Efficient Redundancy Techniques to Reduce Delay in Cloud Systems

by

Gauri Joshi

B. Tech. and M. Tech., Indian Institute of Technology Bombay (2010)
S. M., Massachusetts Institute of Technology (2012)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2016

Certified by. . . . .
Gregory W. Wornell
Sumitomo Professor of Engineering
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

Efficient Redundancy Techniques to Reduce Delay in Cloud Systems

by

Gauri Joshi

Abstract

Cloud services are changing the world by providing millions of people low-cost access to the computing power of data centers. Storing and processing data on shared servers in the cloud provides scalability and flexibility to these services. However the large-scale sharing of resources also causes unpredictable fluctuations in the response time of individual servers. In this thesis we use *redundancy* as a tool to combat this variability. We study three areas of cloud infrastructure: cloud computing, distributed storage, and streaming communication.

In cloud computing, replicating a task on multiple machines and waiting for the earliest copy to finish can reduce service delay. But intuitively, it costs additional computing resources, and increases queueing load on the servers. In the first part of this thesis we analyze the effect of redundancy on queues. Surprisingly, there are regimes where replication not only reduces service delay but also reduces queueing load, thus making the system more efficient. Similarly, we can speed-up content download from cloud storage systems by requesting multiple replicas of a file and waiting for any one. In the second part of the thesis we generalize from replication to coding, and propose the (n, k) fork-join model to analyze the delay in accessing an (n, k) erasure-coded storage system. This analysis provides practical insights into how many users can access a piece of content simultaneously, and how fast they can be served. Achieving low latency is even more challenging in streaming communication because the packets need to be delivered fast and in-order. The third part of this thesis develops erasure codes to transmit redundant combinations of packets and ensure smooth playback.

This thesis blends a diverse set of mathematical tools from queueing, coding theory, and renewal processes. Although we focus on cloud infrastructure, the techniques and insights are applicable to other systems with stochastically varying components.

Thesis Supervisor: Gregory W. Wornell
Title: Sumitomo Professor of Engineering

Acknowledgments

After six fantastic years here at MIT, it is hard to imagine that it is time to leave. This joyous and fulfilling ride would be incomplete without so many people who shared it with me: my teachers, mentors, colleagues, friends, and family. First of all, I would like to express my deepest gratitude to my advisor Gregory Wornell for his incredible support and mentorship. Greg taught me to be patient and keep the big picture in mind while defining and solving a problem. In every meeting, he would bring to light a new perspective that I did not consider before. He has been an ideal advisor: giving me the freedom to explore new research ideas and collaborations, while being extremely supportive whenever I needed help.

My thesis committee members Emina Soljanin and Devavrat Shah have also been an invaluable part of my grad school experience. The seeds of this thesis were planted during the summer I spent interning with Emina at Bell Labs. In the years following, she has been instrumental in helping me connect with the academic community and establish collaborations. I cannot thank her enough for her mentorship. Devavrat's wisdom and infectious enthusiasm for research make him an amazing committee member. He asked some sharp and insightful questions during our meetings that helped shape several key results presented in this thesis.

I had the pleasure to work with wonderful collaborators Yuval Kochman, Da Wang, Yanpei Liu, Ashish Khisti, and Kaveh Mahadaviani, who help me grow in research over the years. I also got the chance to be a teaching assistant for the brand new Introduction to Inference (6.008) class. Working on the assignments, exams and notes with Greg, Lizhong, Ramesh Sridharan and George Chen taught me how much effort goes into designing a class.

I would also like to thank all the faculty members who I interacted with over the years, in particular, my academic advisor Asu Ozdaglar, Lizhong Zheng, Polina Golland, Bob Gallager, Muriel Medard, Alan Oppenheim, Yury Polyanskiy and Guy Bresler. During my graduate years, I also spent two great summers at Google Mountain View where I got a chance to work first-hand with Google's unbelievable com-

puting infrastructure, under the mentorship of Arif Merchant, Alex Shraer, and Brett Schein. This experience pushed me to remove many assumptions in my research, and strive to bring theory closer to practice.

I thank all my current and past labmates for making the Signals, Information and Algorithms (SiA) Lab a vibrant, welcoming home within MIT: Qing, Da, Ying-zong, Atulya, James, Maryam, Yuval, Arya, Ligong, Uri, Or, Hongchao, Lisa, Gal, Ganesh, Joshua, Adam and other alumni and visitors. Qing He has been my office-mate for the past six years, and is one of my closest friends at MIT. Da Wang was an amazing senior who helped me navigate the lab, and MIT in general. I also enjoyed my interactions with neighboring groups in RLE and LIDS. And I thank our lab administrator Tricia O' Donnell for making this community possible. Outside the lab, I had the pleasure to be involved in student groups such as MIT Sangam, Graduate Women of Course 6, and EECS REFS where I learnt to work together in a team. Most importantly, I made incredibly fun and caring friends: Diviya, Mina, Radhika, Sayalee and many others (too many to name here), who stood by me during tough times.

My dream of coming to MIT would not have come true without the world-class academic training I received at my alma mater IIT Bombay. I thank all my teachers whose passion for learning and teaching has brought me this far. Most importantly I thank my first teachers: my parents Vibhavari and Diwakar Joshi for their strife to give me the best possible education. Finally, I want to thank my husband Shreerang Chhatre for his unconditional love and friendship, and for being my pillar of strength whenever I was overwhlemed and stressed. This thesis would be impossible without his support.

Contents

List of Figures

15

Chapter 1

Introduction

1.1 Motivation

The amount of data stored in the Internet cloud is rapidly increasing. According to a recent report [1], there is over one exabyte, that is, over a billion gigabytes worth of files currently stored on the cloud. This content includes videos, photos, documents which can be accessed by the users via services such as YouTube, NetFlix, Dropbox, Google Drive, Microsoft Azure, Amazon S3 etc. These cloud services are changing the world by allowing millions of people low-cost access to the enormous computing power of data centers.

Besides low-cost access, two major demands of users from cloud services are: high reliability, and low delay. There is a large body of research focused on the first demand. In particular, research in information theory and coding [2, 3] aims to find the fundamental limits, and design coding schemes to maximize the information reliably communicated over a noisy channel. Coding is also used distributed storage systems such as [4] to provide reliability against node failures. Recent work [5, 6] proposes new codes that allow efficient repair of failed nodes.

The second demand for low delay, which is relatively less explored in the context of cloud storage and computing, is becoming increasingly important as a measure of the quality. Ensuring fast and seamless service is critical for cloud services because delayed response turns away users, causing revenue loss. For example, Google recently

reported a 20% loss of search traffic when there was a mere 0.5 sec increase in the delay in loading search results. As pointed out in [7] applications are becoming more interactive, and need to respond to user actions quickly (within 100 ms) to feel fluid and natural.

However, guaranteeing such sub-second response times is challenging because there can be random delays in the cloud due to factors such as server outages, virtualization, congestion, and network packet loss. The key reason behind this variability is the large-scale sharing of resources in the cloud, and limited centralized control on their allocation. As noted in [7], such delays are the norm and not an exception in today's cloud systems .

1.2   Goals

In this thesis we seek to understand how *redundancy* can be used to reduce delay in the following three areas of cloud infrastructure: computing, storage and streaming communication. For example, in cloud computing systems, we can assign a task to multiple machines and wait for any one replica to finish. Similarly in cloud storage systems, latency can be reduced by sending redundant requests to multiple replicas of the content and waiting for one of the replicas to be downloaded. In streaming, retransmitting older undecoded packets can enable faster in-order decoding and playback. This reduction in delay comes at the cost of additional resources such as network bandwidth, storage space, or computing time. We characterize the trade-off between delay and resource usage and develop strategies to reduce delay with efficient use of available resources.

This thesis also brings to the forefront a combination of diverse tools from coding theory, queueing and scheduling. The importance of delay as a metric of performance calls for the use of these tools. We believe that our analysis framework and tools can also be applied to a wide variety of other latency-sensitive applications beyond the scope of cloud services, for example crowdsourcing, traffic scheduling etc.

1.3 Summary and Organization

This section summarizes the key contributions of the three parts of this thesis, and describes the organization of chapters within these parts.

1.3.1 Task Replication in Cloud Computing

In Part I we develop an understanding of how launching redundant tasks affects the cost, as well as queueing delay for other tasks. This leads to cost-efficient strategies that can make cloud computing faster, yet sustainable.

Log-concavity of task service time*: Replicating each task and waiting for the earliest copy can help combat random service delays. But it generally costs additional computing resources, and also increases waiting time in queue for subsequent tasks. Chapter 2 provides a framework to answer fundamental questions about queues with redundancy such as: 1) how many replicas to launch? 2) which queues to join? and 3) when should we cancel the redundant tasks? We discover that the *log-concavity* of the task service time X governs the choice of the number of replicas, and when we should cancel them. If X is log-convex, that is, the task service time is highly variable, then launching more replicas reduces latency (service time plus waiting time in queue) as well as the computing cost. Thus surprisingly, adding more replicas improves the overall service rate of the system of servers. On the other hand, if X is log-concave it is more cost-efficient to have fewer replicas, and cancel the redundant tasks earlier.

Replicating Straggling Tasks*: In Chapter 3 we consider a related problem of minimizing the latency of a large computing job with parallel tasks. The tasks that are run on the slowest machines, which are referred to as stragglers, become a bottleneck in the completion of the job. To alleviate the problem of stragglers, systems such as MapReduce launch replicas of the slowest tasks. We provide a theoretical framework to understand how the replication of stragglers affects the latency and the cost

of additional computing time. This analysis helps identify regimes where straggler replication can drastically reduce latency, and also reduce the computing cost. We propose replication strategies that can give a better latency-cost trade-off than the default in MapReduce, as indicated by our experiments with Google Cluster Trace data.

Future Directions*:* In Chapter 4 we discuss several generalizations of the model considered in Chapter 2 and Chapter 3, and future research directions. We also describe the preliminary insights on two future directions that are of particular interest to us: considering heterogeneous servers, and scheduling with the service time distribution is unknown. With these generalizations we aim to devise a unified scheduling policy for cloud computing frameworks.

1.3.2   Fast Content Download from Coded Storage

In Part II we use redundancy to speed-up content download from distributed cloud storage. In cloud storage, content is often replicated at multiple servers. Similar to the cloud computing scenario considered in Part I, we can send requests to multiple replicas and wait for the earliest copy. Instead of replication, erasure coding can also provide diversity, but with lower storage space. For example, with an (n, k) Reed-Solomon code, downloading any k out of n chunks of a file are sufficient to recover it. In Part II, we study the interplay between content download delay and resource cost from such coded storage systems. The key results are summarized below.

Introducing the (n, k) ***fork-join system***: In Chapter 5 we describe the background and previous work on content download from distributed storage. Then we introduce the (n, k) *fork-join system* to model delay in content download. Each download request assigned to n different servers that store coded chunks of the file. It is sufficient for any $k < n$ chunks to recover the file. The $k = 1$ case corresponds to replicated storage.

Latency-Cost Analysis: In Chapter 6 we analyze the latency (expected waiting

time in queue plus service time) of the (n, k) fork-join system. Finding the latency of the (n, n) fork-join queue is a famously hard problem, even with exponential service time. An exact expression for $n = 2$ was given by [8], while only bounds are known for general n [9, 10]. In Chapter 6 we extend these latency bounds to the (n, k) fork-join system, and arbitrary service time distributions. We also find bounds on the cost of computing time spent per job. This analysis can be used to choose the optimal k, and also estimate the maximum request arrival rate supported by the system.

Future Directions: In Chapter 7 we describe future research directions that build on the fork-join queueing framework. A future direction of particular interest is to develop new erasure codes for fast content download. After our work on delay analysis of coded storage, one such class of codes called availability codes was proposed in [11]. These codes allow parallel reads from disjoint groups of nodes and can help support more users simultaneously. Recently, [12] analyzed the performance of these codes using the fork-join framework.

1.3.3 Erasure Coding for Smooth Streaming

Streaming services such as Netflix and YouTube contribute to more than 50% of today's Internet traffic. Unlike traditional file transfer where only total delay matters, streaming requires fast and in-order playback of packets. In streaming communication, the cost of redundancy is the bandwidth used to retransmit lost packets transmit redundant packet combinations to recover from packet losses. Thus the source needs to strike a balance between transmitting new and old packets. In Part III we present erasure codes that combine packets in an effective way to ensure smooth playback with minimum interruptions. Some key insights are summarized below.

Effect of Feedback: Feedback about past packet erasures can help the source adapt future transmissions. Chapter 8 shows that frequent feedback can drastically improve smoothness of playback. We propose easy-to-implement erasure codes that give a close to optimal trade-off between throughput and smoothness of packet delivery.

Multicast Streaming: When there are multiple users with varying channel qualities requesting the same stream, the next packet required by one user may be redundant for the other. In Chapter 9 we propose and analyze coding schemes that balance the priorities given to different users.

In Chapter 10 we present interesting future directions including dynamic bandwidth, packet dropping, and streaming from multiple sources. More broadly, the analysis of *in-order* streaming blends tools from renewal processes and large deviations with traditional coding theory. These tools have are also used in interesting subsequent work; see e.g. [13–15].

Finally, Chapter 11 concludes the thesis and presents broader future research directions beyond the realm of cloud infrastructure.

1.4 Bibliographical Notes

Preliminary versions of Chapter 2 appear in the papers

[16] G. Joshi, E. Soljanin, and G. Wornell, "Efficient Replication of Queued Tasks for Latency Reduction in Cloud Systems", Proceedings of the Allerton Conference on Communication, Control and Computing, Oct 2015

[17] G. Joshi, E. Soljanin, and G. Wornell, "Efficient Redundancy Techniques for Latency Reduction in Cloud Systems", submitted to ACM Transactions on Modeling and Performance Evaluation of Computing Systems, arXiv:1506.0339

Preliminary versions of Chapter 3 appear in the papers

[18] D. Wang, G. Joshi, and G. Wornell, "Efficient Task Replication for Fast Response Times in Parallel Computation", Proceedings of ACM SIGMETRICS, June 2014

[19] D. Wang, G. Joshi, and G. Wornell, "Using Straggler Replication to Reduce Latency in Large-Scale Parallel Computing", Proceedings of ACM SIGMETRICS Distributed Cloud Computing (DCC) Workshop, Jun 2015, arXiv:1503.03128

Preliminary versions of Chapter 6 appear in the papers

[20] G. Joshi, Y. Liu, and E. Soljanin, "Coding for Fast Content Download", Proceedings of Allerton Conference on Communication, Control and Computing, Oct 2012

[21] G. Joshi, Y. Liu, and E. Soljanin, "On Delay-Storage Trade-offs in Content Download from Coded Distributed Storage Systems", IEEE Journal on Selected Areas of Communications, May 2014

[22] G. Joshi, E. Soljanin, and G. Wornell, "Queues with Redundancy: Latency-Cost Analysis", Proceedings of ACM Sigmetrics Mathematical Modeling and Analysis (MAMA) Workshop, Jun 2015

Preliminary versions of Chapter 8 and Chapter 9 appear in papers

[23] G. Joshi, Y. Kochman, and G. Wornell, "Effect of Block-wise Feedback on the Throughput-Delay Trade-off in Streaming", Proceedings of the INFOCOM Workshop on Communication and Networking Techniques for Contemporary Video, Apr 2014

[24] G. Joshi, Y. Kochman, and G. Wornell, "Throughput-Smoothness Trade-offs in Multicasting an Ordered Packet Stream", Proceedings of the IEEE International Conference on Network Coding, June 2014

[25] G. Joshi, Y. Kochman, and G. Wornell, "On Throughput-Smoothness Trade-offs in Streaming Communication", arXiv:1511.08143, Nov 2015

Part I

Task Replication in Cloud Computing

Chapter 2

Replication of Queued Tasks

2.1 Introduction

An increasing number of applications are now hosted on the cloud, and latency is an important quality metric in these services. Users expect fast response from the cloud services, as seamless as using a personal computer to run the application. And this requirement is becomes more stringent with the emergence of more interactive and collaborative applications.

A major advantage of hosting applications on the cloud is that the large-scale sharing of resources provides scalability and flexibility. However, a side-effect of the loosely coordinated sharing of resources is the variability in the latency experienced by the users. This can be due to various factors such as queueing, pre-emption by other jobs with higher priority, server outages etc. The problem becomes further aggravated when the user is executing a job with several parallel tasks on the cloud, because the slowest task becomes the bottleneck in job completion. Thus, ensuring seamless, low-latency service to the end-user is a challenging problem in cloud systems.

One method to reduce latency that is gaining significant attention in recent years is the use of redundancy. In cloud computing, running a task on multiple machines and waiting for the earliest copy to finish can significantly reduce the latency [7]. However, redundancy can result in increased use of resources such as computing time, and network bandwidth. For example, in frameworks such as Amazon EC2

and Microsoft Azure which offer computing as a service, the server time spent is proportional to the money spent in renting the machines. In this work we provide an understanding of when the benefits of task replication in cloud computing outweigh the additional computing and network cost.

2.1.1 Related Works

Scheduling for Parallel Computing: There is a rich literature on scheduling for parallel computing, especially by the 1990s. One line of work is bin-packing strategies that aim to maximize the efficient of packing jobs with different computing time and resource requirements into processors. This is an NP-complete problem in general, but there are good heuristics available [26, 27]. Another line of work is load-balancing strategies that take into account the existing load on the servers when assigning jobs. Some examples are work stealing [28], where lightly loaded servers steal jobs from heavily loaded servers. To avoid polling all the servers, randomized load balancing strategies such as power-of-choice scheduling [29, 30] assign each job to the shortest of d randomly chosen queues.

A common thread in these approaches is that they need estimates of the resource requirements of jobs, and queue lengths and memory utilization at the servers. Cloud computing frameworks consists of thousands of non-dedicated, geographically dispersed servers that are being shared by many processes simultaneously. Thus, centralized and scalable monitoring of the system to get complete and accurate estimates of the job sizes and server load is very challenging. This calls for 'stateless' scheduling strategies that can handle random fluctuations in response time without monitoring the servers.

Systems Work on Task Replication: The idea of task replication to cope with server variability started to be used in late 1990s and early 2000s [31, 32]. It was implemented at a large-scale in Google's MapReduce [33] via the back-up tasks option. Several recent works in systems such as [34, 35] further explore techniques to launch redundant replicas of straggling tasks are launched to reduce latency. Although the use of redundancy has been explored in systems literature, there is little work on

the rigorous analysis of how it affects latency, and in particular the cost of resources. Next we review some of the theoretical work on the effect of redundancy on queueing delay and resource utilization in cloud systems.

