phase changing materials are available, and in such cases, these techniques can be combined with COOP to enable more power oversubscription.

Our research is relevant to multi-resource allocation [195, 196] and data center demand response (broadly interpreted as reshaping the power demand towards a desired goal) [43, 197–202, 213]. In addition to problem differences, our formulation and proposed mechanism are also different to those prior studies. Specifically, the prior studies on data center demand response [43, 198–202, 213] have all been focused on cutting power on a best-effort basis at the data center level, whereas we propose supply function bidding to address *multi-level* power capping. A detailed comparison is provided in Section 5.4.5.

5.8 Concluding Remarks

This chapter proposes COOP, a market-based approach for incentivizing and coordinating tenants' power reductions in the event of a power emergency in a multi-tenant

data center. COOP uses a supply function bidding mechanism motivated by literature in electricity markets. We demonstrate the effectiveness of COOP by building a prototype and illustrating that COOP is efficient in minimizing the total performance cost, even compared to the ideal case OPT. We also demonstrate that COOP is "win-win", increasing the data center operator's profit and reducing tenants' cost by providing financial compensation for power reductions.

BIBLIOGRAPHY

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

[2] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, I. Stoica. "Spark: Cluster Computing with Working Sets". In: *USENIX HotCloud*. 2010.

[3] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. "Apache flink: Stream and batch processing in a single engine". In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36.4 (2015).

[4] *Hadoop*. http://hadoop.apache.org.

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

[6] S. T. L. S. Ghemawat H. Gobioff. "The Google File System". In: *ACM SOSP*. 2003.

[7] *Applications and Organizations using Hadoop*. http://wiki.apache.org/hadoop/PoweredBy.

[8] *Apache Hadoop*. https://en.wikipedia.org/wiki/Apache_Hadoop.

[9] *The Next Generation of Apache Hadoop MapReduce*. http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen/.

[10] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. "Building a high-level dataflow system on top of Map-Reduce: the Pig experience". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1414–1425.

[11] *Hive*. http://wiki.apache.org/hadoop/Hive.

[12] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. "Storm@ twitter". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 147–156.

[13] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino. "Apache tez: A unifying framework for modeling and building data processing applications". In: *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*. ACM. 2015, pp. 1357–1369.

[14] J. Kreps, N. Narkhede, J. Rao, et al. "Kafka: A distributed messaging system for log processing". In: *Proceedings of the NetDB*. 2011, pp. 1–7.

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

[16] J. Dean and L. Barroso. “The Tail at Scale”. In: *Communications of the ACM* 2 (2013).

[17] 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.

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

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

[20] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. “Sparrow: Distributed, Low Latency Scheduling”. In: *ACM SOSP*. 2013.

[21] S.Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. “BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data”. In: *EuroSys*. ACM. 2013.

[22] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears. “MapReduce Online”. In: *USENIX NSDI*. 2010.

[23] *Interactive Big Data analysis using approximate answers*. http://tinyurl.com/k5favda. 2013.

[24] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. “SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets”. In: *Proceedings of the VLDB Endowment* 2 (2008).

[25] M. Zaharia, D. Borthakur, J. S. 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.

[26] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. “Low Latency Geo-distributed Data Analytics”. In: *SIGCOMM*. 2015.

[27] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese. “WANalytics: Analytics for a Geo-distributed Data-intensive World”. In: *CIDR*. 2015.

[28] A. Vulimiri, C. Curino, B. Godfrey, J. Padhye, and G. Varghese. “Global Analytics in the Face of Bandwidth and Regulatory Constraints”. In: *NSDI*. 2015.

[29] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. “Query-based Data Pricing”. In: *Proceedings of the 31st symposium on Principles of Database Systems*. 2012.

[30] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. “Toward Practical Query Pricing with QueryMarket”. In: *SIGMOD*. 2013.