Note that in general, the use of redundancy to reduce latency is not new. One of the earliest instances is the use of multiple routing paths [36] to send packets in networks. See [37, Chapter 7] for a detailed survey of other related work. Recently a similar idea has been studied in systems [38].

Exponential Service Time: In distributed storage systems, erasure coding can be used to store a content file on n servers such that it can be recovered by accessing any k out of the n servers. In [20, 21] we model this as an (n, k) fork-join queueing systems and found bounds on the expected latency with exponential service time. This is a generalization of the (n, n) fork-join system, which was actively studied in queueing literature [8–10] around two decades ago. In recent years, there is a renewed interest in fork-join queues due to their application to distributed computing frameworks such as MapReduce.

Another related model with a centralized queue instead of queues at each of the n servers was analyzed in [39]. Most recently, an analysis of latency with heterogeneous task classes for the replicated ($k = 1$) case is presented in [40]. Other related works include [12, 41, 42].

General Service Time: Few practical systems have exponentially distributed service time. For example, studies of download time traces from Amazon S3 [43, 44] indicate that the service time is not exponential in practice, but instead a shifted exponential. For service time distributions that are 'new-worse-than-used' [45], it is shown in [46] that it is optimal to replicate each task at maximum number of servers. The choice of scheduling policy for new-worse-than-used (NWU) and new-better-than-used (NBU) distributions is also studied in [47–49]. The NBU and NWU notions are closely related to the log-concavity of service time studied in this work.

The Cost of Redundancy: If we assume exponential service time then redundancy does not cause any increase in cost of server time. But since this is not true in practice, it is important to determine the cost of using redundancy. Simulation

results with non-zero fixed cost of removal of redundant requests are presented in [48]. The total computing time $\mathbb{E}[C]$ spent per job is considered in [18, 50] (Chapter 3 of this thesis) for a distributed system without considering queueing of requests. In [22] (Chapter 6 of this thesis) we present a latency-cost analysis of the (n, k) fork-join system, which generalizes the replication framework considered in this chapter.

2.1.2 Contributions

Assigning a task to multiple servers and waiting for the earliest copy to finish is an effective method to combat the variability in response time of individual servers, and thus reduce average latency. But replication may result in higher cost of computing resources, as well as an increase in queueing delay due to higher traffic load. Thus it is non-trivial to answer fundamental design questions such as:

1. How many replicas to launch?

2. Which servers to assign the replicas to?

3. When to issue and cancel redundant tasks?

This chapter provides a framework to answer such questions about queues with redundancy, and understand when and how task replication gives a cost-efficient latency reduction. A key insight is that a property called the 'log-concavity' of the task service distribution is a key factor in determining whether replication helps. If the service distribution is log-convex, then adding maximum replication reduces both latency (waiting time plus service time) as well as cost. And if it is log-concave, then less redundancy, and early cancellation of redundant tasks is more effective.

The rest of the chapter is organized as follows. In Section 2.2 we describe the system model, and the performance metrics used to compare replication strategies. In Section 2.4 we first consider the case when each task is replicated at all servers, and determine the best way to cancel redundant tasks. In Section 2.5 we generalize to partial replication and gain insights into the number of replicas, choice and servers, and when to cancel the redundant replicas.

2.2 Problem Formulation

2.2.1 System Model

Consider a distributed system with n statistically identical servers. Tasks arrive to the system at rate λ per second according to a Poisson process[1]. All incoming tasks are assigned to first-come first-served queues at one or more servers. The number of replicas and how they are issued and canceled is same for all tasks. The replication strategy is defined concretely in Section 2.2.2 below.

After a task reaches the head of its queue, the time taken to serve it can be random due to various factors such as disk seek time, network congestion, and sharing of computing resources between multiple processes. We model it by the service time $X > 0$, with cumulative distribution function (CDF) $F_X(x)$ and assume that it is i.i.d. across requests and servers. Dependence of service time across servers can be modeled by adding a constant to service time X. We use $\bar{F}_X(x) = \Pr(X > x)$ to denote the tail distribution (inverse CDF) of X. And the notation $X_{k:n}$ stands for the kth smallest of n i.i.d. random variables $X_1, X_2, \ldots, X_n$.

2.2.2 Replication Strategy

We focus on three parameters (r, π, c) of a replication strategy: the number of replicas r, choice of servers π and the cancellation policy c. Each task is replicated at r out of the n servers. The servers are chosen according to a scheduling policy π.

We focus on 'symmetric' scheduling policies π, defined formally as follows.

Definition 1 (Symmetric Scheduling Policy). *A scheduling policy is said to be symmetric if the expected fraction of tasks assigned to each server is equal across the servers.*

Examples of symmetric policies illustrated in Fig. 2-1 are:

---

[1]The Poisson assumption is required only for the exact analysis and bounds on latency. Other results on comparing different replication strategies holds for any arrival process.
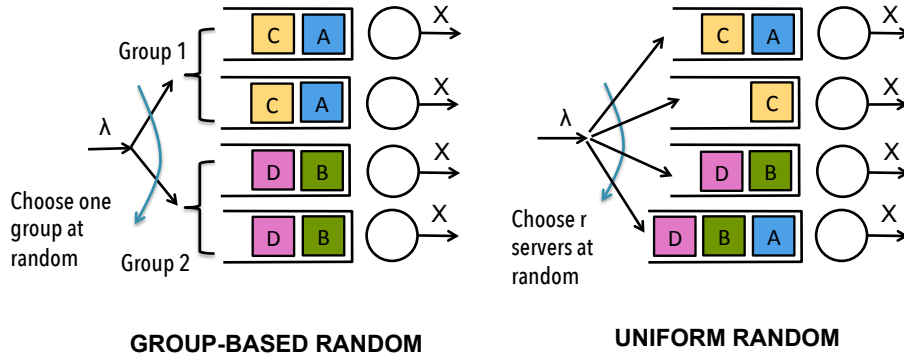
Figure 2-1: Examples of symmetric scheduling policies for $r = 2$ and $n = 4$.
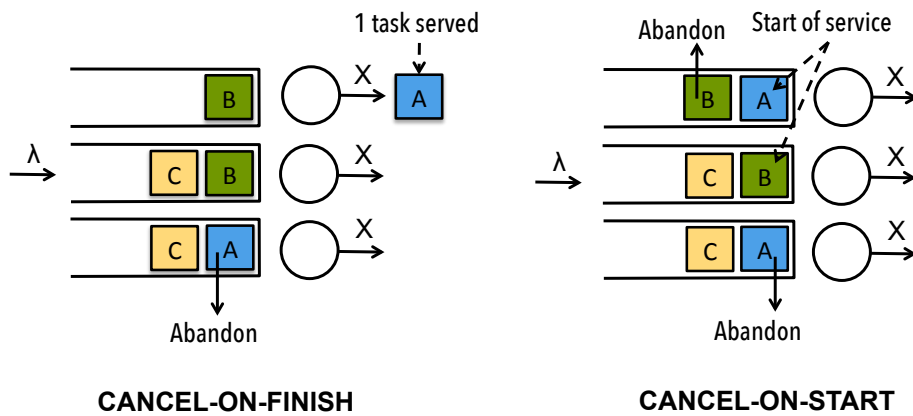
Figure 2-2: Illustration of the cancel-on-finish and cancel-on-finish policies for $r = 2$ and $n = 3$.

- **Group-based random:** This policy holds when r divides n. The n servers are divided into n/r groups of r servers each. Each task is replicated across all servers in one of these groups, chosen uniformly at random.

- **Uniform Random:** Each task is replicated at any r out of n servers, chosen uniformly at random.

Note that these two policies are 'stateless': they do not require information about the state of the servers (queue lengths, memory load etc.) or about the task size and resource requirements. Another stateless symmetric policy is round-robin scheduling. Some common state-aware policies that are also symmetric include join-the-shortest-queue, least-work-left, and power-of-choice scheduling.

The third parameter of the replication strategy (r, π, c) is the cancellation policy

c. We consider the following two cancellation policies illustrated in Fig. 2-2:

- **Cancel-on-finish:** When any one replica of a task is served, all other replicas are canceled and abandon their queues immediately.

- **Cancel-on-start:** When any one replica of a task reaches the head of its queue and starts service, all other replicas are canceled and abandon their queues immediately. If multiple replicas start service simultaneously, we retain any one chosen uniformly at random.

The cancel-on-start strategy has been previously explored in [35] and it is shown to be effective in reducing latency, without increasing load on the system.

Remark 1 (Relation to Power-of-choice Scheduling). *Power-of-choice scheduling proposed by Mitzenmacher in [29] is closely related to the replication strategy for any r, with $\pi =$ uniform random scheduling and $c =$ cancel-on-start. In power-of-r scheduling, a task is assigned to the shortest of r randomly chosen queues. Instead joining the shortest, we assign the task to all r queues, and retain the earliest replica that starts service. As a result the cancel-on-start policy is able to find the queue with the 'least work left' among the r queues. Thus cancel-on-start will give lower latency than power-of-r scheduling.*

Although we focus on these three parameters (r, π, c) here, there can be more general replication strategies. For example, instead of issuing replicas upfront, we can delay some of them to save the computing time spent by servers. Our analysis also provides insights into whether delaying replicas improves the performance of the system for a given task service distribution $\bar{F}_X$ (see Section 2.5.3).

2.2.3 Performance Metrics

We now define the metrics of the latency and cost that are used to compare different replication strategies (r, π, c). Our objective in the rest of the chapter will be to find the strategy that gives the best latency-cost trade-off.

Definition 2 (Expected Latency). *The expected latency $\mathbb{E}[T]$ is defined as the expected time from the arrival of a task until any one replica is served. It includes the waiting time in queue and the time spent at the servers until the task is served.*

Although $\mathbb{E}[T]$ is a good indicator of the average behavior, system designers are often interested in the tail $\Pr(T > t)$ of the latency. For many queueing problems, determining the distribution of response time T requires the assumption of exponential service time. In order to consider arbitrary, non-exponential service time distribution F_X, we settle for analyzing the expected latency $\mathbb{E}[T]$ here. In Section 4.1.1 we give preliminary insights on extending our analysis to determine the distribution of T.

Definition 3 (Expected Computing Cost). *The expected computing cost $\mathbb{E}[C]$ is the expected total time spent by the servers task and its replicas.*

Note that $\mathbb{E}[C]$ does not include the time spent in the queue. Thus, if we use the cancel-on-start policy that cancels replicas before they start service, then $\mathbb{E}[C] = \mathbb{E}[X]$ for any r and π. On the other hand, the cancel-on-finish policy may result in higher $\mathbb{E}[C]$ because multiple servers may spend redundant time serving replicas of the same task. In computing-as-a-service frameworks such as Amazon Web Services (AWS), the expected computing cost is proportional to money spent on renting machines from the cloud.

Although we focus on $\mathbb{E}[C]$ to account for the cost of redundancy, other practical costs can be accounted for as follows:

- In practice there will be a non-zero delay in canceling replicas when a task starts or finishes service. The cancellation delay can be added to the $\mathbb{E}[C]$ as additional time spent by the servers.

- The signaling overhead of making Remote-Procedure Calls (RPCs) to issue and cancel redundant tasks is proportional to the number of replicas r. To account for it, we can impose an upper limit on $r \leq r_{max}$.

2.3   Preliminary Concepts

We now present some preliminary concepts that are vital for understanding the results presented in the rest of the chapter.

2.3.1   Using $\mathbb{E}[C]$ to Compare Systems

Since the cost metric $\mathbb{E}[C]$ is the expected time spent by servers on each task, higher $\mathbb{E}[C]$ implies higher expected waiting time for subsequent tasks. As the arrival rate λ increases, the waiting time becomes a dominant part of the latency. The maximum supported arrival rate λ can be expressed in terms of $\mathbb{E}[C]$ as given by the following claim.

Claim 1 (λ_{max} in terms of $\mathbb{E}[C]$). *For a system of n servers with a given replication strategy (r, π, c), and any arrival process with rate λ, the maximum λ such that expected latency $\mathbb{E}[T] < \infty$ is*

$$\lambda_{max} = \frac{n}{\mathbb{E}[C]} \tag{2.1}$$

Proof of Claim 1. For a symmetric policy, the mean time spent by each server per task is $\mathbb{E}[C]/n$. Thus the server utilization is $\rho = \lambda \mathbb{E}[C]/n$. To keep the system stable such that $\mathbb{E}[T] < \infty$, the server utilization must be less than 1. The result in (2.1) follows from this.   □

Definition 4 (Service Capacity λ_{max}^*). *The service capacity of the system λ_{max}^* is defined as the maximum λ_{max} over all replication strategies (r, π, c).*

Definition 5 (High and Low Traffic Regimes). *When $\lambda \to \lambda_{max}^*$, the system is said to be in the high traffic regime, such that expected waiting time in queue dominates the latency $\mathbb{E}[T]$. When $\lambda \to 0$, the system is said to be in the low traffic regime, such that expected waiting time tends to zero and $\mathbb{E}[T]$ only comprises of the expected service time.*

From Claim 1 and Definition 4 we can imply that $\mathbb{E}[C]$ can be used to compare different replication policies in the high traffic regime, as given by Corollary 1 below.

Corollary 1. *Given two symmetric strategies (r, π, c) and (r', π', c'), the strategy that has lower $\mathbb{E}[C]$, also has lower $\mathbb{E}[T]$ in the high traffic regime $(\lambda \to \lambda^*_{max})$.*

Corollary 1 serves as a powerful technique to compare different replication strategies in Section 2.5.3.

2.3.2   Log-concavity of $\bar{F}_X$

If we replicate a task at r idle servers and wait for any 1 copy to finish, the expected computing cost $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$, where $X_{1:r} = \min(X_1, X_2, \ldots, X_r)$, the minimum of r i.i.d. realizations of random variable X. The behavior of this function $r\mathbb{E}[X_{1:r}]$ depends on when the tail distribution $\bar{F}_X$ of service time is 'log-concave' or 'log-convex'. Log-concavity of $\bar{F}_X$ is defined formally as follows.

Definition 6 (Log-concavity and log-convexity of $\bar{F}_X$)**.** *The tail distribution $\bar{F}_X$ is said to be log-concave (log-convex) if $\log \Pr(X > x)$ is concave (convex) in x for all $x \in [0, \infty)$.*

For brevity, when we say X is log-concave (log-convex), we mean that $\bar{F}_X$ is log-concave (log-convex). Lemma 1 below gives how $r\mathbb{E}[X_{1:r}]$ varies with r for log-concave (log-convex) $\bar{F}_X$. It is central to proving several key results in this chapter.

Lemma 1. *If X is log-concave (log-convex), then $r\mathbb{E}[X_{1:r}]$ is non-decreasing (non-increasing) in r.*

The proof of Lemma 1 can be found in Appendix A.

The numerical results in this chapter use the shifted exponential, and hyper exponential as examples of log-concave and log-convex distributions respectively. The shifted exponential, denoted by $\mathsf{ShiftedExp}(\Delta, \mu)$ is an exponential with rate μ, plus a constant shift $\Delta \geq 0$. The hyper-exponential distribution, denoted by $\mathsf{HyperExp}(\mu_1, \mu_2, p)$ is a mixture of two exponentials with rates μ_1 and μ_2 where the exponential with rate μ_1 occurs with probability p. Interestingly, the exponential distribution $\mathsf{Exp}(\mu)$ is both log-concave and log-convex. It is a special case of the shifted

exponential when $\Delta = 0$. Similarly, it is a special case of the hyper exponential distribution when $\mu_1 = \mu_2 = \mu$.

Log-concave distributions like the shifted exponential are more common in practical systems. Log-convex service times although less common, occur when there is high variability in task service time. For example, CPU service times are often approximated by the hyperexponential distribution. Many practical systems also have service times that are neither log-concave nor log-convex. We use the Pareto distribution $\mathsf{Pareto}(x_m, \alpha)$ as an example of a distribution that is neither log-concave nor log-convex. The Pareto distribution has been observed to fit service times in data centers [7,51]. Its tail distribution is given by,

$$\Pr(X > x) = \begin{cases} \left(\frac{x_m}{x}\right)^\alpha & x \geq x_m, \\ 1 & \text{otherwise.} \end{cases} \tag{2.2}$$

Log-concavity of X implies that X is 'new-better-than-used', a notion considered in [46]. Other names for new-better-than-used distributions are 'light-everywhere' in [48] and 'new-longer-than-used' in [49]. Many random variables with log-concave (log-convex) $\bar{F}_X$ are also light (heavy) tailed respectively, but neither property implies the other. Unlike the tail of a distribution which characterizes how the maximum $\mathbb{E}[X_{n:n}]$, behaves for large n, log-concavity (log-convexity) of characterizes the behavior of the minimum $\mathbb{E}[X_{1:n}]$, which is of primary interest in this work.

Some properties of log-concavity relevant to this work are given in Appendix A. We refer readers to [52] for more properties and examples of log-concave distributions.

2.3.3 Relative Task Start Times

Since the replicas of each task experience different waiting times in their respective queues, they may start service at different times. The relative start times of the replicas is an important factor affecting the latency and cost. We denote the relative start times by $t_1 \leq t_2 \leq \cdots t_n$ where $t_1 = 0$ without loss of generality. For instance, if $r = 3$ replicas of task start at absolute times 3, 4 and 7, then their relative start

times are $t_1 = 0$, $t_2 = 4 - 3 = 1$ and $t_3 = 7 - 3 = 4$. If we launch $r < n$ replicas of each task, then $t_{r+1}, \cdots t_n$ are infinite.

Let S be the time from when the earliest replica starts service, until any one replica finishes. It is the minimum of $X_1 + t_1, X_2 + t_2, \cdots X_n + t_n$, where X_i are i.i.d. The tail distribution of S is given by

$$\Pr(S > s) = \prod_{i=1}^{n} \Pr(X > s - t_n) \tag{2.3}$$

The computing cost C can be expressed in terms of S and t_i as follows.

$$C = S + (S - t_2)^+ + \cdots + (S - t_n)^+. \tag{2.4}$$

Using (2.4) we get several crucial insights in the rest of the chapter. For instance, in Section 2.5 we show that when $\bar{F}_X$ is log-convex, having $t_1 = t_2 = \cdots = t_n = 0$ gives the lowest $\mathbb{E}[C]$. Then using Claim 1 we can infer that $r = n$ with cancel-on-finish is optimal when $\bar{F}_X$ is log-convex.

2.4 Full Replication $(r = n)$

In this section we analyze the latency and cost with full replication $(r = n)$, and compare the two cancellation policies: cancel-on-finish and cancel-on-start. Since we replicate each task at all n servers, the choice of servers π is trivial. From this analysis we get the insight that cancel-on-start is better if $\bar{F}_X$ is log-concave. On the other hand, if $\bar{F}_X$ is log-convex, using cancel-on-finish is better.

2.4.1 Latency-Cost Analysis

Lemma 2. *If we launch $r = n$ replicas of each task and cancel-on-finish, the latency T of the system is equivalent in distribution to that of an $M/G/1$ queue with service time $X_{1:n}$.*

Proof. Consider the first task that arrives when all servers are idle. The n replicas
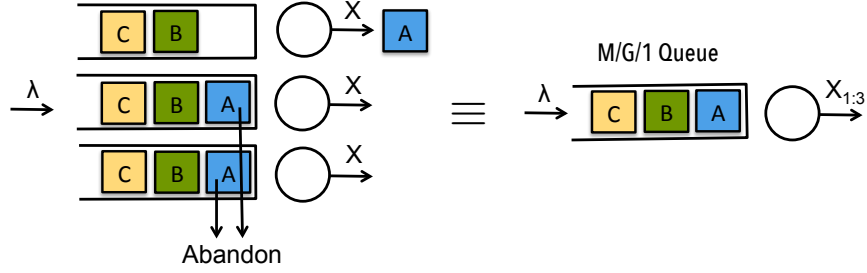
Figure 2-3: When $r = n$ and $c =$ cancel-on-finish, the system is equivalent to an $M/G/1$ queue with service time $X_{1:n}$, the minimum of n i.i.d. random variables $X_1, X_2, \ldots, X_n$.

start service at their respective servers simultaneously. The earliest replica finishes after time $X_{1:n}$, and all others are canceled immediately. So, the replicas of the subsequent task also start simultaneously at the n servers as illustrated in Fig. 2-3. Thus, the arrival and departure events, and as a result the latency T is equivalent in distribution to an $M/G/1$ queue with service time $X_{1:n}$. $\qquad \square$

Theorem 1. *If we launch $r = n$ replicas of each task and cancel-on-finish, the expected latency and computing cost are given by*

$$\mathbb{E}\left[T\right] = \mathbb{E}\left[T^{M/G/1}\right] = \mathbb{E}\left[X_{1:n}\right] + \frac{\lambda \mathbb{E}\left[X_{1:n}^2\right]}{2(1 - \lambda \mathbb{E}\left[X_{1:n}\right])} \tag{2.5}$$

$$\mathbb{E}\left[C\right] = n \cdot \mathbb{E}\left[X_{1:n}\right] \tag{2.6}$$

where $X_{1:n} = \min(X_1, X_2, \ldots, X_n)$ for i.i.d. $X_i \sim F_X$.

Proof. By Lemma 2, the latency for $r = n$ and cancel-on-finish is equivalent in distribution to an $M/G/1$ queue with service time $X_{1:n}$. The expected latency of an $M/G/1$ queue is given by the Pollaczek-Khinchine formula (2.5). The expected cost $\mathbb{E}\left[C\right] = n\mathbb{E}\left[X_{1:n}\right]$ because each of the n servers spends $X_{1:n}$ time on each task. This can also be seen by noting that $S = X_{1:n}$ when $t_i = 0$ for all i, and thus $C = nX_{1:n}$ in (2.4). $\qquad \square$

In Corollary 2 and Corollary 3 we characterize how $\mathbb{E}\left[T\right]$ and $\mathbb{E}\left[C\right]$ vary with n. The behavior of $\mathbb{E}\left[C\right]$ follows from Lemma 1.

Corollary 2. *For any service distribution F_X, the expected latency $\mathbb{E}\left[T\right]$ in Theorem 1 is non-increasing with n.*

Corollary 3. *If $\bar{F}_X$ is log-concave (log-convex), then $\mathbb{E}\left[C\right]$ is non-decreasing (non-increasing) in n.*
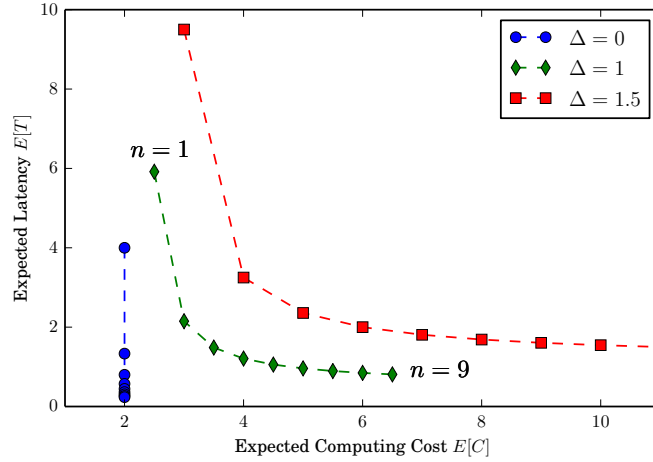
Figure 2-4: The service time $X \sim \Delta + \mathrm{Exp}(\mu)$ (log-concave), with $\mu = 0.5$, $\lambda = 0.25$. As n increases along each curve, $\mathbb{E}\left[T\right]$ decreases and $\mathbb{E}\left[C\right]$ increases. Only when $\Delta = 0$, latency reduces at no additional cost.
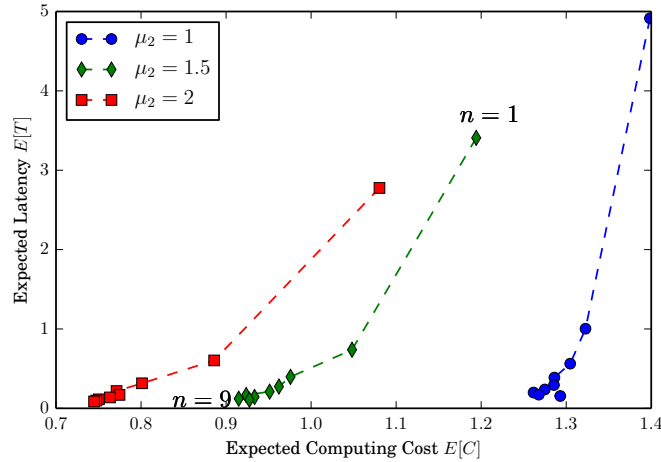
Figure 2-5: The service time $X \sim \mathsf{HyperExp}(0.4, \mu_1, \mu_2)$ (log-convex), with $\mu_1 = 0.5$, different values of μ_2, and $\lambda = 0.5$. Expected latency and cost both reduce as n increases along each curve.

Fig. 2-4 and Fig. 2-5 show the expected latency versus cost for log-concave and log-convex $\bar{F}_X$ respectively. In Fig. 2-4, the arrival rate $\lambda = 0.25$, and X is shifted

exponential $\mathsf{ShiftedExp}(\Delta, 0.5)$, with different values of Δ. For $\Delta > 0$, there is a trade-off between expected latency and cost. Only when $\Delta = 0$, that is, X is a pure exponential (which is generally not true in practice), we can reduce latency without any additional cost. In Fig. 2-5, arrival rate $\lambda = 0.5$, and X is hyperexponential $\mathsf{HyperExp}(0.4, 0.5, \mu_2)$ with different values of μ_2. We get a simultaneous reduction in $\mathbb{E}[T]$ and $\mathbb{E}[C]$ as n increases. The cost reduction is steeper as μ_2 increases.

Let us now analyze full replication $(r = n)$ with the cancel-on-start cancellation policy, where we cancel redundant tasks as soon as any task reaches the head of its queue. Intuitively, cancel-on-start can save computing cost, but the latency could increase due to the loss of diversity advantage provided by retaining redundant tasks.

Lemma 3. *If we launch $r = n$ replicas of each task and cancel-on-start, the latency T of the system is equivalent in distribution to that of an $M/G/n$ queue with service time X.*

Proof. With the cancel-on-start cancellation policy, as soon as any replica of a task reaches the head of its queue, all others are canceled immediately. The redundant replicas help find the queue with the least work left, and exactly one replica of each task is served by the first server that becomes idle. Thus, as illustrated in Fig. 2-6, the latency is equivalent in distribution to an $M/G/n$ queue. $\qquad\square$

Theorem 2. *If we launch $r = n$ replicas of task and cancel-on-start, the expected latency and computing cost are given by*

$$\mathbb{E}[T] = \mathbb{E}\left[T^{M/G/n}\right], \tag{2.7}$$

$$\mathbb{E}[C] = \mathbb{E}[X], \tag{2.8}$$

where $T^{M/G/n}$ is the response time of an $M/G/n$ queueing system with service time $X \sim F_X$.

The proof of Theorem 2 follows directly from Lemma 3. The exact analysis of mean response time $\mathbb{E}\left[T^{M/G/n}\right]$ has long been an open problem in queueing theory.
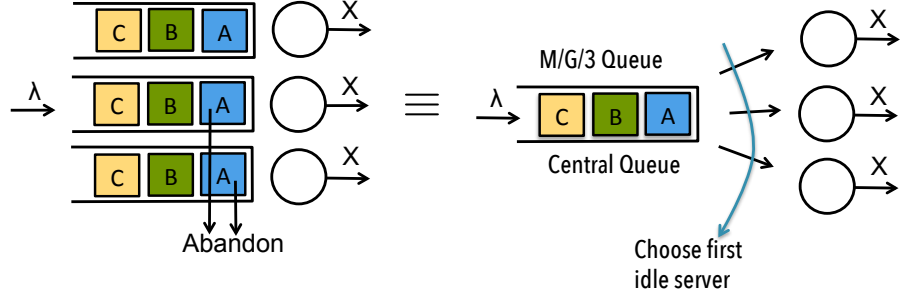
Figure 2-6: When $r = n$ and $c = $ cancel-on-start, the system is equivalent to an $M/G/n$ queueing system with each server taking time $X \sim F_X$ to serve task, i.i.d. across servers and tasks.

A well-known approximation given by [53] is,

$$\mathbb{E}\left[T^{M/G/n}\right] \approx \mathbb{E}\left[X\right] + \frac{\mathbb{E}\left[X^2\right]}{2\mathbb{E}\left[X\right]^2}\mathbb{E}\left[W^{M/M/n}\right] \tag{2.9}$$

where $\mathbb{E}\left[W^{M/M/n}\right]$ is the expected waiting time in an $M/M/n$ queueing system with load $\rho = \lambda\mathbb{E}\left[X\right]/n$. It can be evaluated using the Erlang-C model [54, Chapter 14]. and is given by

$$\mathbb{E}\left[W^{M/M/n}\right] = \frac{\rho(n\rho)^n}{n!(1-\rho)^2\lambda}\left(\sum_{i=0}^{n-1}\frac{(n\rho)^i}{i!} + \frac{(n\rho)^n}{n!(1-\rho)}\right)^{-1}. \tag{2.10}$$

2.4.2 Cancel-on-finish or Cancel-on-start?

Using Theorem 1 and Theorem 2 we can now compare the cancel-on-finish and cancel-on-start policies. First let us compare the computing cost $\mathbb{E}\left[C\right]$ with the two cancellation policies. Corollary 4 below follows from Lemma 1.

Corollary 4. *If $\bar{F}_X$ is log-concave (log-convex), then $r = n$ and cancel-on-start gives lower (higher) $\mathbb{E}\left[C\right]$ than $r = n$ with cancel-on-finish.*

Next we compare the latency $\mathbb{E}\left[T\right]$ in two extreme traffic regimes: low traffic ($\lambda \to 0$) and high traffic ($\lambda \to \lambda_{max}^*$), where λ_{max}^* is the service capacity of the system introduced in Definition 4.

46
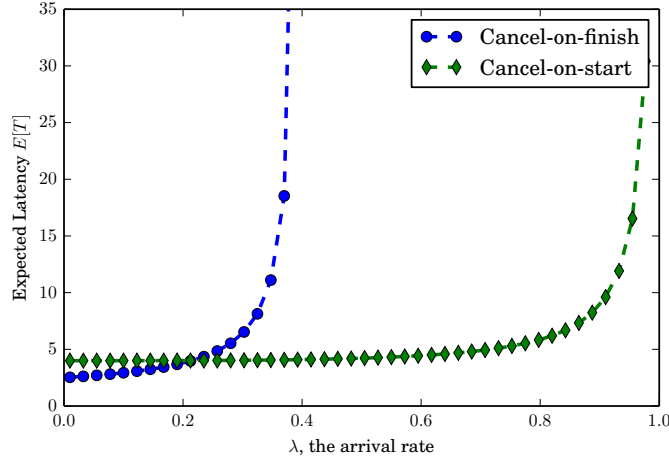
Figure 2-7: For $r = n = 4$ and service time $X \sim \mathsf{ShiftedExp}(2, 0.5)$ which is log-concave, the cancel-on-start policy is better in the high traffic regime, as given by Corollary 5.
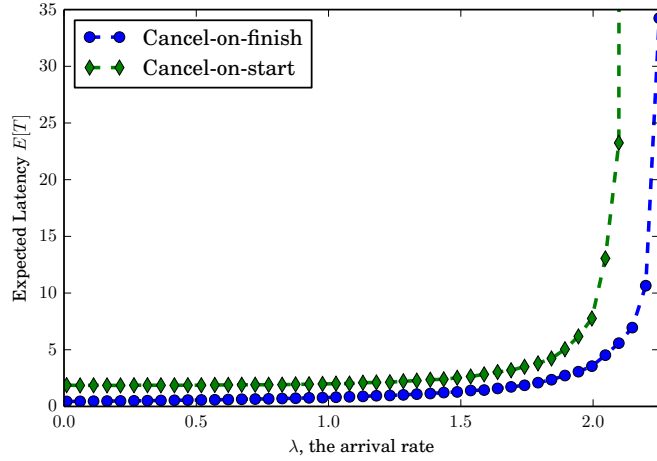
Figure 2-8: For $r = n = 4$ and $X \sim \mathsf{HyperExp}(0.1, 1.5, 0.5)$, which is log-convex, the cancel-on-start policy is worse in both low and high traffic regimes, as given by Corollary 5.

Corollary 5. *Cancel-on-finish gives lower* $\mathbb{E}[T]$ *than cancel-on-start in low traffic for any* X. *In high traffic, cancel-on-start gives lower (higher)* $\mathbb{E}[T]$ *than cancel-on-finish if* $\bar{F}_X$ *is log-concave (log-convex).*

The low traffic insights in Corollary 5 follow by substituting $\lambda = 0$ in Theorem 1 and Theorem 2. By Corollary 1, in the high traffic regime, the system with lower $\mathbb{E}[C]$ has lower $\mathbb{E}[T]$. Thus the high traffic insights in Corollary 5 follow from Corollary 4.

47

Fig. 2-7 and Fig. 2-8 illustrate Corollary 5. Fig. 2-7 shows a comparison of $\mathbb{E}[T]$ with the cancel-on-finish and cancel-on-start cancellation policies for $r = n = 4$, and service time $X \sim \mathsf{ShiftedExp}(2, 0.5)$. We observe that cancel-on-start gives lower $\mathbb{E}[T]$ in the high traffic regime. In Fig. 2-8 we observe that when X is $\mathsf{HyperExp}(0.1, 1.5, 0.5)$ which is log-convex, cancel-on-start is worse in both small and large traffic regimes.
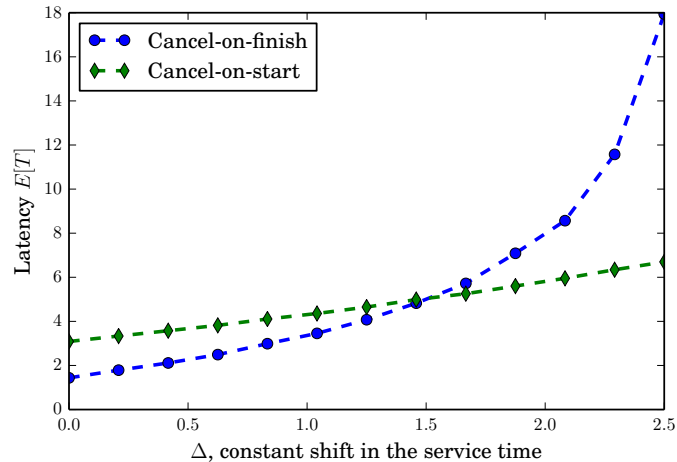
Figure 2-9: For the full replication of tasks at $n = 4$ servers, with shifted exponential service time $X = \mathsf{ShiftedExp}(\Delta, 0.5)$, cancel-on-start gives lower latency for larger Δ. The task arrival rate $\lambda = 0.25$.

In general, cancel-on-start is better when X is less variable (lower coefficient of variation). For example, a comparison of $\mathbb{E}[T]$ with cancel-on-finish and cancel-on-start as Δ, the constant shift of service time $\mathsf{ShiftedExp}(\Delta, \mu)$ varies is illustrated in Fig. 2-9. When Δ is small, there is more randomness in the service time of a task, and hence keeping redundant tasks running gives more diversity, and thus lower $\mathbb{E}[T]$. But as Δ increases, task service times are more deterministic due to which it is better to cancel the redundant tasks early.

2.5 Partial Replication $(r \leq n)$

For applications with a large number of servers n, full replication of each task can be expensive due to the network cost of issuing and cancel the redundant tasks. In

this section, we analyze the latency and cost with partial replication ($r \leq n$) and determine the best replication strategy in different regimes.

2.5.1 Latency-Cost Analysis: Group-based policy

The latency and cost with partial replication is hard in general, but is tractable for the group-based random scheduling policy. Recall that in the group-based random policy, each task is replicated across one of n/r groups chosen uniformly at random. Then each group behaves like an independent system of r servers with full replication, and arrival rate $\lambda r/n$. Thus, the expected latency and cost with cancel-on-finish and cancel-on-start are given by Lemma 4 and Lemma 5 below. Their proof follows from Theorem 1 and Theorem 2, with n replaced by r and λ replaced by $\lambda r/n$ respectively.