[31] L. Fleischer and Y. Lyu. “Approximately Optimal Auctions for Selling Privacy when Costs are Correlated with Data”. In: *Proceedings of the 13th ACM Conference on Electronic Commerce*. 2012.

[32] C. Li, D. Li, G. Miklau, and D. Suciu. “A Theory of Pricing Private Data”. In: *ACM Transactions on Database Systems* (2014).

[33] X. Fu, X. Wang, and C. Lefurgy. “How Much Power Oversubscription is Safe and Allowed in Data Centers”. In: *ICAC*. 2011.

[34] D. Wang, C. Ren, and A. Sivasubramaniam. “Virtualizing Power Distribution in Datacenters”. In: *ISCA*. 2013.

[35] D. Wang, S. Govindan, A. Sivasubramaniam, A. Kansal, J. Liu, and B. Khessib. “Underprovisioning Backup Power Infrastructure for Datacenters”. In: *ASPLOS*. 2014.

[36] H. Lim, A. Kansal, and J. Liu. “Power budgeting for virtualized data centers”. In: *USENIX ATC*. 2011.

[37] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner. “United states data center energy usage report”. In: (2016). URL: https://eta.lbl.gov/publications/united-states-data-center-energy.

[38] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad. “Opportunities and Challenges for Data Center Demand Response”. In: *IGCC*. 2014.

[39] VMware. *Distributed Power Management Concepts and Use*. URL: http://www.vmware.com/files/pdf/Distributed-Power-Management-vSphere.pdf.

[40] Harbor Ridge Capital. *Colocation Data Centers: Overview, Trends & M&A*. URL: http://www.harborridgecap.com.

[41] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu. “Hopper: Decentralized speculation-aware cluster scheduling at scale”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 379–392.

[42] X. Ren, P. London, J. Ziani, and A. Wierman. “Datum: Managing Data Purchasing and Data Placement in a Geo-Distributed Data Market”. In: *IEEE/ACM Transactions on Networking* (2018).

[43] N. Chen, X. Ren, S. Ren, and A. Wierman. “Greening Multi-Tenant Data Center Demand Response”. In: *IFIP Performance*. 2015.

[44] M. A. Islam, X. Ren, S. Ren, A. Wierman, and X. Wang. “A market approach for handling power emergencies in multi-tenant data center”. In: *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 432–443.

[45] J. Liu, K. Shih, W. Lin, R. Bettati, and J. Chung. “Imprecise Computations”. In: *Proceedings of the IEEE* (1994).

[46] S. Lohr. *Sampling: design and analysis*. Thomson, 2009.

[47] J. Hellerstin, P. Haas, and H. Wang. “Online Aggregation”. In: *ACM SIGMOD*. 1997.

[48] M. Garofalais and P. Gibbons. “Approximate Query Processing: Taming the Terabytes”. In: *VLDB*. 2001.

[49] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012.

[50] L. Kleinrock. *Queueing systems, volume II: computer applications*. John Wiley & Sons New York, 1976.

[51] C. Liu and J. Layland. “Scheduling Algorithms for Multiprogramming in a Hard-real-time Environment”. In: *Journal of the ACM (JACM)* (1973).

[52] 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.

[53] E. Bortnikov, A. Frank, E. Hillel, S. Rao. “Predicting Execution Bottlenecks in Map-Reduce Clusters”. In: *USENIX HotCloud*. 2012.

[54] 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.

[55] 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.

[56] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. “A Study of Skew in MapReduce Applications”. In: *Open Cirrus Summit*. 2011.

[57] J. Dean. “Achieving Rapid Response Times in Large Online Services”. In: *Berkeley AMPLab Cloud Seminar*. 2012.

[58] S. Resnick. *Heavy-tail phenomena: probabilistic and statistical modeling*. Springer, 2007.

[59] J. C. Gittins. “Bandit Processes and Dynamic Allocation Indices”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1979).

[60] I. Sonin. “A Generalized Gittins Index for a Markov Chain and Its Recursive Calculation”. In: *Statistics & Probability Letters* (2008).

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

[62] W. Baek and T. Chilimbi. “Green: a Framework for Supporting Energy-conscious Programming Using Controlled Approximation”. In: *ACM Sigplan Notices*. 2010.

[63] M. Tokic and G. Palm. “Value-difference Based Exploration: Adaptive Control between Epsilon-greedy and Softmax”. In: *KI 2011: Advances in Artificial Intelligence*. Springer, 2011.

[64] A. Baratloo, M. Karaul, Z. Kedem, and P. Wycko. “Charlotte: Metacomputing on the Web”. In: *9th Conference on Parallel and Distributed Computing Systems*. 1996.

[65] E. K. D. Anderson J. Cobb. “SETI@home: An Experiment in Public-Resource Computing”. In: *Comm. ACM*. 2002.

[66] M. Rinard and P. Diniz. “Commutativity Analysis: a New Analysis Framework for Parallelizing Compilers”. In: *ACM PLDI*. 1996.

[67] D. Paranhos, W. Cirne, and F. Brasileiro. “Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids”. In: *Euro-Par*. 2003.

[68] G. Ghare and S. Leutenegger. “Improving Speedup and Response Times by Replicating Parallel Programs on a SNOW”. In: *JSSPP*. 2004.

[69] W. Cirne, D. Paranhos, F. Brasileiro, L. Goes, and W. Voorsluys. “On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems”. In: *Parallel Computing*. 2007.

[70] *Cloudera Impala*. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.

[71] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. “Dremel: Interactive Analysis of Web-Scale Datasets”. In: *VLDB*. 2010.

[72] E. Boutin, J. Ekanayake, W. Kin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. “Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing”. In: *USENIX OSDI*. 2014.

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

[74] F. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. “Decentralized Task-aware Scheduling for Data Center Networks”. In: *ACM SIGCOMM*. 2014.

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

[76] 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.

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

[78] 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.

[79] J. Tan, X. Meng, and L. Zhang. “Delay Tails in MapReduce Scheduling”. In: *ACM SIGMETRICS Performance Evaluation Review* (2012).

[80] M. Zaharia, D. Borthakur, J. S. 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.

[81] L. Schrage. “A Proof of the Optimality of the Shortest Remaining Processing Time Discipline”. In: *Operations Research* 16.3 (1968), pp. 687–690.

[82] K. Pruhs, J. Sgall, and E. Torng. “Online scheduling”. In: *Handbook of scheduling: algorithms, models, and performance analysis* (2004), pp. 15–1.

[83] M. Lin, L. Zhang, A. Wierman, and J. Tan. “Joint Optimization of Overlapping Phases in MapReduce”. In: *Performance Evaluation* (2013).

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

[85] Y. Wang, J. Tan, W. Yu, L. Zhang, and X. Meng. “Preemptive ReduceTask Scheduling for Fast and Fair Job Completion”. In: *USENIX ICAC* (2013).

[86] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B. Chun. “Making Sense of Performance in Data Analytics Frameworks”. In: *USENIX NSDI*. 2015.

[87] E. Bortnikov, A. Frank, E. Hillel, and S. Rao. “ Predicting Execution Bottlenecks in Map-Reduce Clusters”. In: *USENIX HotCloud*. 2012.

[88] N. Yadwadkar, G. Ananthanarayanan, and R. Katz. “ Wrangler: Predictable and Faster Jobs using Fewer Resources”. In: *ACM SoCC*. 2014.

[89] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. “Size-based scheduling to improve web performance”. In: *ACM Transactions on Computer Systems (TOCS)* 21.2 (2003), pp. 207–233.

[90] A. Wierman. “Fairness and scheduling in single server queues”. In: *Surveys in Operations Research and Management Science* 16.1 (2011), pp. 39–48.