Lemma 4 ($\pi =$ group-based random, $c =$ cancel-on-finish)**.** *If each task is replicated at r servers according to the group-based random policy, and tasks are canceled according to the cancel-on-finish policy, then the expected latency and cost are given by*

$$\mathbb{E}\left[T\right] = \mathbb{E}\left[X_{1:r}\right] + \frac{\lambda r \mathbb{E}\left[X_{1:r}^2\right]}{2(n - \lambda r \mathbb{E}\left[X_{1:r}\right])} \tag{2.11}$$

$$\mathbb{E}\left[C\right] = r\mathbb{E}\left[X_{1:r}\right] \tag{2.12}$$

From Corollary 1 and (2.15) we can infer that the maximum arrival rate λ that can be supported when each task is replicated at r servers and $c =$ cancel-on-finish is,

$$\lambda_{max} = \frac{n}{\mathbb{E}\left[C\right]} = \frac{n}{r\mathbb{E}\left[X_{1:r}\right]} \tag{2.13}$$

Lemma 5 ($\pi =$ group-based random, $c =$ cancel-on-start)**.** *If each task is replicated at r servers according to the group-based random policy, and tasks are canceled according*

49

to the cancel-on-start policy, then the expected latency and cost are given by

$$\mathbb{E}[T] = \mathbb{E}\left[T^{M/G/r}\right] \tag{2.14}$$

$$\mathbb{E}[C] = \mathbb{E}[X] \tag{2.15}$$

where $T^{M/G/r}$ is the response time of an $M/G/r$ queueing system with arrival rate $\lambda r/n$ and service time distribution F_X.

Using Lemma 4 and Lemma 5 we can find the number of replicas r, and the cancellation policy that gives the best latency-cost trade-off for any given service distribution F_X and arrival rate λ. For example, Fig. 2-10 shows the latency versus cost trade-off for $n = 12$ servers, and task service time $X \sim \mathsf{Pareto}(1, 2.2)$. The number of replicas r, which is also the size of each group increases along each curve, and different curves correspond to different values of arrival rate λ. As the arrival rate λ increases, replicating each task costs more and also increases the queueing delay. Thus the optimal r^* decreases as λ increases.
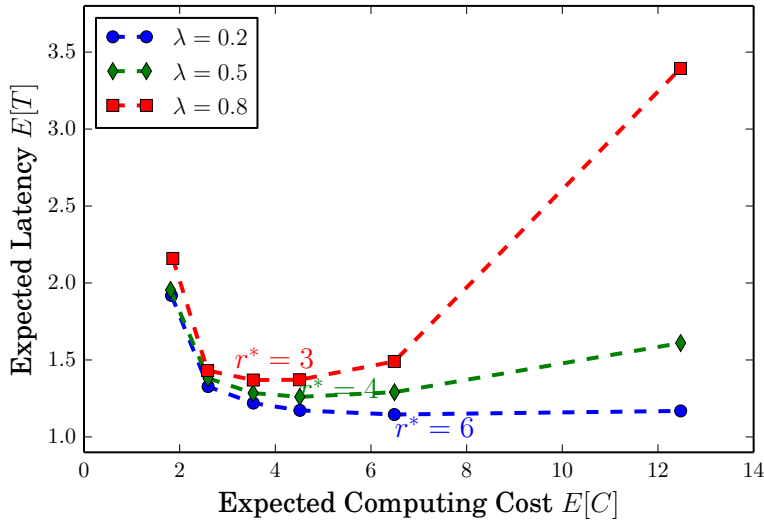
Figure 2-10: Latency versus cost for $n = 12$ servers with $c =$ cancel-on-finish, r increasing as 1, 2, 3, 4, 6, and 12 along each curve. The task service time $X \sim \mathsf{Pareto}(1, 2.2)$. As λ increases the replicas increase queueing delay. Thus the optimal r^* that minimizes $\mathbb{E}[T]$ shifts downward as λ increases.

2.5.2 Bounds on expected cost $\mathbb{E}[C]$

For other non-group-based symmetric policies, it is difficult to directly analyze of $\mathbb{E}[T]$ and $\mathbb{E}[C]$ because the replicas of the task can start service at different times in their respective queues. Instead, we develop bounds on $\mathbb{E}[C]$ for log-concave and log-convex $\bar{F}_X$, via which we can infer the best replication strategy in Section 2.5.3.

The scheduling policy π and cancellation policy c determine the relative starting times of the tasks $0 = t_1 \leq t_2 \leq \cdots \leq t_n$. For instance, if $c =$ cancel-on-start, $t_1 = 0$ and all other t_i are infinity because only one task enters service. In another instance if $\pi=$ group-based random, and $c =$ cancel-on-finish as considered in Section 2.5.1, $t_i = 0$ for $1 \leq i \leq r$, and $t_{i+1}, \ldots, t_n$ are infinite. For other symmetric policies the relative task start times are random. Theorem 3 below gives bounds on $\mathbb{E}[C]$ that are independent of the relative task start times.

Theorem 3. *Suppose a task is replicated at r out of n servers according to any symmetric scheduling policy. For any relative task start times t_i, $\mathbb{E}[C]$ can be bounded as follows.*

$$r\mathbb{E}[X_{1:r}] \geq \mathbb{E}[C] \geq \mathbb{E}[X] \qquad \text{if } \bar{F}_X \text{ is log-concave} \tag{2.16}$$

$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}] \quad \text{if } \bar{F}_X \text{ is log-convex} \tag{2.17}$$

If $t_i = 0$ for all $1 \leq i \leq n$, $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$ for any F_X. In the other extreme case, when $t_1 = 0$ and $t_2 = t_3 = \cdots = t_n = \infty$, $\mathbb{E}[C] = \mathbb{E}[X]$ for any F_X.

To prove Theorem 3 we take expectation on both sides in (2.4), and show that for log-concave and log-convex $\bar{F}_X$, we get the bounds in (2.16) and (2.17), which are independent of the relative task start times t_i. The detailed proof is given in Appendix B.

In Fig. 2-11 we show the bounds on $\mathbb{E}[C]$ alongside simulation values for different scheduling policies, when X is $\mathsf{ShiftedExp}(0.25, 0.5)$ (log-concave). We observe that the upper bound $r\mathbb{E}[X_{1:r}]$ is tight for group-based random scheduling (plotted only when r divided $n = 6$). The upper bound is slightly loose for other scheduling policies.
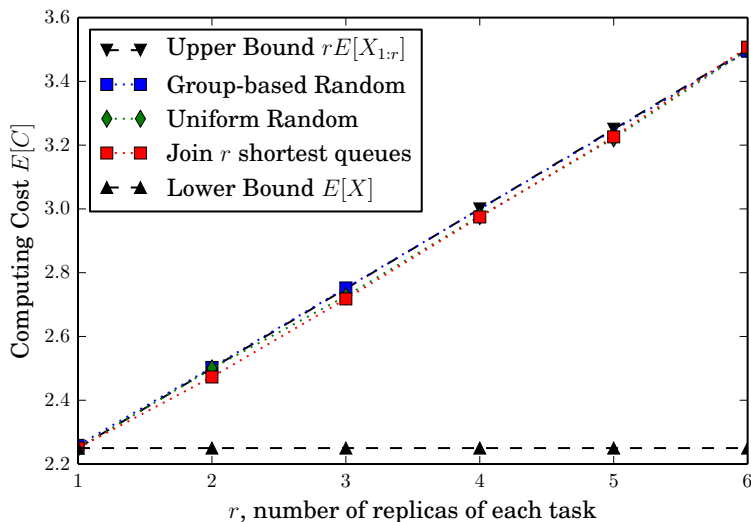
Figure 2-11: Expected cost $\mathbb{E}[C]$ versus r for $X \sim \mathsf{ShiftedExp}(0.25, 0.5)$, $n = 6$ servers and different scheduling policies. The upper bound $r\mathbb{E}[X_{1:r}]$ is exact for the group-based random policy, and fairly tight for other policies.

The upper and lower bounds coincide at $r = 1$ and $r = n$. Similarly for log-convex the actual $\mathbb{E}[C]$ is close to its lower bound $r\mathbb{E}[X_{1:r}]$.

In the sequel, we use the bounds in Theorem 3 to gain insights into the best replication strategy (r, π, c) when $\bar{F}_X$ is log-concave or log-convex.

2.5.3 Optimal Replication strategy

In this section, our objective is to determine the replication strategy (r, π, c) that achieve the best latency-cost trade-off. In particular, we study two extreme traffic regimes: low traffic $(\lambda \to 0)$ and high traffic $(\lambda \to \lambda^*_{max})$, where λ^*_{max} is the service capacity of the system introduced in Definition 4.

Low Traffic Regime $(\lambda \to 0)$

In the low traffic regime with $\lambda \to 0$, the waiting time in queue tends to zero. Thus all replicas of a task start service at the same time, irrespective of the scheduling policy π. We only consider the cancel-on-finish strategy here; cancel-on-start is equivalent to $r = 1$ with cancel-on-finish in low traffic. The expected latency and cost are then

given by

$$\mathbb{E}[T] = \mathbb{E}[X_{1:r}] \tag{2.18}$$

$$\mathbb{E}[C] = r\mathbb{E}[X_{1:r}] \tag{2.19}$$

From (2.18) it follows that the maximum redundancy strategy, $r = n$ with cancel-on-finish, minimizes latency $\mathbb{E}[T]$ for any service distribution $\bar{F}_X$. The expected cost $\mathbb{E}[C]$ in (2.19) however may increase or decrease depending upon the log-concavity of $\bar{F}_X$ as given by Lemma 1. If $\bar{F}_X$ is log-concave (log-convex), then $r = 1$ ($r = n$) minimizes the cost.

High Traffic Regime $(\lambda \to \lambda_{max}^*)$

The bounds on $\mathbb{E}[C]$ in Section 2.5.2, together with Corollary 1 serve as a powerful tool to compare different strategies in the high traffic regime. The replication strategy that minimizes $\mathbb{E}[C]$ also minimizes $\mathbb{E}[T]$ in high traffic.

Theorem 4 (Optimal Replication Strategy). *For log-convex $\bar{F}_X$, $r = n$ with cancel-on-finish is optimal. For log-concave $\bar{F}_X$, $r = n$ with cancel-on-start is optimal in high traffic.*

Proof. By Corollary 1, the optimal replication strategy in high traffic is the one that minimizes $\mathbb{E}[C]$. For log-convex $\bar{F}_X$, $r = n$ with cancel-on-finish achieves the lower bound $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$ in (2.17) with equality. Thus, $r = n$ with cancel-on-finish is the optimal strategy in the high traffic regime.

For log-concave $\bar{F}_X$, both $r = 1$ with cancel-on-finish and $r = n$ with cancel-on-start achieve the lower bound $\mathbb{E}[C] = \mathbb{E}[X]$ in (2.16) with equality. However, $r = n$ with cancel-on-start gives lower latency because the redundant tasks help find the shortest queue in the system. □

Due to the network cost of issuing and canceling the replicas, there may be an upper limit $r \le r_{max}$ on the number of replicas. The optimal strategy under this constraint is given by Lemma 6 below.

Lemma 6 (Optimal Strategy under $r \leq r_{max}$). *For log-convex $\bar{F}_X$, $r = r_{max}$ with cancel-on-finish is optimal. For log-concave $\bar{F}_X$, $r = r_{max}$ with cancel-on-start is optimal in high traffic.*

The proof is similar to Theorem 4 with n replaced by r_{max}.

The bounds on $\mathbb{E}[C]$ also help determine the optimal policy (r, π, c) when some parameters are fixed. For example, if only cancel-on-finish is possible, then the optimal r for any symmetric scheduling policy π is given by the following lemma.

Lemma 7 (Optimal r given c =cancel-on-finish). *For any symmetric policy π and c =cancel-on-finish, $r = 1$ ($r = n$) is optimal in high traffic for log-concave (log-convex) $\bar{F}_X$.*

Proof. If $\bar{F}_X$ is log-convex, it follows from Theorem 4 that the optimal strategy is $r = n$. For log-concave $\bar{F}_X$, both $r = 1$ with cancel-on-finish and $r = n$ with cancel-on-start achieve the lower bound $\mathbb{E}[C] = \mathbb{E}[X]$ in (2.17) with equality. But since c can only be cancel-on-finish, $r = 1$ is optimal. $\square$

In Fig. 2-12 and Fig. 2-13 we plot $\mathbb{E}[T]$ versus λ for different values of r. Each task is assigned to r out of $n = 6$ servers according to the group-based random policy. For each curve, the λ at which latency goes to infinity is given by,

$$\lambda_{max} = \frac{n}{r\mathbb{E}[X_{1:r}]} \tag{2.20}$$

In Fig. 2-12 the service time distribution is $\mathsf{ShiftedExp}(\Delta, \mu)$ (which is log-concave) with $\Delta = 1$ and $\mu = 0.5$. When $\lambda \to 0$, more redundancy (higher r) gives lower $\mathbb{E}[T]$, but in the high traffic regime, $r = 1$ gives lowest $\mathbb{E}[T]$. On the other hand in Fig. 2-13, for a log-convex distribution $\mathsf{HyperExp}(p, \mu_1, \mu_2)$, in the high traffic regime $\mathbb{E}[T]$ decreases as r increases.

Remark 2. *Lemma 7 was previously proven for new-better-than-used (new-worse-than-used) instead of log-concave (log-convex) $\bar{F}_X$ in [46, 48], using a combinatorial argument. Using Theorem 3, we get an alternative, and arguably simpler proof of*
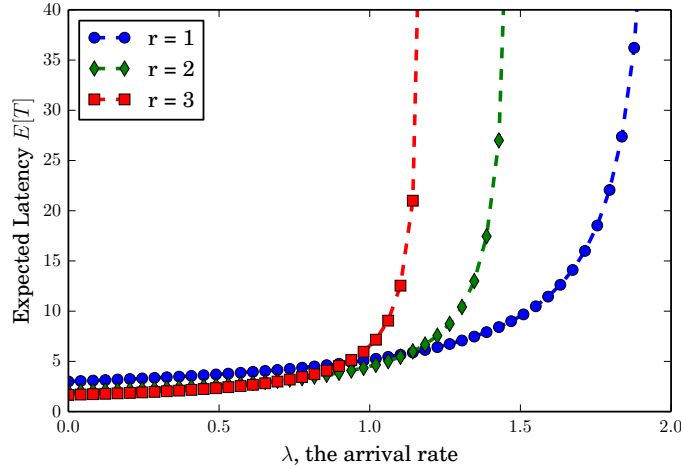
Figure 2-12: For $X \sim \mathsf{ShiftedExp}(1, 0.5)$ which is log-concave, less (more) replicas gives lower expected latency in the low (high) λ regime.
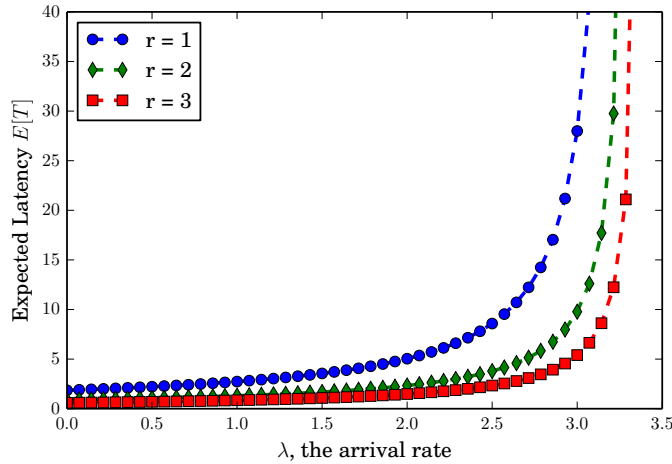
Figure 2-13: For $X \sim \mathsf{HyperExp}(p, \mu_1, \mu_2)$ with $p = 0.1$, $\mu_1 = 1.5$, and $\mu_2 = 0.5$ which is log-convex, more replicas (larger r) gives lower expected latency for all λ.

this result. Our version is somewhat weaker because log-concavity implies new-better-than-used but the converse is not true in general (see Property 3 in Appendix A).

Given r and $c =$ cancel-on-finish, we can also compare different policies π of choosing the r servers for each task. The choice of the r servers determines the relative starting times of the tasks. If all the r tasks start at the same time (true for the group-based random policy), $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$. By comparing with the bounds in Theorem 3 that hold for any relative task start times we get the following result.

Lemma 8 (Choosing π given r, and $c = $ cancel-on-finish). *Given r, if $\bar{F}_X$ is log-concave (log-convex), the symmetric policy that results in the tasks starting at the same time ($t_i = 0$ for all $1 \le i \le r$) results in higher (lower) $\mathbb{E}[T]$ in the high traffic regime than one that results in $0 < t_i < \infty$ for one or more i.*
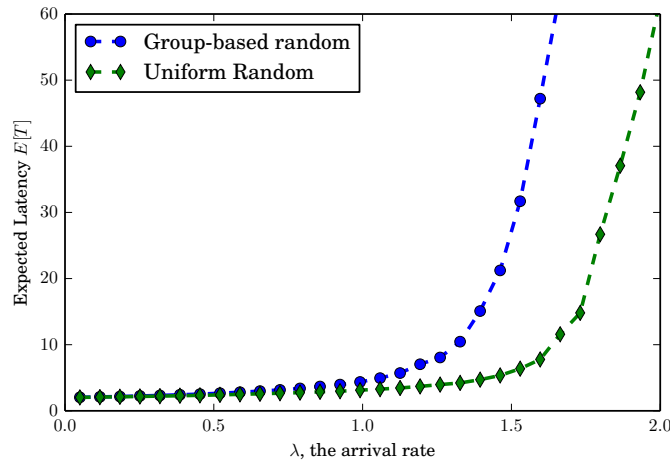
Figure 2-14: For service distribution $\mathsf{ShiftedExp}(1, 0.5)$ which is log-concave, uniform random scheduling (which staggers relative task start times) gives lower $\mathbb{E}[T]$ than group-based random for all λ. The system parameters are $n = 6$, $r = 2$.
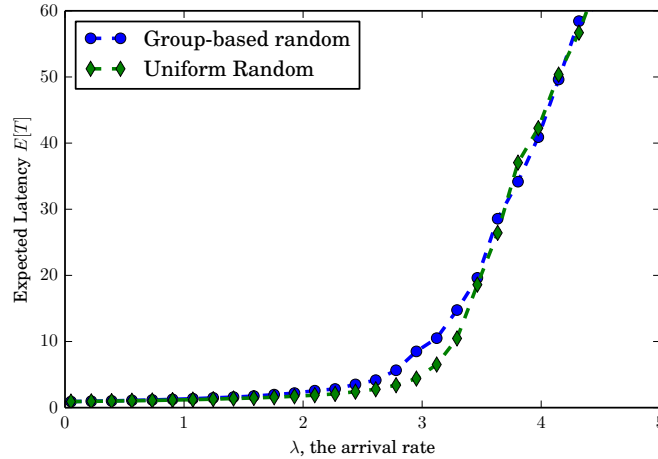
Figure 2-15: For service distribution $\mathsf{HyperExp}(0.1, 1.5, 0.5)$ which is log-convex, group-based scheduling gives lower $\mathbb{E}[T]$ than uniform random in the high λ regime. The system parameters are $n = 6$, $r = 2$.