[91] A. Wierman and M. Harchol-Balter. “Classifying scheduling policies with respect to unfairness in an M/GI/1”. In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 31. 1. ACM. 2003, pp. 238–249.

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

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

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

[95] A. Richa, M. Mitzenmacher, and R. Sitaraman. "The power of two random choices: A survey of techniques and results". In: *Combinatorial Optimization* (2001).

[96] M. Bramson, Y. Lu, and B. Prabhakar. "Randomized load balancing with general service time distributions". In: *Proceedings of Sigmetrics*. 2010, pp. 275–286.

[97] *Sparrow*. https://github.com/radlab/sparrow.

[98] *Apache Thrift*. https://thrift.apache.org/.

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

[100] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. "Multi-Resource Packing for Cluster Schedulers". In: *ACM SIGCOMM*. 2014.

[101] *Study Identifies Common Pain Points in Big Data Projects*. http://www.bigdataexchange.com/tag/list-of-third-party-data-providers/. 2015.

[102] *Qlik*. http://www.qlik.com/us/products/qlik-data-market.

[103] *Factual*. https://www.factual.com/. 2015.

[104] *Infochimps*. http://www.infochimps.com/. 2015.

[105] *Xignite*. http://www.xignite.com/. 2015.

[106] *The IUPHAR/BPS Guide to Pharmacology*. http://www.guidetopharmacology.org/. 2015.

[107] *Google BigQuery Public Datasets*. https://cloud.google.com/bigquery/public-data/.

[108] *Azure Public Datasets*. https://docs.microsoft.com/en-us/azure/sql-database/sql-database-public-data-sets.

[109] *AWS Public Datasets*. https://aws.amazon.com/public-datasets/.

[110] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. Ganger, P. Gibbons, and O. Mutlu. "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds". In: *NSDI*. 2017.

[111] R. Viswanathan, G. Ananthanarayanan, and A. Akella. “Clarinet: Wan-aware Optimization for Analytics Queries”. In: *OSDI*. 2016.

[112] J. Krarup and P. Pruzan. “The Simple Plant Location Problem: Survey and Synthesis”. In: *European Journal of Operational Research* (1983).

[113] M. Charikar, S. Guha, É. Tardos, and D. Shmoys. “A Constant-factor Approximation Algorithm for the K-median Problem (Extended Abstract)”. In: *STOC*. 1999.

[114] S. Guha and S. Khuller. “Greedy Strikes Back: Improved Facility Location Algorithms”. In: *Journal of Algorithms* (1999).

[115] K. Jain and V. Vazirani. “Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-dual Schema and Lagrangian Relaxation”. In: *J. ACM* (2001).

[116] D. Hochbaum. “Heuristics for the Fixed Cost Median Problem”. In: *Math. Program.* (1982).

[117] V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[118] F. Uriel. “A Threshold of ln n for Approximating Set Cover”. In: *J. ACM* (1998).

[119] D. Erlenkotter. “A Dual-Based Procedure for Uncapacitated Facility Location”. In: *Operations Research* (1978).

[120] J. Beasley. “Lagrangean Heuristics for Location Problems”. In: *European Journal of Operational Research* (1993).

[121] K. Al-Sultan and M. Al-Fawzan. “A Tabu Search Approach to the Uncapacitated Facility Location Problem”. In: *Annals of Operations Research* (1999).

[122] M. Korkel. “On the Exact Solution of Large-scale Simple Plant Location Problems ”. In: *European Journal of Operational Research* (1989).

[123] D. Tuzun and L. Burke. “A Two-phase Tabu Search Approach to the Location Routing Problem ”. In: *European Journal of Operational Research* (1999).

[124] D. Ghosh. “Neighborhood Search Heuristics for the Uncapacitated Facility Location Problem ”. In: *European Journal of Operational Research* (2003).

[125] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. “Query-Market Demonstration: Pricing for Online Data Markets”. In: *Proceedings of the VLDB Endowment* (2012).