Lemma 8 is illustrated in Fig. 2-14 and Fig. 2-15 for $n = 6$ and $r = 2$. The r tasks may start at different times with the uniform random policy, whereas they

always start simultaneously with the group-based random policy. Thus, in the high λ regime, that uniform random policy results lower latency for log-concave $\bar{F}_X$, as observed in Fig. 2-14. But for log-convex $\bar{F}_X$, group-based forking is better in the high λ regime as seen in Fig. 2-15. For low λ, uniform random policy is better for any $\bar{F}_X$ because it gives lower expected waiting time in queue.

Remark 3. *Lemma 8 helps compare a group-based policy with a non-group-based policy, but not two non-group-based policies. In general a policy that 'staggers' the relative task start times to a larger extent is better (worse) for log-concave (log-convex) $\bar{F}_X$. But the exact nature of the staggering of task start times that results in better latency-cost trade-off remains to be understood.*
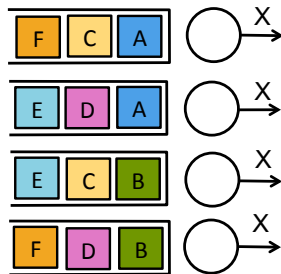
Figure 2-16: Diversity scheduling policy that staggers task start times

For example, a policy that staggers the task start times is illustrated in Fig. 2-16. The replicas of each task to queues that have different tasks waiting in front.

Remark 4 (Delaying replicas)**.** *Instead of issuing all replicas of a task upfront, the replicas could be delayed. Depending on the waiting time in each queue, these delays can result in relative task start times $0 = t_1 \leq t_2 \leq \cdots \leq t_r$ of the replicas. Then the bounds of $\mathbb{E}[C]$ in Theorem 3 can be used to infer whether delaying replicas improves the latency-cost performance of the replication strategy.*

	Log-concave service time		Log-convex service time	
	Latency-optimal	**Cost-optimal**	**Latency-optimal**	**Cost-optimal**
Number of replicas r	Low λ: $r = n$, High λ: $r = 1$	$r = 1$	$r = n$	$r = n$
When to cancel the replicas?	Low λ: Cancel-on-finish , High λ: Cancel-on-start	Cancel-on-start	Cancel-on-finish	Cancel-on-finish

Table 2.1: Summary of insights on how to replicate tasks and cancel them

2.6 Concluding Remarks

2.6.1 Summary

In cloud systems, the large-scale sharing of a pool of commodity servers results in random fluctuations in server response time. Also, there is limited centralized monitoring of the server loads, and the size of incoming computing tasks. As a result, we need scheduling policies that can handle service time variability and ensure fast execution of tasks, with little or no knowledge of the state of the system. In this work, we study the effectiveness of task replication in coping with service variability and reducing latency.

Intuitively, replicating tasks at multiple servers and waiting for the earliest copy reduces latency, but uses additional resources such as computing time at the servers. Replication can also increase queueing delay for subsequent tasks. We analyze the impact of redundancy on two metrics: the latency (expected service plus waiting time in queue) $\mathbb{E}[T]$, and the computing cost (total expected server time spent per job) $\mathbb{E}[C]$. Using these metrics our objective is to determine the best replication strategy: the number of replicas r, choice of servers π and cancellation policy c.

We identify that the *log-concavity* of service time is a key factor governing the choice of a redundancy strategy. The main insights are summarized in Table 2.1. Surprising, for log-convex service distributions, more redundancy reduces both latency and cost. Thus replication in fact improves the efficiency of the system of servers. On the other hand, for log-concave service time, more redundancy is bad in the high

traffic regime. Thus it is better to cancel redundant tasks early, or launch fewer replicas to begin with. Also, our computing cost metric $\mathbb{E}[C]$ serves as a powerful tool to compare different redundancy strategies under high traffic.

2.6.2 Future Directions

Several generalizations of the system model have been studied in recent works, while others open for future research. In Chapter 4 we describe some of these research directions, and present preliminary insights on two directions of particular interest to us: considering heterogeneous servers, and scheduling task replication when the service time F_X is unknown.

In Chapter 3 we study one such generalization studied in depth. Consider a job with many parallel tasks, such that all tasks need to finish to complete the job. Then the slowest tasks, or the stragglers become a bottleneck. In Chapter 3 we develop an understanding of how to best replicate these slowest tasks, or 'stragglers' to minimize latency with little or no additional resource usage.

Another generalization is to consider 'coding' of tasks instead of replication such that any k out of n tasks are sufficient to complete the job. Part II analyzes the interplay between latency and cost by introducing and analyzing the (n, k) fork-join system.

Chapter 3

Straggler Replication in Parallel Computing

3.1 Introduction

In Chapter 2 we considered replication of tasks in parallel computing at the task level, and determined the best way to replicate assuming that all tasks use the same replication strategy. Computing frameworks such as MapReduce/Hadoop [33] and Apache Spark [55] employ massive parallelization by dividing a large job into many tasks that can be executed parallely on different machines. These frameworks can be used to run optimization and machine learning algorithms that can be easily divided into independent parallel tasks, for example alternating direction method of multipliers (ADMM) [56] and Markov Chain Monte-Carlo (MCMC) [57].

A key challenge in executing a job that consists of a large number of parallel tasks is the latency in waiting for the slowest tasks, or the "stragglers" to finish. As pointed out in [7, Table 1], the latency of executing many parallel tasks could be significantly larger (140 ms) than the median latency of a single task (1 ms). MapReduce and Apache Spark launch a "backup" copy of the straggling tasks to speed up the job [33, 58]. This is also referred to as "speculative execution". A line of systems work [34, 35, 58] and references therein further developed this idea. For example, Apache Spark implements "speculative execution" to allow relaunching slow

running tasks [59].

While task replication has been studied in systems literature and also adopted in practice, there is not much work on mathematical analysis of replication strategies. In this chapter we develop a mathematical framework to analyze such straggler replication strategies. In particular, the choice of a straggler replication strategy involves optimizing the following aspects:

- Fractions of remaining tasks declared as stragglers

- How many replicas to launch for each straggler

- Whether to kill original copy or not

We characterize how these aspects impact the trade-off between latency and computing cost, and identify regimes where replicating a small fraction of tasks drastically reduces latency while also saving computing cost. These insights allow one to apply optimization to search for scheduling policies based on one's sensitivity to computing latency and computing cost.

3.1.1 Organization

The rest of this chapter is organized as follows. In Section 3.2 we introduce notation, formulate the problem, and define performance metrics used in this chapter. In Section 3.3 we provide an analysis of single-fork task replication policies and defer all proofs to Appendix D. Then in Section 3.4 we describe an algorithm that finds a good scheduling policy for execution time distributions that are not analytically tractable (e.g., empirical distributions from real-world traces). In Section 3.5 we conclude with a discussion of the implications and future perspectives.

3.2 Problem Formulation

We now describe the system model, and propose the performance metrics used to evaluate a task replication strategy.

3.2.1 Notation

First, we define some notation used in this chapter. Lower-case letters (e.g., x) denote a particular value of the corresponding random variable, which is denoted in upper-case letters (e.g., X). We denote the cumulative distribution function (c.d.f.) of X by $F_X(x)$. Its complement, the tail distribution is denoted by $\bar{F}_X(x) \triangleq 1 - F_X(x)$. We denote the upper end point of F_X by

$$\omega(F_X) \triangleq \sup\{x : F_X(x) < 1\}. \tag{3.1}$$

For i.i.d. random variables $X_1, X_2, \cdots, X_n$, we define $X_{j:n}$ as the j-th order statistic, i.e., the j-th smallest of the n random variables.

3.2.2 System Model

We consider a job consisting of n *parallel tasks*, where n is large[1] and each task is assigned to a different machine. We use the probability distribution F_X to model the random variation in machine response time due to factors such as congestion, queueing, virtualization, and competing jobs being run on the same machines, and assume this execution time distribution is independent and identically distributed (i.i.d.) across machines. The *identical* assumption of F_X implies tasks in this job are assigned to machines with processing power proportional to task size, with the simplest case being a group of homogeneous tasks are assigned to a group of homogeneous machines. The *independent* assumption of F_X could be satisfied when machine response times fluctuate independently over time, or when each new task (or new replica) is assigned to a new machine that is not previously used to run tasks of the current job. Note that we treat the variability that F_X captures as an exogenous factor from a user's perspective—in general a user renting machines from a cloud computing service has little or no control over other jobs that share the resources.[2]

---

[1]Analysis of real-world trace data shows that it is common for a job to contain hundreds or even thousands of tasks [51].

[2]A system designer may be able to influence this variability by adjusting the resource sharing among different jobs, another interesting direction that is beyond the scope of this work.

3.2.3 Scheduling Policy

A *scheduling policy* or *scheduler* assigns one or more replicas of each task to different machines, possibly at different time instants. The scheduler receives instantaneous feedback notifying it when a machine finishes its assigned task. There is *no intermediate feedback* indicating the status of processing of a task. Upon receiving notification that at least one replica of each of the n tasks has finished, the scheduler *kills all* the other replicas immediately. We focus our attention on a set of policies called *single-fork policies*, defined as follows.

Definition 7 (Single-fork scheduling policy). *A single-fork scheduling policy $\pi(p, r)$ launches all n tasks at time 0. It waits until $(1 - p)n$ tasks finish. For each of the remaining pn straggling tasks, it chooses one of the following two actions:*

- ***replicate and keep the original copy** ($\pi_{\text{keep}}(p, r)$): launch r new replicas;*

- ***replicate and kill the original copy** ($\pi_{\text{kill}}(p, r)$): kill the original copy and launch $r + 1$ new replicas.*

When the earliest replica of a task finishes, all the other remaining replicas of the same task are terminated.

Note that in both scenarios there are a total of $r + 1$ replicas running after the forking point. Fig. 3-1 illustrates these two cases of keeping or killing the original copy of a task. For simplicity of notation we assume that p is such that pn is an integer. We note that $p = 0$ corresponds to running n tasks in parallel and waiting for all to finish, which is the baseline case without any replication or killing any original tasks.

Remark 5 (Backup tasks in MapReduce). *The idea of "backup" tasks in Google's MapReduce [33], and "speculative execution" in Apache Spark [59] corresponds to a single-fork policy with $r = 1$ and π_{keep}. The value of p is tuned dynamically and hence not specified in [33]. The* `spark.speculation.quantile` *configuration corresponds to p in the single-fork policy.*

(a) Keep the original copy (π_{keep})   (b) Kill the original copy (π_{kill})
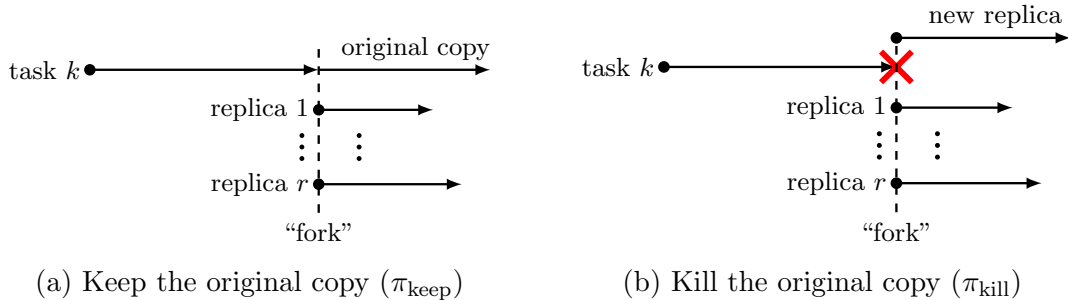
Figure 3-1: Single-fork policy illustration

Although we focus on single-fork policies in this chapter, the analysis can be generalized to multi-fork policies, where new replicas of straggling tasks are launched at multiple times during the execution of the job [60, Section 6.4]. Although forking multiple times can give a better latency-cost trade-off, the additional cost of launching and killing replicas may outweigh the incremental benefit.

3.2.4   Performance Metrics

We now define the latency and cost metrics used to compare straggler replication policies and understand when and how replication is useful.

Definition 8 (Expected latency). *Given a scheduling policy, the expected latency* $\mathbb{E}[T]$ *is the expected value of* T, *the time taken for at least one replica of each of the* n *tasks to finish. It can be expressed as*

$$\mathbb{E}[T] = \mathbb{E}\left[\max_{i \in \{1,2,\dots,n\}} T_i\right], \tag{3.2}$$

where T_i *is the time when at least one replica of task* i *finishes. More specifically, suppose the scheduler launches* r *replicas of each of the* n *tasks at times* $t_{i,j}$ *for* $j = 0, 1, 2, \dots r$, *then*

$$T_i = \min_{0 \leq j \leq r} (t_{i,j} + X_{i,j}), \tag{3.3}$$

where $X_{i,j}$ *are i.i.d. draws from the execution time distribution* F_X.

Definition 9 (Expected cost). *The expected computing cost $\mathbb{E}[C]$ is the sum of the running times of all machines, normalized by n, the number of tasks in the job. The running time is the time from when the task is launched on a machine, until it finishes, or is killed by the scheduler. More specifically, suppose the scheduler launches r replicas of each of the n tasks at times $t_{i,j}$ for $j = 0, 1, 2, \ldots r$, then*

$$C \triangleq \frac{1}{n} \sum_{i=1}^{n} \sum_{j=0}^{r} (T_i - t_{i,j})^+ , \tag{3.4}$$

where T_i is given in (3.3) and $(x)^+ = \max(0, x)$.

Infrastructure as a Service (IaaS) providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform charge users by the time and the number of machines used. Then the money spent by a user to rent the machines is proportional to our cost metric $\mathbb{E}[C]$.

Fig. 3-2 illustrates the execution of a job with two tasks, and evaluation of the corresponding latency T and cost C. Given two tasks, we launch two replicas of task 1 $t_{1,1} = 0$ and $t_{1,2} = 2$, and two replicas of task 2 at $t_{2,1} = 0$ and $t_{2,2} = 5$. The task execution times are $X_{1,1} = 8$, $X_{1,2} = 7$, $X_{2,1} = 11$, and $X_{2,2} = 5$. Machine M_1 finishes the task first at time $t = 8$, $T_1 = 8$ and the second replica running on M_2 is terminated before it finishes executing. Similarly, machine M_4 finishes task 2 at time $T_2 = 10$, and the replica running on M_3 is terminated. Thus the latency of the job is $T = \max\{T_1, T_2\} = 10$. The cost is the sum of all running times normalized by n, i.e., $C = (8 + 6 + 10 + 5)/2 = 14.5$.

3.3 Single-fork policy analysis

In this section we analyze the trade-off between the performance metrics $\mathbb{E}[T]$ and $\mathbb{E}[C]$ for the single-fork policy defined in Definition 7. This analysis gives the insight that the choice of the best single fork policy $\pi(p, r)$ depends on two key characteristics of F_X: 1) whether the tail is heavy, light or exponential, and 2) whether the distribution is new-longer-than-used, new-shorter-than-used or neither. In Section 3.3.2 we
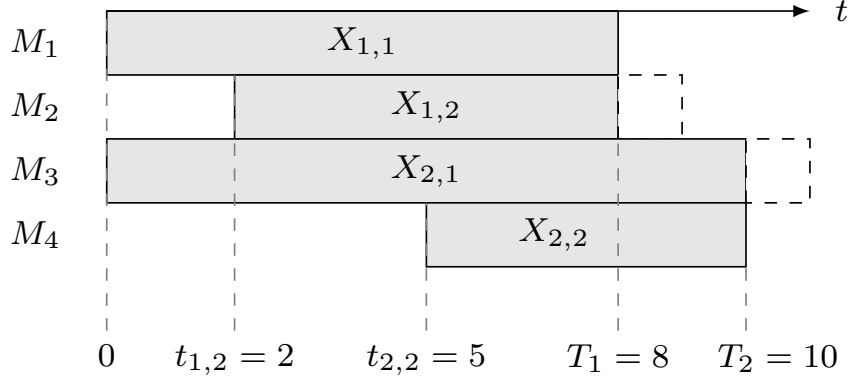
Figure 3-2: Illustration of T and C for a job with two tasks, and two replicas of each task. The latency $T = \max(8, 10) = 10$, and the computing cost is $C = (8 + 6 + 10 + 5)/2 = 14.5$.

demonstrate this insight that Shifted exponential and Pareto distributions that have been observed in cloud computing frameworks. All proofs are deferred to Appendix D.

3.3.1 Performance characterization

As defined in Definition 7, under the single-fork policy, a task is a "straggler" if it has not finished execution by the time of "forking". The amount of additional time after the "fork" point needed for a straggling task to complete depends on the number of replicas r and whether we kill or keep the original copy. We called this *residual execution time* and denote it by random variable Y. Its distribution F_Y can be expressed in terms of F_X as given by Lemma 9 below.

Lemma 9 (Residual execution time of stragglers). *As $n \to \infty$, the tail distribution $\bar{F}_Y$ of the residual execution time (after the forking point) of each of the pn straggling tasks is*

$$\bar{F}_Y(y) = \begin{cases} \bar{F}_X(y)^{r+1} & \text{for } \pi_{\mathrm{kill}}(p, r), \\ \frac{1}{p}\bar{F}_X(y)^r \, \bar{F}_X\left(y + F_X^{-1}(1-p)\right) & \text{for } \pi_{\mathrm{keep}}(p, r), \end{cases} \tag{3.5}$$

given that $p > 0$ and $y \geq 0$.