[126] *Visipedia Project*. http://www.vision.caltech.edu/visipedia/. 2015.

[127] J. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al. “Spanner: Google’s Globally Distributed Database”. In: *ACM Transactions on Computer Systems* (2013).

[128] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. Dhoot, A. Kumar, A. Agiwal, et al. “Mesa: Geo-replicated, Near Real-time, Scalable Data Warehousing”. In: *Proceedings of the VLDB Endowment* (2014).

[129] A. Rabkin, M. Arye, S. Sen, V. Pai, and M. Freedman. “Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area”. In: *NSDI*. 2014.

[130] M. Balazinska, B. Howe, and D. Suciu. “Data Markets in the Cloud: An Opportunity for the Database Community”. In: *Proceedings of the VLDB Endowment* (2011).

[131] R. Cummings, K. Ligett, A. Roth, Z. Wu, and J. Ziani. “Accuracy for Sale: Aggregating Data with a Variance Constraint”. In: *ITCS*. 2015.

[132] R. Tang, H. Wu, Z. Bao, S. Bressan, and P. Valduriez. “The Price Is Right: Models and Algorithms for Pricing Data”. In: *M. Castellanos, U. Dayal, E. A. Rundensteiner (eds) Database and Expert Systems Applications. DEXA 2014. Lecture Notes in Business Information Processing* 8056 (2013).

[133] R. Tang, A. Amarilli, P. Senellart, and S. Bressan. “Get a Sample for a Discount”. In: *H. Decker, L. Lhotská, S. Link, M. Spies, R. Wagner (eds) Database and Expert Systems Applications. DEXA 2014. Lecture Notes in Computer Science* 8644 (2014).

[134] A. Muschalle, F. Stahl, A. Loser, and G. Vossen. “Pricing Approaches for Data Markets”. In: *M. Castellanos, U. Dayal, E.A. Rundensteiner (eds) Enabling Real-Time Business Intelligence. BIRTE 2012. Lecture Notes in Business Information Processing* 154 (2013).

[135] R. Stahl and G. Vossen. “Data Quality Scores for Pricing on Data Marketplaces”. In: *Nguyen N.T., Trawi?ski B., Fujita H., Hong TP. (eds) Intelligent Information and Database Systems. ACIIDS 2016. Lecture Notes in Computer Science* 9621 (2016).

[136] R. Stahl and G. Vossen. “Name Your Own Price on Data Marketplaces”. In: 28 (Jan. 2017), pp. 155–180.

[137] C. Hung, L. Golubchik, and M. Yu. “Scheduling Jobs across Geo-distributed Datacenters”. In: *Proceedings of the 6th ACM Symposium on Cloud Computing*. 2015.

[138] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. “Live Video Analytics at Scale with Approximation and Delay-Tolerance”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 377–392. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang.

[139] A. Vakali and G. Pallis. “Content delivery networks: status and trends”. In: *IEEE Internet Computing* 7.6 (Nov. 2003), pp. 68–74. ISSN: 1089-7801. DOI: 10.1109/MIC.2003.1250586.

[140] G. Peng. “CDN: Content Distribution Network”. In: *CoRR* cs.NI/0411069 (2004). URL: http://arxiv.org/abs/cs.NI/0411069.

[141] G. Pallis and A. Vakali. “Insight and Perspectives for Content Delivery Networks”. In: *Commun. ACM* 49.1 (Jan. 2006), pp. 101–106. ISSN: 0001-0782. DOI: 10.1145/1107458.1107462. URL: http://doi.acm.org/10.1145/1107458.1107462.

[142] A.-M. K. Pathan and R. Buyya. “A taxonomy and survey of content delivery networks”. In: *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report* 4 (2007).

[143] J. D. Guyton and M. F. Schwartz. “Locating Nearby Copies of Replicated Internet Servers”. In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM ’95. Cambridge, Massachusetts, USA: ACM, 1995, pp. 288–298. ISBN: 0-89791-711-1. DOI: 10.1145/217382.217463. URL: http://doi.acm.org/10.1145/217382.217463.

[144] Z.-M. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. “A novel server selection technique for improving the response time of a replicated service”. In: *INFOCOM’98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. IEEE. 1998, pp. 783–791.

[145] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. “On the placement of Internet instrumentation”. In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 1. 2000, 295–304 vol.1. DOI: 10.1109/INFCOM.2000.832199.

[146] P. Krishnan, D. Raz, and Y. Shavitt. “The Cache Location Problem”. In: *IEEE/ACM Trans. Netw.* 8.5 (Oct. 2000), pp. 568–582. ISSN: 1063-6692. DOI: 10.1109/90.879344. URL: http://dx.doi.org/10.1109/90.879344.

[147] M. Gritter and D. R. Cheriton. “An Architecture for Content Routing Support in the Internet”. In: *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*. USITS’01. San Francisco, California: USENIX Association, 2001, pp. 4–4. URL: http://dl.acm.org/citation.cfm?id=1251440.1251444.

[148] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. “On the placement of web server replicas”. In: *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. IEEE. 2001, pp. 1587–1596.

[149] J. Kangasharju, J. Roberts, and K. W. Ross. “Object Replication Strategies in Content Distribution Networks”. In: *Comput. Commun.* 25.4 (Mar. 2002), pp. 376–383. ISSN: 0140-3664. DOI: 10.1016/S0140-3664(01)00409-1. URL: http://dx.doi.org/10.1016/S0140-3664(01)00409-1.

[150] C. Gkantsidis and P. R. Rodriguez. “Network coding for large scale content distribution”. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 4. Mar. 2005, 2235–2245 vol. 4. DOI: 10.1109/INFCOM.2005.1498511.

[151] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev. “FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs”. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 381–394. ISBN: 978-1-931971-218. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/flavel.

[152] F. Chen, R. K. Sitaraman, and M. Torres. “End-User Mapping: Next Generation Request Routing for Content Delivery”. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM ’15. London, United Kingdom: ACM, 2015, pp. 167–181. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787500. URL: http://doi.acm.org/10.1145/2785956.2787500.

[153] C. Dwork. “Differential Privacy”. In: *Encyclopedia of Cryptography and Security*. 2011.

[154] *Microsoft Azure*. https://azure.microsoft.com/en-us/. 2015.

[155] J. Wiener and N. Boston. *Facebook’s top open data problems.* https://research.facebook.com/blog/1522692927972019/facebook-s-top-open-data-problems/. 2014.

[156] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy. “The Unified Logging Infrastructure for Data Analytics at Twitter”. In: *Proceedings of the VLDB Endowment* (2012).

[157] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[158] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. 1997.

[159] *Google Data Center FAQ*. http://www.datacenterknowledge.com/archives/2012/05/15/google-data-center-faq/. 2012.

[160] M. Newman. “Power Laws, Pareto Distributions and Zipf’s Law”. In: *Contemporary physics* (2005).

[161] M. Balazinska, B. Howe, P. Koutris, D. Suciu, and P. Upadhyaya. “A Discussion on Pricing Relational Data”. In: *In Search of Elegance in the Theory and Practice of Computation*. 2013.

[162] J. Zhang. “Approximating the two-level facility location problem via a quasi-greedy approach”. In: *Mathematical Programming* (2006).

[163] A. Venkatraman. “Global census shows datacentre power demand grew 63% in 2012”. In: *ComputerWeekly.com*. 2012.

[164] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. “The Cost of a Cloud: Research Problems in Data Center Networks”. In: *SIGCOMM Comput. Commun. Rev.* 39.1 (Dec. 2008).

[165] L. A. Barroso, J. Clidaras, and U. Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2013.