The proof of Lemma 9 is given in Appendix D. For example, if we kill the original

67

copy and choose $r = 2$, the tail of distribution $\bar{F}_Y = \bar{F}_X^2$, because two identical replicas with distribution F_X are launched at the forking point. For a job with a large number of tasks n, the expected latency and cost can be expressed in terms of F_X, F_Y and the single-fork policy parameters p and r as given by Theorem 5 below.

Theorem 5 (Single-Fork Latency and Cost). *For a computing job with n tasks, and task execution time distribution F_X, the latency and cost metrics as $n \to \infty$ are*

$$\mathbb{E}[T] = F_X^{-1}(1-p) + \mathbb{E}[Y_{pn:pn}], \tag{3.6}$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + (r+1)p \cdot \mathbb{E}[Y], \tag{3.7}$$

where F_Y is specified in Lemma 9 and $\mathbb{E}[Y_{pn:pn}]$ is the expected maximum of pn i.i.d. random variables drawn from F_Y.

Note that the choice of keeping or killing the original copy impacts the latency and cost metrics purely through the distribution of Y (as shown in (3.5)), and hence is not explicitly mentioned in Theorem 5.

A key observation from Theorem 5 is that the execution time before forking, $F_X^{-1}(1-p)$, is a quantity *independent with respect to n* and monotonically non-increasing with p, while the execution time after forking, $\mathbb{E}[Y_{pn:pn}]$, is monotonically non-decreasing with pn. In certain regimes, increasing p (and with proper choice of r), the time reduction in first stage outweighs the time increase in the second stage, reducing the overall execution latency. We now give a sketch of the proof of Theorem 5. A detailed proof can be found in Appendix D.

Proof Sketch of Theorem 5 . The latency T of a single fork policy $\pi(p, r; n)$ can be decomposed into $T^{(1)}$, the time to execute the first $(1-p)n$ tasks, and $T^{(2)}$, the time to execute the pn straggling tasks. The expected value of $T^{(1)}$ is

$$\mathbb{E}[T^{(1)}] = \mathbb{E}[X_{(1-p)n:n}] \approx F_X^{-1}(1-p) \quad \text{for large } n, \tag{3.8}$$

where (3.8) follows from the Central Value Theorem (Theorem 14) which states that the $((1-p)n)^{th}$ order statistic concentrates sharply around $F_X^{-1}(1-p)$ as $n \to \infty$.

The second part of the latency, $T^{(2)}$ is the maximum of the residual time Y for each of the pn straggling tasks finish. Thus, $\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}\left[Y_{pn:pn}\right]$. The behavior of the maximum order statistic of a large number of random variables is given by the Extreme Value Theorem Theorem 16, and its behavior as $n \to \infty$ depends on the domain of attraction of X and is given by Lemma 10 below. The domain of attraction $\mathrm{DA}\left(\cdot\right)$ of X in turn depends on its tail behavior (exponential, heavy or light). For example, exponentially decaying distributions belong to $\mathrm{DA}\left(\Lambda\right)$ while heavy-tailed distributions belong to $\mathrm{DA}\left(\Phi_\xi\right)$.

Similarly, the expected cost $\mathbb{E}\left[C\right]$ can be evaluated by decomposing it into two parts: before and after the replication of straggling tasks. The details can be found in the proof in Appendix D. $\qquad\square$

Lemma 10. *The asymptotic behavior of* $\mathbb{E}\left[Y_{pn:pn}\right]$ *as* $n \to \infty$ *is given by*

$$\mathbb{E}\left[Y_{pn:pn}\right] = \begin{cases} \tilde{a}_{pn}\gamma_{\mathrm{EM}} + \tilde{b}_{pn} & F_X \in \mathrm{DA}\left(\Lambda\right), \\ \tilde{a}_{pn}\Gamma\left(1 - \frac{1}{(r+1)\xi}\right) & F_X \in \mathrm{DA}\left(\Phi_\xi\right), \\ \tilde{b}_{pn} - \tilde{a}_{pn}\Gamma\left(1 + \frac{1}{((1-l)r+1)\xi}\right) & F_X \in \mathrm{DA}\left(\Psi_\xi\right). \end{cases}$$

where we set $l = 0$ *for* π_{kill} *and* $l = 1$ *for* π_{keep}, *and* $\mathrm{DA}\left(\cdot\right)$ *denotes domain of attraction, which can be determined for a distribution using Lemma 23 and Theorem 15. The terms* $\tilde{a}_{pn}$ *and* $\tilde{b}_{pn}$ *are the normalizing constants of* F_Y *given in Theorem 16,* γ_{EM} *is the Euler-Mascheroni constant,*

$$\gamma \triangleq \int_1^\infty \left(\frac{1}{\lfloor x \rfloor} - \frac{1}{x}\right) dx \approx 0.577, \tag{3.9}$$

and $\Gamma(\cdot)$ *is the Gamma function,*

$$\Gamma(t) \triangleq \int_0^\infty x^{t-1}e^{-x}\,dx. \tag{3.10}$$

To decide whether to kill or to keep the original copy of the straggling task, we are essentially comparing the additional time needed for the original time to finish

and the completion time for a new copy. This depends on $F_X(\cdot)$ and in Lemma 11 we identify distributions F_X for which killing the original task is better than keeping the original task for any r and p, and vice versa.

Lemma 11 (Whether to kill or keep original task). *If X satisfies the new-longer-than-used property, that is,*

$$\Pr(X > x + a | X > a) \leq \mathbb{P}[X > x] \ \text{for all } x, a \geq 0, \tag{3.11}$$

$\pi_{\text{keep}}(p, r)$ gives lower latency $\mathbb{E}[T]$ than $\pi_{\text{kill}}(p, r)$ for any r and p. Similarly, if X is new-shorter-than-used with the reverse inequality in (3.11), $\pi_{\text{kill}}(p, r)$ leads to lower $\mathbb{E}[T]$. In addition, for the asymptotic case $n \to \infty$, we can apply Theorem 5 and relax the condition in (3.11) to

$$\Pr(X > x + F_X^{-1}(1 - p) | X > F_X^{-1}(1 - p)) \leq \mathbb{P}[X > x] \ \text{for all } x \geq 0. \tag{3.12}$$

To prove this result we observe that when X is new-longer-than-used, Y (defined in (3.5)) with $\pi_{\text{kill}}(p, r)$ stochastically dominates the case of keeping the original task. The detailed proof is presented in Appendix D.

Remark 6. *The notions 'new-longer-than-used' and 'new-shorter-than-used' are closely related to log-concave and log-convex distributions defined Chapter 2. All log-concave (log-convex) distributions are new-longer-than-used (new-shorter-than-used), but the converse is not true. Thus, if X is log-concave (log-convex) then $\pi_{\text{kill}}(p, r)$ gives lower latency than $\pi_{\text{keep}}(p, r)$ for any r and p.*

Thus, the hyper-exponential distribution which is log-convex is also new-shorter-than-used, and the shifted exponential distribution which is log-concave is also new-longer-than-used. In [61] 'new-longer-than-used' is called 'new-better-than-used' in the context of residual life-time. Note that these notions are related to light and heavy-tailed distributions but neither implies the other. For example, the Pareto distribution is a heavy-tailed distribution, but is neither new-shorter-than-used nor new-longer-than-used.

3.3.2 Examples of the Effect of Tail Distribution

By Theorem 5 the scaling of $\mathbb{E}\left[Y_{pn:pn}\right]$ with n depends on whether the task service time X is heavy, light or exponential tailed. And by Lemma 11, we know that the choice between π_{kill} or π_{keep} is governed by whether X is new-longer-than-used or new-shorter-than used.

In this section we consider two execution time distributions: Shifted exponential and Pareto, for which the latency-cost trade-off in Theorem 5 is tractable. The shifted exponential distribution has an exponential tail, while Pareto distribution has a heavy tail. The shifted exponential distribution is new-longer-than-used. The tail $\Pr(X > x)$ of the Pareto distributionis new-shorter-than-used for $x \geq x_m$, but not otherwise. Thus, Pareto is neither new-longer-than-used nor new-shorter-than-used. For these two distributions we demonstrate how the tail, and the residual life of the service time distribution affect the choice of the best single-fork policy.

Shifted exponential execution time

Consider that the task execution time distribution F_X is a *shifted exponential distribution* $\mathsf{ShiftedExp}\left(\Delta, \mu\right)$. Its tail distribution function is given by

$$\Pr(X > x) = \begin{cases} e^{-\mu(x-\Delta)} & \text{for } x \geq \Delta, \\ 1 & \text{otherwise.} \end{cases} \tag{3.13}$$

The shifted exponential distribution has an exponentially decaying tail. It is lower bounded by a constant Δ, aiming to capture the delay due to machine start-up or task initialization. Due to this constant Δ, the shifted exponential distribution is new-longer-than-used. The special case $\Delta = 0$ corresponds to the pure exponential distribution, which is both new-longer-than-used and new-shorter-than-used, which implies that it is memoryless.

Theorem 6. *For a computing job with n tasks, if the execution time distribution of*

71
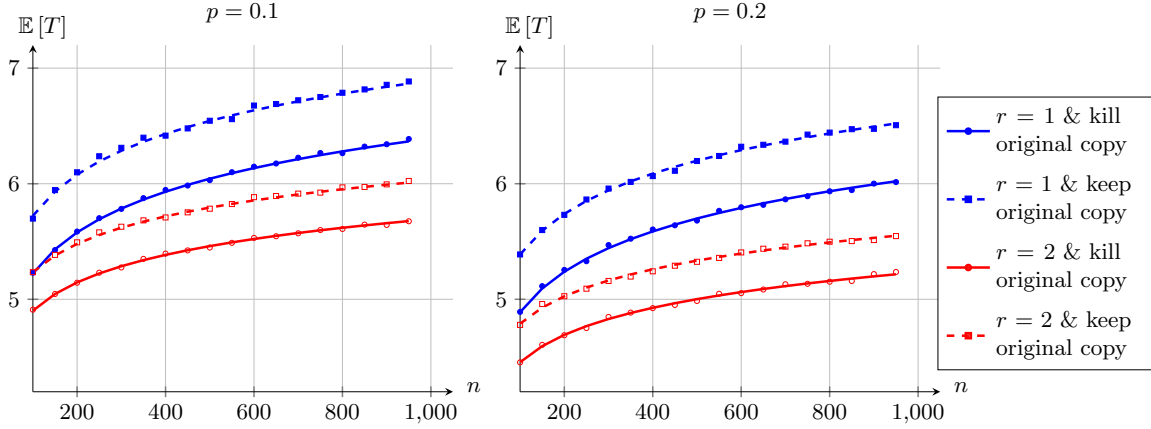
Figure 3-3: Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the shifted exponential distribution ShiftedExp $(1, 1)$.

each task is ShiftedExp (Δ, μ), then as $n \to \infty$, the latency and cost metrics are

$$\mathbb{E}[T] = \begin{cases} \frac{2r+1}{r+1}\Delta + \frac{1}{(r+1)\mu} \left(\ln n - r \ln p + \gamma_{\mathrm{EM}}\right) & \text{for } \pi_{\mathrm{keep}}(p, r) \\ 2\Delta + \frac{1}{(r+1)\mu} \left(\ln n - r \ln p + \gamma_{\mathrm{EM}}\right) & \text{for } \pi_{\mathrm{kill}}(p, r) \end{cases}, \qquad (3.14)$$

$$\mathbb{E}[C] = \begin{cases} \Delta + \frac{1}{\mu} + p \left[\Delta + r\frac{(1-e^{-\mu\Delta})}{\mu}\right] & \text{for } \pi_{\mathrm{keep}}(p, r) \\ \Delta + \frac{1}{\mu} + p(r+2)\Delta & \text{for } \pi_{\mathrm{kill}}(p, r) \end{cases}, \qquad (3.15)$$

where γ_{EM} is the Euler-Mascheroni constant defined in (3.9).

Fig. 3-3 compares the latency obtained from Monte-Carlo simulation and analytical calculations for the shifted exponential distribution, indicating that the latency obtained from analytical calculation is very close to the simulated performance for $n \geq 100$, especially for the case with killing the original task.

We can draw the following observations from Theorem 6. Given r and whether we kill or keep the original task, replicating earlier (larger p) gives an $\Theta(\ln p)$ decrease in latency, and a linear increase the cost. This is also illustrated in Figures 3-4a and 3-4b for execution time distribution ShiftedExp $(1, 1)$ and $n = 400$. Fig. 3-4c illustrates the latency-cost trade-off.

For the special case of $\Delta = 0$ by Theorem 6, the cost $\mathbb{E}[C] = 1/\mu$, which is

independent of p and r. But latency always reduces with r and p. This suggests that we can achieve arbitrarily low latency without any increase in cost. However, in practice the minimum time to complete a task is non-zero. Thus, pure exponential task service time is not a useful model for the purpose of analyzing task replication.
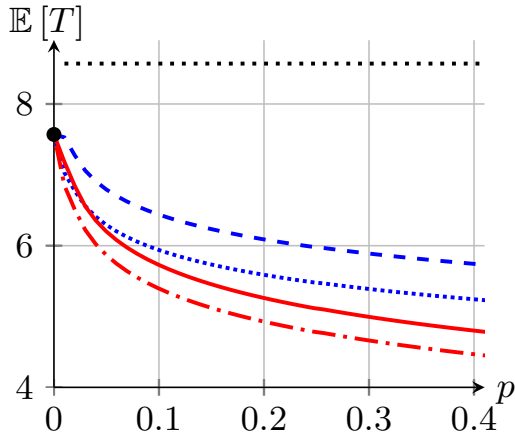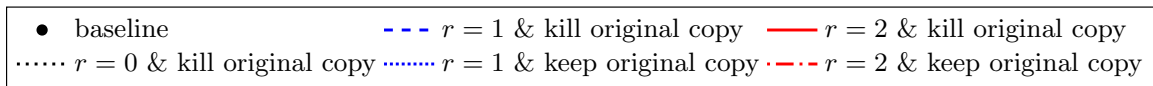
(a) Expected latency $\mathbb{E}[T]$
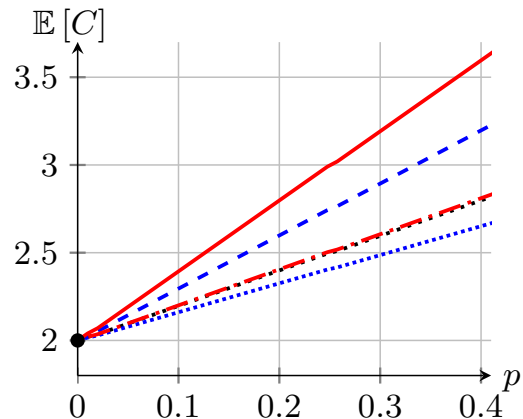
(b) Expected cost $\mathbb{E}[C]$
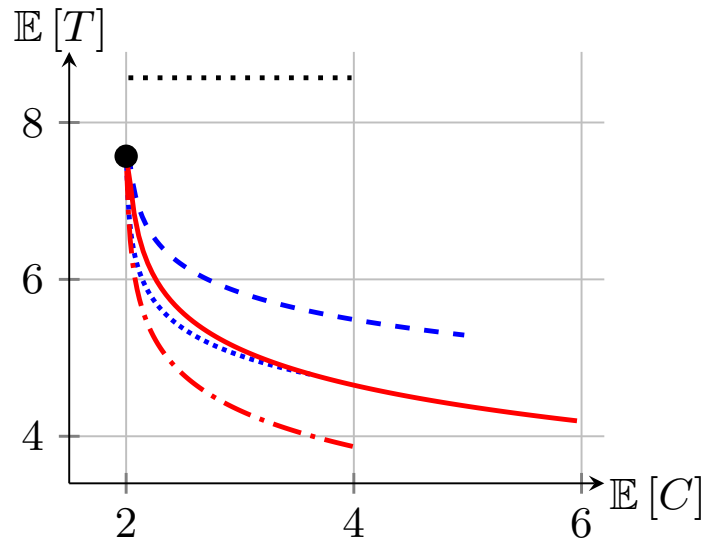
(c) Trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$

Figure 3-4: Characterization for $\mathsf{ShiftedExp}(1,1)$ and $n = 400$, by varying p in the range of $[0.05, 0.95]$.
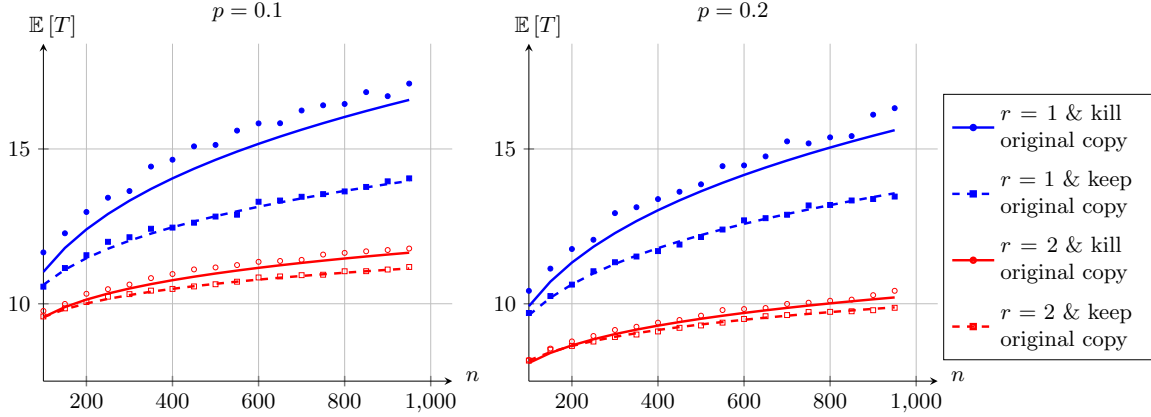
Figure 3-5: Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the Pareto distribution $\mathsf{Pareto}(2, 2)$.

Pareto execution time

The tail distribution function of the Pareto distribution $\mathsf{Pareto}(\alpha, x_m)$ is

$$
\Pr(X > x) \triangleq
\begin{cases}
\left(\frac{x_m}{x}\right)^{\alpha} & x \geq x_m, \\
1 & \text{otherwise}
\end{cases}
\tag{3.16}
$$

It has been observed to fit task execution time distributions in data centers [7, 51]. The Pareto distribution has a heavy-tail that decays polynomially. Pareto is neither new-longer-than-used nor new-shorter-than-used.

Theorem 7. *For a computing job with n tasks, if the execution time distribution of each task is $\mathsf{Pareto}(\alpha, x_m)$, then as $n \to \infty$, the latency and cost metrics are*

$$
\mathbb{E}[T] = x_m p^{-1/\alpha} + \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right)\tilde{a}_{pn},
\tag{3.17}
$$

$$
\mathbb{E}[C] = x_m \frac{\alpha}{\alpha - 1} - x_m \frac{p^{1-1/\alpha}}{\alpha - 1} + (r+1)p\mathbb{E}[Y].
\tag{3.18}
$$

The values of $\tilde{a}_{pn}$ and $\mathbb{E}[Y]$ depend on the whether we choose to keep or kill the original task, and are given as follows.

74

Case 1: Killing the original task

$$\tilde{a}_{pn} = (pn)^{\frac{1}{(r+1)\alpha}} x_m, \tag{3.19}$$

$$\mathbb{E}[Y] = \frac{(r+1)\alpha}{(r+1)\alpha - 1} x_m. \tag{3.20}$$

Case 2: Keeping the original task

The term $\tilde{a}_{pn}$ is the solution to

$$n^{1/\alpha} x_m^{r+1} = x_m p^{-1/\alpha} \tilde{a}_{pn}^r + \tilde{a}_{pn}^{r+1}. \tag{3.21}$$

and $\mathbb{E}[Y]$ is evaluated numerically as discussed in the proof.

Similar to Fig. 3-3, Fig. 3-5 compares the latency obtained from simulation and analytical calculations for the Pareto distribution, which again demonstrates the effectiveness of the asymptotic theory. Based on Theorem 7, we can derive how $\mathbb{E}[T]$ scales with n in the following corollary.

Corollary 6. *For a computing job with n tasks, if the execution time distribution of each task is $\mathsf{Pareto}\,(\alpha, x_m)$, then the expected latency satisfies*

$$\mathbb{E}[T] = \Theta\left(n^{1/(\alpha(r+1))}\right).$$

Corollary 6 indicates that

- the heavier the tail (smaller α), the faster $\mathbb{E}[T]$ grows with n;

- the latency reduction due to redundancy r diminishes as r increases due to the $1/(r+1)$ factor in the exponent.

In Figures 3-6a and 3-6b we plot the expected latency and cost as p varies, for different values of r. The black dot is the baseline case $(p = 0)$, where no replication is used and we simply wait for the original copies of all n tasks to finish. Note that $r = 0$ and keeping the original copy is also equivalent to the baseline case, and thus not plotted in the figures. The diminishing return of increasing r in terms of latency

reduction is clearly demonstrated. In addition, we observe that a small amount of replication (small p and r) can reduce latency significantly in comparison with the baseline case. But as p increases further, the latency may increase (as observed for $r = 0$) because of the second term in (3.6).
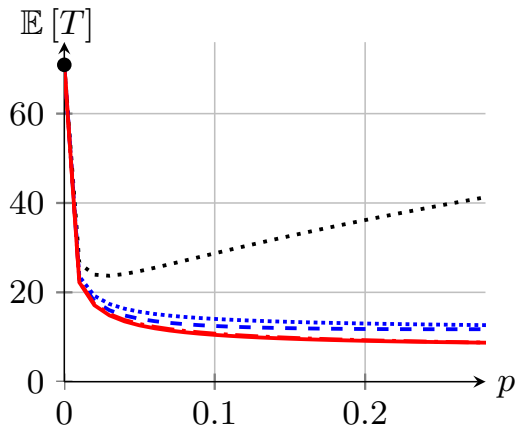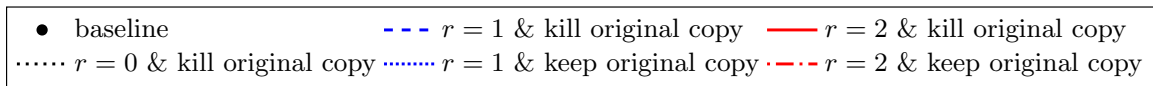
(a) Expected latency $\mathbb{E}[T]$
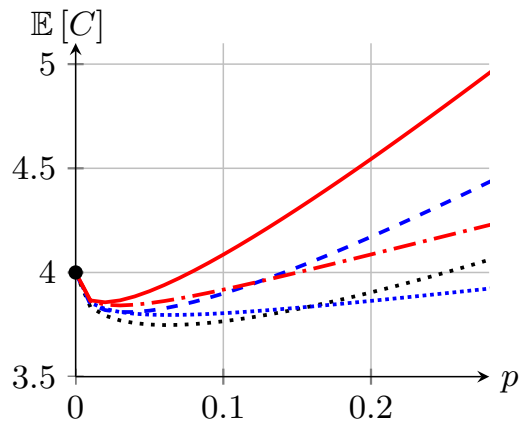
(b) Expected cost $\mathbb{E}[C]$
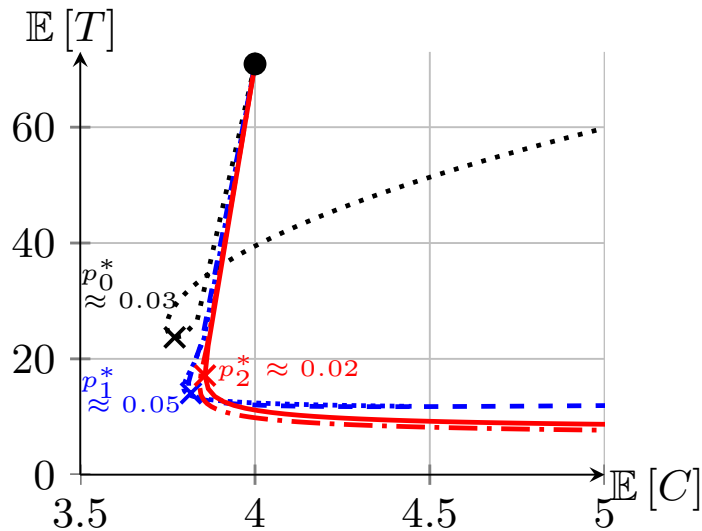
(c) Trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$

Figure 3-6: Characterization for $\mathsf{Pareto}\,(2, 2)$ and $n = 400$, by varying p in the range of $[0.05, 0.95]$.

Intuition suggests that replicating earlier (larger p) and more (higher r) will increase the cost $\mathbb{E}[C]$. But Figures 3-6a and 3-6b show that this is not necessarily true.

Since we kill replicas of task when one of its replicas finish, there could in fact be a saving in the computing cost. However this benefit diminishes as p and r increase above a certain threshold.

Fig. 3-6c shows the latency versus the computing cost for different values of r, with p varying along each curve. Depending upon the latency requirement and limit on the cost, one can choose an appropriate operating point on this trade-off. This plot again demonstrates the non-intuitive phenomenon that it is possible to reduce latency (from 70 to about 15 for $r = 1$ and $r = 2$ cases) and computing cost simultaneously.

3.4   Empirical execution time distributions

In certain practical systems it may be difficult to fit the empirical behavior of the task execution time to a well-characterized distribution, thus making the latency-cost analysis using the framework presented in Section 3.3 difficult. In this section we propose an algorithm to estimate the latency and cost from the empirical distribution of task execution time. Applying our algorithm to the Google Cluster Trace data [62], we show that it is possible to improve upon the performance of the default replication policy in MapReduce-style frameworks.

3.4.1   Latency and Cost Estimation

To estimate the latency and cost from empirical execution time samples, we apply the bootstrapping method [63] that uses the empirical distribution as an approximation of the true distribution.

Since the performance metrics $\mathbb{E}[T]$ and $\mathbb{E}[C]$ are functions of both X and Y, we need samples for both X and Y. Drawing samples of Y is more involved, especially for the case of killing the original task. To handle this, we leverage our analysis in Lemma 9 and compute $\hat{F}_Y(\cdot)$ based on (3.5), thus avoiding excessive sampling. We present the algorithm for performance characterization in Algorithm 3.1.

By Theorem 14, the standard deviation of the error in estimating $\mathbb{E}[C]$ and $\tilde{T}_1$, first term in $\mathbb{E}[T]$, converges to zero as $O(1/\sqrt{mn})$, where m is the number of times the

Algorithm 3.1 Latency and cost estimation
___

INPUT: $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, n task execution duration samples (no replication, no original task killing)

Compute the empirical c.d.f. $\hat{F}_X(x)$ from $\mathbf{x}$

Compute c.d.f. $\hat{F}_Y(y)$ using (3.5)

for $i = 1, 2, \ldots m$ **do**

    Draw n samples $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n]$ from $\hat{F}_X$

    Sort $\hat{\mathbf{x}}$ in ascending order: $[\hat{x}_{(1)}, \hat{x}_{(2)}, \ldots, \hat{x}_{(n)}]$

    $k \leftarrow n(1 - p)$; $k' \leftarrow np$

    $\tilde{T}_1^{(i)} \leftarrow \hat{x}_{(k)}$ (the k-th smallest sample in $\hat{\mathbf{x}}$)

    $\tilde{C}_1^{(i)} \leftarrow \sum_{j=1}^{k} \hat{x}_{(j)}$

    Draw k' samples $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{k'}]$ from $\hat{F}_Y$

    $\tilde{T}_2^{(i)} \leftarrow \max_{1 \leq j \leq k'} \hat{y}_j$

    $Y_{sum}^{(i)} \leftarrow \sum_{j=1}^{k'} \hat{y}_j$

    $\tilde{C}_2^{(i)} \leftarrow pn\tilde{T}_1^{(i)} + (r + 1)Y_{sum}^{(i)}$

    $\tilde{T}^{(i)} \leftarrow \tilde{T}_1^{(i)} + \tilde{T}_2^{(i)}$

    $\tilde{C}^{(i)} \leftarrow \frac{1}{n} \left[\tilde{C}_1^{(i)} + \tilde{C}_2^{(i)} \right]$

end for

$\tilde{T} \leftarrow$ mean of $\tilde{T}^{(i)}$ for $i = 1, 2, \ldots m$

$\tilde{C} \leftarrow$ mean of $\tilde{C}^{(i)}$ for $i = 1, 2, \ldots m$

OUTPUT: $[\tilde{T}, \tilde{C}]$
___

sampling procedure is repeated. And generally $\tilde{T}_2$, the maximum order statistic term in $\mathbb{E}[T]$, converges to zero as $O(1/\sqrt{m})$. Thus, the estimation of $\tilde{C}$ is more robust than that of $\tilde{T}$. Nonetheless, with large enough m, we can make the estimation errors of both metrics small enough.

3.4.2 Demonstration using Google Cluster Trace

The Google Cluster Trace data [62] gives timestamps of events such as SCHEDULE, EVICT, FINISH, FAIL, KILL etc. for each of the tasks of computing jobs that are run on Google's cluster machines. In this section we apply Algorithm 3.1 to two jobs in the Google Cluster Trace, and observe performance trade-offs in task execution distribution based on real-world data.

In our demonstration we only consider tasks with SCHEDULE and FINISH times, as we would like to obtain samples that represent a normal execution (not killed or

(a) Job 1: Google cluster Job (b) Job 2: Google cluster Job (c) Job 3: Tail-shortened Job
   6252284914 (1026 tasks)   6252315810 (488 tasks)   6252315810 (485 tasks)
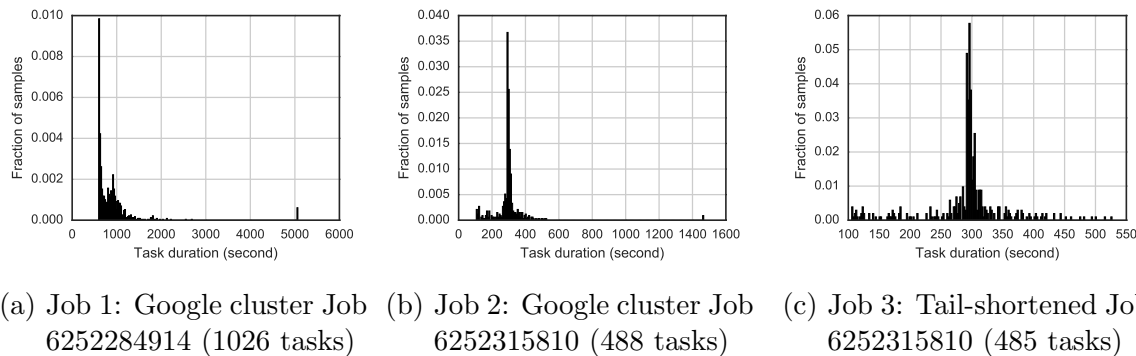
Figure 3-7: Normalized histogram of the task execution times

evicted). In a few rare cases, a task is associated with multiple SCHEDULE and FINISH events due to duplicate execution. For these we choose to keep the first occurrences in each event category.

We choose two jobs (Job ID 6252284914 and 6252315810) with different number of tasks. For each task in a job, we obtain the task execution time by calculating the time difference between SCHEDULE and FINISH. The normalized histograms of the task execution times of the two jobs are shown in Fig. 3-7a and Fig. 3-7b respectively. Both the distributions indicate heavy tail behavior of the task execution time, where the heavy tail is more pronounced in Fig. 3-7a. In addition, to show the importance of stragglers, we modify the trace for Job 6252315810 by removing the 3 samples with execution time longer than 1400 seconds, leading to the execution time distribution shown in Fig. 3-7c.

We then apply these execution time samples as inputs to Algorithm 3.1 with $m = 1000$. By varying the value of r ($r \in \{1, 2, 3\}$) and p ($0 \leq p \leq 0.5$), we plot the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-offs for all three jobs in Figures 3-8 to 3-10.

For the two Google cluster jobs (Job 1 and 2), we observe that a small amount of replication (small p) reduces both $\mathbb{E}[T]$ and $\mathbb{E}[C]$ significantly, demonstrating the effectiveness of replication for real-world execution time distributions. In both cases, it is better to replicate while *keeping* the original task, because at the "fork" point, the additional time needed for the original copy to finish is more likely to be shorter than the execution time of a new copy. We also observe that for the Job 2 (Job ID

(a) Original copy kept π_{keep} (b) Original copy killed π_{kill}
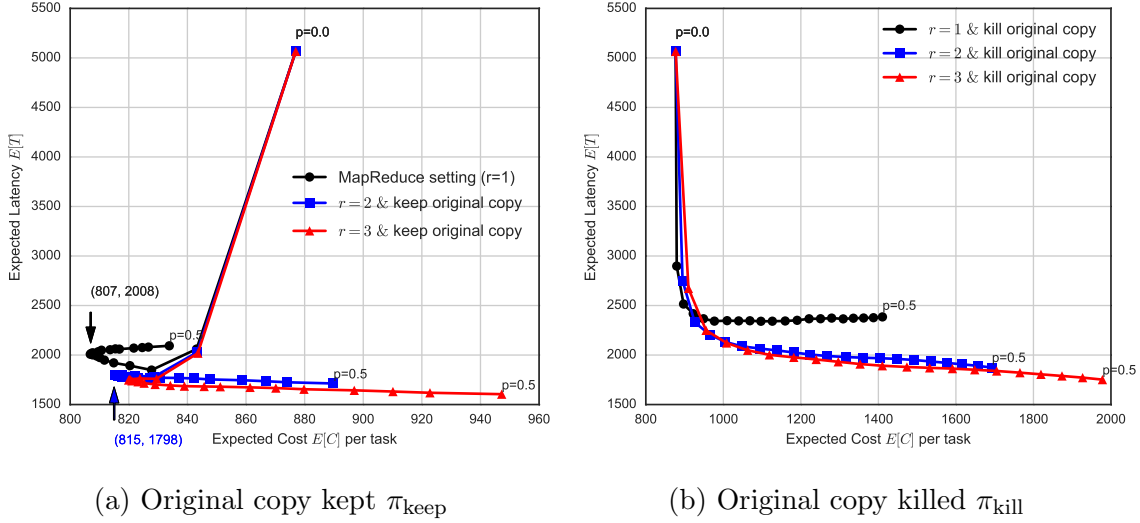
Figure 3-8: The $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off for Job 1 (ID 6252284914) with 1026 tasks. Each pair of adjacent dots corresponds to change in p by 0.01.

6252315810), too much redundancy may hurt, because at some point increasing p actually leads to increases in both $\mathbb{E}[T]$ and $\mathbb{E}[C]$. However, this phenomenon does not exist for Job 1 (Job ID 6252284914) when $r = 2$ or $r = 3$. We conjecture this is due to the tail in Fig. 3-7a is heavier than that in Fig. 3-7b.

We recall that the back-up tasks option in MapReduce uses $r = 1$ and keeps the original task, and show that for certain jobs it may be more desirable to improve the performance trade-off by using more replicas, such as in Job 1, where a higher r could lead to lower latency $\mathbb{E}[T]$ with a slightly higher cost $\mathbb{E}[C]$. For example, $\pi_{\text{keep}}(p, r = 1)$ achieves $(\mathbb{E}[C], \mathbb{E}[T]) = (807, 2008)$, while $\pi_{\text{keep}}(p, r = 2)$ achieves $(\mathbb{E}[C], \mathbb{E}[T]) = (815, 1798)$. For Job 2, the trade-off improvement via using a higher r is less significant, as Fig. 3-9 indicates. Finally, for both jobs we observe that increasing r has a diminishing effect on the reduction of $\mathbb{E}[T]$.