[166] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy. “Energy storage in datacenters: what, where, and how much?” In: *SIGMETRICS*. 2012.

[167] Ponemon Institute. *2013 Cost of Data Center Outages*. 2013. URL: http://goo.gl/6mBFTV.

[168] Emerson Network Power. *Addressing the Leading Root Causes of Downtime*. 2013. URL: http://goo.gl/b14XaF.

[169] X. Fan, W.-D. Weber, and L. A. Barroso. “Power provisioning for a warehouse-sized computer”. In: *ISCA*. 2007.

[170] Uptime Institute. *Data Center Industry Survey*. 2014.

[171] R. Johari and J. N. Tsitsiklis. “Parameterized Supply Function Bidding: Equilibrium and Efficiency”. In: *Oper. Res.* 59.5 (Sept. 2011), pp. 1079–1089.

[172] Y. Xu, N. Li, and S. H. Low. “Demand Response with Capacity Constrained Supply Function Bidding”. In: *IEEE Transactions on Power Systems* (2015). DOI: 10.1109/TPWRS.2015.2421932.

[173] Datacenter Map. *Colocation USA*. URL: http://www.datacentermap.com/usa/.

[174] NRDC. “Scaling Up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers”. In: *Issue Paper* (Aug. 2014).

[175] Akamai. *Environmental Sustainability Policy*. URL: http://www.akamai.com/html/sustainability/our_commitment.html.

[176] Microsoft. *Global Infrastructure*. URL: http://www.globalfoundationservices.com/.

[177] A. C. Riekstin, S. James, A. Kansal, J. Liu, and E. Peterson. “No More Electrical Infrastructure: Towards Fuel Cell Powered Data Centers”. In: *SIGOPS Oper. Syst. Rev.* 48.1 (May 2014), pp. 39–43.

[178] CBRE. *Q4 2013: National Data Center Market Update*. 2013.

[179] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. "Managing distributed ups energy for effective power capping in data centers". In: *ISCA*. 2012.

[180] W. Zheng and X. Wang. "Data Center Sprinting: Enabling Computational Sprinting at the Data Center Level". In: *ICDCS*. 2015.

[181] I. Manousakis, Í. Goiri, S. Sankar, T. D. Nguyen, and R. Bianchini. "CoolProvision: Underprovisioning Datacenter Cooling". In: *SoCC*. 2015.

[182] Google. *Compute Engine Incident #15056*. URL: https://status.cloud.google.com/incident/compute/15056.

[183] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar. "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters". In: *ASPLOS*. 2012.

[184] EnerNOC. *Ensuring U.S. Grid Security and Reliability: U.S. EPA's Proposed Emergency Backup Generator Rule*. 2013. URL: http://www.whitehouse.gov/sites/default/files/omb/assets/oira_2060/2060_12102012-2.pdf.

[185] S. Ong, P. Denholm, and E. Doris. *Impacts of Commercial Electric Utility Rate Structure Elements on the Economics of Photovoltaic Systems*. Tech. rep. National Renewable Energy Laboratory (NREL), Golden, CO., 2010.

[186] L. L. Andrew, M. Lin, and A. Wierman. "Optimality, fairness, and robustness in speed scaling designs". In: *SIGMETRICS*. 2010.

[187] A. Wierman, L. L. H. Andrew, and A. Tang. "Power-aware speed scaling in processor sharing systems: Optimality and robustness". In: *Perform. Eval.* 69.12 (Dec. 2012), pp. 601–622.

[188] *Equinix, Customer portal*. URL: http://www.equinix.com/services/support/customer-portal/.

[189] PJM. "Emergency Demand Response (Load Management) Performance Report – 2012/2013". In: (Dec. 2012).

[190] J. dePreaux. "Wholesale and Retail Data Centers - North America and Europe - 2013". In: *IHS* (July 2013). URL: https://technology.ihs.com/api/binary/492570.

[191] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. "Towards Energy Proportionality for Large-scale Latency-critical Workloads". In: *ISCA*. 2014.

[192] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. "Dynamic right-sizing for power-proportional data centers". In: *IEEE Infocom*. 2011.

[193] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. "ApproxHadoop: Bringing Approximations to MapReduce Frameworks". In: *ASPLOS*. 2015.

[194] Internap. *Colocation services and SLA*. URL: http://www.internap.com/internap/wp-content/uploads/2014/06/Attachment-3-Colocation-Services-SLA.pdf.

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

[196] S. M. Zahedi and B. C. Lee. "REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors". In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS. 2014.

[197] M. Guevara, B. Lubin, and B. C. Lee. "Navigating Heterogeneous Processors with Market Mechanisms". In: *HPCA*. 2013.

[198] Z. Liu, I. Liu, S. Low, and A. Wierman. "Pricing Data Center Demand Response". In: *SIGMETRICS*. 2014.

[199] C. Wang, N. Nasiriani, G. Kesidis, B. Urgaonkar, Q. Wang, L. Y. Chen, A. Gupta, and R. Birke. "Recouping Energy Costs From Cloud Tenants: Tenant Demand Response Aware Pricing Design". In: *e-Energy*. 2015.

[200] L. Zhang, S. Ren, C. Wu, and Z. Li. "A Truthful Incentive Mechanism for Emergency Demand Response in Colocation Data Centers". In: *INFOCOM*. 2015.

[201] C. Wang, B. Urgaonkar, G. Kesidis, U. V. Shanbhag, and Q. Wang. "A Case for Virtualizing the Electric Utility in Cloud Data Centers". In: *HotCloud*. 2014.

[202] M. A. Islam, H. Mahmud, S. Ren, and X. Wang. "Paying to Save: Reducing Cost of Colocation Data Center via Rewards". In: *HPCA*. 2015.

[203] T. Roughgarden. "Algorithmic Game Theory". In: *Commun. ACM* 53.7 (July 2010), pp. 78–86.

[204] L. Zhang, S. Ren, C. Wu, and Z. Li. "A Truthful Incentive Mechanism for Emergency Demand Response in Colocation Data Centers". In: *INFOCOM*. 2015.

[205] *CloudSuite - The Search Benchmark*. URL: http://parsa.epfl.ch/cloudsuite/.

[206] J. R. Lorch and A. J. Smith. "PACE: A new approach to dynamic voltage scaling". In: *IEEE Trans. Computers* 53 (7 July 2004), pp. 856–869.

[207] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. "It's not easy being green". In: *SIGCOMM Comput. Commun. Rev.* (2012).

[208] D. Wang, C. Ren, S. Govindan, A. Sivasubramaniam, B. Urgaonkar, A. Kansal, and K. Vaid. "ACE: Abstracting, Characterizing and Exploiting Peaks and Valleys in Datacenter Power Consumption". In: *SIGMETRICS*. 2013.

[209] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller. “SHIP: Scalable hierarchical power control for large-scale data centers”. In: *PACT*. 2009.

[210] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar. “The Need for Speed and Stability in Data Center Power Capping”. In: *IGCC*. 2012.

[211] L. Liu, C. Li, H. Sun, Y. Hu, J. Gu, T. Li, J. Xin, and N. Zheng. “HEB: Deploying and Managing Hybrid Energy Buffers for Improving Datacenter Efficiency and Economy”. In: *ISCA*. 2015.

[212] M. Skach, M. Arora, C.-H. Hsu, Q. Li, D. Tullsen, L. Tang, and J. Mars. “Thermal Time Shifting: Leveraging Phase Change Materials to Reduce Cooling Costs in Warehouse-scale Computers”. In: *ISCA*. 2015.

[213] S. Ren and M. A. Islam. “Colocation Demand Response: Why Do I Turn Off My Servers?” In: *ICAC*. 2014.