For the tail-shortened trace histogram in Fig. 3-7c, killing the original copy increases the latency, because it is too "impatient"—the original copy is likely to finish before a new copy of the task. On the other hand, if we keep the original copy, adding a small amount of redundancy again can reduce latency and computing cost simultaneously, as shown in Fig. 3-10a. Lastly, Fig. 3-10 indicates that killing and replicating can lead to worse performance trade-off, so one needs to apply replication
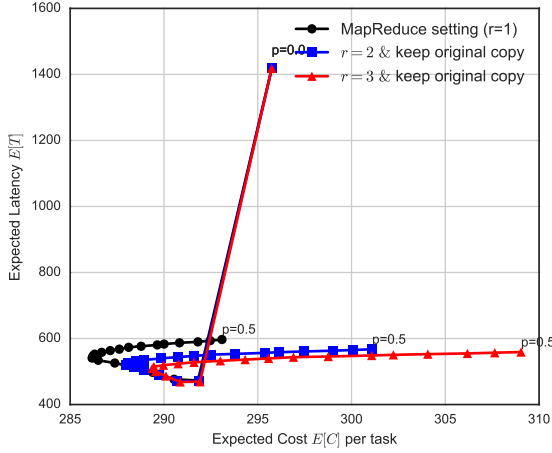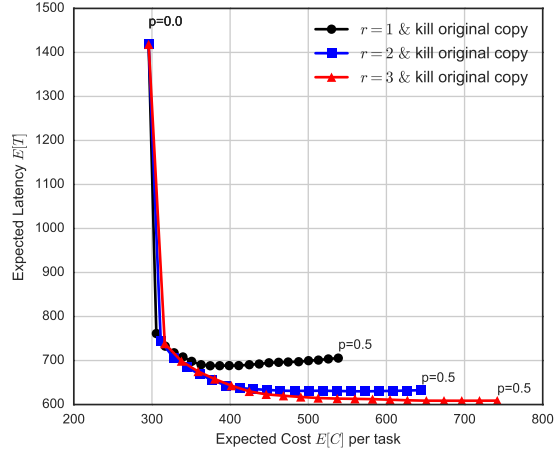
(a) Original copy kept π_{keep} (b) Original copy killed π_{kill}

Figure 3-9: The $\mathbb{E}\left[T\right]$-$\mathbb{E}\left[C\right]$ trade-off for Job 2 (ID 6252315810) with 488 tasks. Each pair of adjacent dots corresponds to change in p by 0.01.
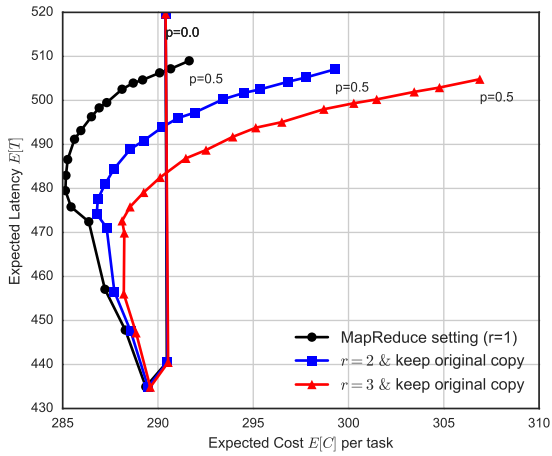
(a) Trade-off with original copy kept π_{keep} (b) Trade-off with original copy killed π_{kill}
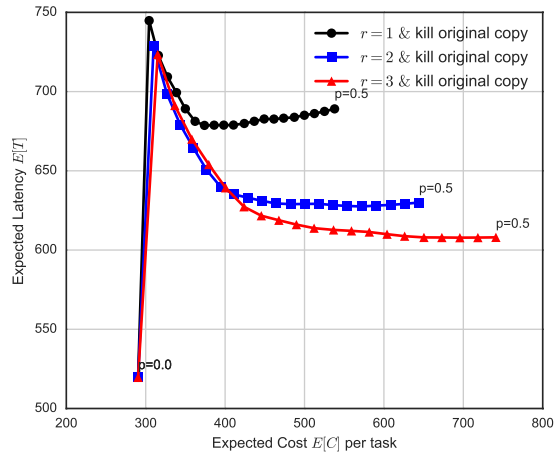
Figure 3-10: The $\mathbb{E}\left[T\right]$-$\mathbb{E}\left[C\right]$ trade-off for the Job 3 (tail-shortened Job 2) with 485 tasks. Each pair of adjacent dots corresponds to change in p by 0.01.

with care.

We draw the following observations from the above examples:

- Replication often reduces $\mathbb{E}\left[T\right]$ and $\mathbb{E}\left[C\right]$ simultaneously.

- A small amount of replication usually suffices and too much replication leads to a sharp increase in $\mathbb{E}\left[C\right]$.

81

- If F_X is new-longer-than-used, we should keep the original task.

3.4.3 Scheduling policy selection

With the trade-off between latency $\mathbb{E}[T]$ and computing cost $\mathbb{E}[C]$ provided in Algorithm 3.1, a user can formulate an optimization problem to choose the best scheduling policy based on one's sensitivity to latency and computing cost. In addition, one can incorporate additional constraints, such as $r_{\max}$, the maximum number of copies to replicate r, due to the communication overhead of issuing and canceling tasks.

For example, a latency-sensitive user may choose to define the optimal scheduling policy via the following constrained optimization problem:

$$\text{minimize} \quad \mathbb{E}[T(\pi)], \tag{3.22}$$

$$\text{subject to} \quad \mathbb{E}[C(\pi)] \leq \mathbb{E}[C(\pi_0)], \tag{3.23}$$

$$r \leq r_{\max},$$

where π_0 is the baseline scheduling policy without replication and $r_{\max}$ the maximum allowed number of copies for a task. On the other hand, a cost-sensitive user may choose to define the optimal scheduling policy via the following optimization problem:

$$\text{minimize} \quad \mathbb{E}[T(\pi)] + \lambda n \mathbb{E}[C(\pi)], \tag{3.24}$$

$$\text{subject to} \quad r \leq r_{\max},$$

where λ indicates the relative importance of computing cost, because $\mathbb{E}[C]$ is approximately proportional to the cost of cloud computing instances. While it is difficult to determine closed-form optimal solutions to (3.22) and (3.24), we observe that constrained optimization methods such as the Constrained Optimization BY Linear Approximation (COBYLA) method [64] are effective in searching for the optimal solution due to the low dimensionality of the search space. In Table 3.1, we present the scheduling policies obtained via these two different optimization formulations.

	Baseline		Latency-sensitive					Cost-sensitive with $\lambda = 0.1$				
Job	$\mathbb{E}[T]$	$\mathbb{E}[C]$	p^*	r^*	keep/kill	$\mathbb{E}[T]$	$\mathbb{E}[C]$	p^*	r^*	keep/kill	$\mathbb{E}[T]$	$\mathbb{E}[C]$
Job 1	5068	882	0.343	4	keep	1676	881	0.234	1	keep	2213	806
Job 2	1418	296	0.038	4	keep	463	291	0.181	4	keep	542	286
Job 3	520	290	0.044	4	keep	432	290	0.173	1	keep	480	285

Table 3.1: Scheduling policy obtained via latency-sensitive optimization in (3.22) and cost-sensitive optimization in (3.24).

3.5 Concluding remarks

3.5.1 Main Implications

Replication of the slowest tasks of a computing job (straggling tasks) has been observed to be highly effective in frameworks such as MapReduce to speed-up job completion. In this chapter we provide a theoretical framework to understand the effect of straggler replication on the job completion latency, and the additional computing time spent on running the replicas.

Using tools from extreme value theory, we characterize the latency-cost trade-off in terms of the task execution time distribution F_X. We focus on three parameters of a replication strategy: 1) fraction of slowest tasks of a job that are considered as stragglers, 2) number of replicas of each straggling task, and 3) whether we should kill the original copy of the task and relaunch it on a new machine.

This analysis gives the insight that the scaling of job completion latency with the number of tasks depends on the whether the tail of F_X is heavy, light, or exponential. And, the choice of whether we should kill or keep the original task depends on whether F_X is new-longer-than-used or new-shorter-than-used. For example, if F_X is the shifted exponential distribution, which is new-longer-than-used, then it is better to keep the original task running. Our latency-cost analysis helps identify regimes where replicating a small fraction of stragglers can drastically reduce latency and reduce computing cost as well.

We also propose a bootstrapping-based algorithm to estimate the latency and cost from empirical traces of execution time. Using this algorithm on the Google Cluster Trace data, we demonstrate that careful choice of the replication strategy can improve

the latency-cost trade-off as compared to the default option in MapReduce.

3.5.2   Future Directions

Generalizations of this straggler replication model include considering heterogeneous servers, dependencies between tasks (some tasks need to complete in order to begin others), and taking into account queueing delay of tasks as considered in Chapter 2 for the single task case. Another direction is to analyze approximate computing, where we need only a subset of the tasks of a job to complete, a relevant model for information retrieval and machine learning jobs. This idea is developed in the context of coded distributed storage in Part II. Finally, we aim to develop an algorithm that learns the task execution time distribution F_X online, and uses it to decide when and how many replicas to launch. This has an exploration-exploitation trade-off, similar to the multi-arm bandit problems studied in reinforcement learning [65]. Preliminary insights on this problem are described in the Chapter 4.

Chapter 4

Future Directions

In Chapter 2 we analyze the effect of task replication on queues in cloud systems. Chapter 3 generalizes the model to consider replication of the stragglers in a job with many parallel tasks. In recent years there is a flurry of works building on this idea of using redundancy to reduce delay in cloud systems. In Section 4.1 we describe some of these generalizations of the model considered in Chapter 2. Next we describe preliminary insights on two generalizations of particular interest to us: heterogeneous servers (Section 4.2), and unknown service distributions (Section 4.3). Broader research directions of beyond the realm of cloud infrastructure, such as crowdsourcing are described in Chapter 11.

4.1   Recent Model Generalizations

4.1.1   Exact Analysis of T

In Chapter 2 and Chapter 3 the latency metric is $\mathbb{E}[T]$, the expected waiting plus service time. Although the expected value is a good indicator of the average behavior, often system designers are interested in the tail, for example the 99th percentile latency. For a majority of queueing problems, determining the distribution of response time T requires the assumption of exponential service time. Considering the expected latency $\mathbb{E}[T]$ in Chapter 2 allowed us to look at arbitrary service time F_X and dis-

cover that log-concavity is the key property governing the choice of the replication strategy.

The exact analysis of T is tractable in some regimes. For the full replication $(r = n)$ case, we can determine the distribution of T using Lemma 2 and Lemma 3. For example, when $r = n$ and cancel-on-finish, T is equivalent in distribution to the latency of an $M/G/1$ queue with service time $X_{1:n}$. Then transform analysis [54, Chapter 25] can be used to determine the distribution of T. The Laplace-Stieltjes transform $T(s)$ of the probability density function of $f_T(t)$ of T is given by,

$$T(s) = \frac{sX_{1:n}(s)\left(1 - \frac{\lambda}{\mathbb{E}[X_{1:n}]}\right)}{s - \lambda(1 - X_{1:n}(s))}, \tag{4.1}$$

where $X_{1:n}(s)$ is the Laplace-Stieltjes transform of the service time distribution $f_{X_{1:n}}(x)$.

When $r < n$, the analysis of T is hard in general. An exact analysis of T when each task is replicated at $r < n$ servers chosen uniformly random is presented in [66], albeit for exponential service time. The analysis is highly non-trivial problem even with the exponential service time assumption and requires using a numerical package to solve a set of differential equations.

To go to general service times [49, 67] uses a very interesting approach of determining latency-optimality gaps. For new-better-than-used (NBU) and new-worse-than-used (NWU) service distributions they propose replication strategies and show that their latency T is within a provably small gap from the latency of the optimal scheduling policy. The proof approach uses a series of elegant stochastic ordering arguments. However, it is hard to generalize this arbitrary service distributions beyond the NBU and NWU distributions.

4.1.2 Cancellation Overheads

So far we assumed that cancellation of redundant tasks is instantaneous. However in practice there may be a significant delay in cancelling tasks, which can diminish the benefits of replication.

We now present preliminary insights in the low traffic regime where all servers are idle. Consider that a task is replicated at r servers. As soon as one replica finishes, that server contacts other replicas to cancel them. This cancellation process takes Δ seconds of time at each of the r servers. Then the latency and cost are given by

$$\mathbb{E}[T] = \min(X_1, X_2, \ldots, X_r) + \Delta \tag{4.2}$$

$$\mathbb{E}[C] = r(\min(X_1, X_2, \ldots, X_r) + \Delta) \tag{4.3}$$

From (4.2) and (4.3), we observe that the latency and cost with cancellation delay is equivalent to that for instantaneous cancellation with service time $X' = X + \Delta$. Adding Δ makes the effective service time X' 'more log-concave'. Thus less or no replication would give a better latency-cost trade-off.

Going beyond the low traffic regime and analyzing the effect of cancellation delays in presence of queueing delays is a hard problem. It has been systematically studied in recent work [68] for the two server case with exponential service time. In [68] the cancellation delay is also considered to be an exponential random variable. Building on this work and understanding how cancellation delays affect servers with an arbitrary given distribution F_X is an open problem.

4.1.3 Correlated Tasks

In our analysis we assume that the service time X is i.i.d. across tasks and servers. However, in practice the time taken to serve a task is usually proportional to its size. Thus when the same task is assigned to two servers, there would be correlation between their service times.

To account for this, we can replace X by $X' = d + X$, where d is the part of the service time proportional to the size of the task, and X is the random delay due to the server. More generally, d can be a random variable D. Analysis of the effect of D is presented in recent work [69]. In the context of storage systems we also studied correlated task service times in [21].

4.1.4 Data Locality

In both Chapter 2 and Chapter 3, we assume that all servers have the data required to execute each task that enters the systems. However very often this is not true in practice because servers may need to fetch the necessary data to run a task and thus take a longer time to finish it. Consider these effects of data locality is a future research direction.

Tasks scheduling policies taking into account these data locality constraints are proposed in [70], without replication of tasks. They consider a hierarchical structure of servers in a data center such that serving a task close to its data (for example within the same rack) results in faster service than at a server that is further away. Analyzing how replication strategies would be affected by these variations in service rates is open problem. A possible first step to approach it is to consider heterogeneous servers as described in Section 4.2.

The effect of data locality constraints on the task can also be considered via heterogeneous task classes, which are analyzed in [40]. Tasks belonging to a particular class can only be assigned to one or more servers that have the necessary data to serve data. Thus some classes of tasks can be replicated more than others. The paper [40] provides an understanding of whether the replication of one class adversely affects another class of tasks that are not replicated.

4.1.5 Coded and Approximate Computing

In Chapter 3 we consider a job with n parallel tasks such that all tasks need to complete to finish the job. Then the slowest tasks become a bottleneck and we need to apply straggler replication techniques to cut the tail latency.

In the context of storage systems described in Part II, we consider that a file is divided into k chunks and coded into n chunks. Then if we request all n chunks, any k are sufficient to recover the file. Thus, coding helps avoid the problem the stragglers. Coding can also be applied in the context of cloud computing, as explored in [71, 72] recently. In [72], the authors propose methods to 'code' in machine learning problems

such as matrix multiplication, and demonstrate significant latency reductions.

Even without coding over the tasks, there are applications where an approximate result is sufficient to complete a computing job. For example, when stochastic gradient descent is run in a distributed fashion [73], each task corresponds to a gradient update using a small batch of training samples. Then it is not necessary to complete all the tasks for the parameters to reach the desired accuracy. Developing a rigorous understanding of latency in such problems is an open future direction.

4.2 Heterogeneous servers

In this section we describe preliminary insights on scheduling tasks to a set of heterogeneous servers. Task replication on heterogeneous servers has been considered in [67] for NBU and NWU service time distributions. The paper provides the insight that it is optimal to replicate tasks on a set of servers that all have NWU distributions. Conversely, for a set of servers that all have NBU distributions, it is optimal to not replicate tasks. However, whether we can pair one server with a NWU distribution with another that has a NBU distribution is unknown. More generally, scheduling task replication on heterogeneous servers with arbitrary service distributions that may be neither NBU or NWU, is an open problem.

In the sequel we formulate the problem concretely and describe our initial steps towards the solution. We get the insight that servers can be 'paired' via replication such that together, their service rate is higher than the sum of the individual servers. Thus, counter-intuitively, task replication can in fact improve the efficiency of the system of servers.

4.2.1 Problem Formulation

Consider a system of K servers, illustrated in Fig. 4-1. The time taken by server i to finish a task assigned to it is a random variable X_i, independent and identically distributed across tasks assigned to that server. The service times are also independent across different servers. We consider that there are a large number of tasks n in
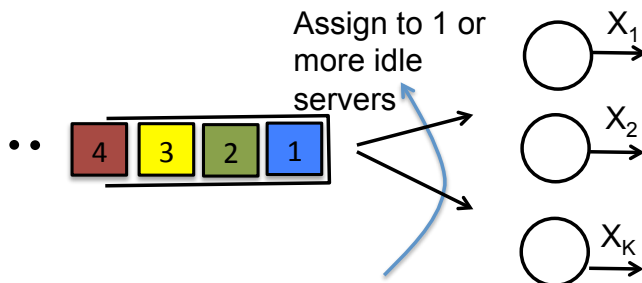
Figure 4-1: System of K servers with heterogeneous service times $X_1, X_2, \ldots X_K$, independent across the servers.

a centralized queue. The scheduler can assign each task to one or more idle servers.

The difference between this model and that of Chapter 2 is that for tractability of analysis, we consider a centralized queue of tasks rather than distributed queues at each server. Also, we assume that a large number of tasks are already in the queue instead of considering Poisson task arrivals. There is a centralized scheduler that assigns each task to one or more servers. As soon as one replica finishes, the others are cancelled immediately.

We evaluate the performance of a scheduling policy π in terms of the throughput $\bar{R}$ which is defined as follows.

Definition 10 (Throughput $\bar{R}$). *Let T_n be the time when the nth task departs from the system of K servers. Then the throughput is defined as*

$$\bar{R} = \lim_{n \to \infty} \frac{n}{T_n} \tag{4.4}$$

Our objective is to maximize $\bar{R}$ for $n \to \infty$. In the sequel we show that $\bar{R}$ can be expressed in terms of the expected computing time which is defined as follows.

Definition 11 (Expected Computing Time $\mathbb{E}[C]$). *The expected computing time $\mathbb{E}[C]$ is the total expected time spent by the servers per task.*

Claim 2. *The throughput-optimal policy has to be a work-conserving, such that it does not allow any server to be idle when one or more tasks are remaining in the central queue.*

90

Proof. Suppose a task has r replicas that start at times $0 = t_1 \leq t_2 \leq \ldots t_r$ at different servers. The time spent by the task in the system is $S = \min(X_1, X_2 + t_2, \ldots X_r + t_r)$. Using a non-work-conserving policy that idles one or more of these servers for a non-zero time will strictly increase S. This in turn will result in a strict increase T_n, and thus decrease $\bar{R}$. Even at the end of the execution when only the last task is left in the system, it is always sub-optimal to let any server idle. The throughput-optimal policy would replicate the last remaining task at all K servers. Thus, the throughput-optimal policy has to be work-conserving. □

Claim 3. *The scheduling policy that minimizes* $\mathbb{E}[C]$ *maximizes the throughput* $\bar{R}$.

Proof. By Claim 2, the total busy time of each server is exactly equal to T_n under the optimal policy. By law of large numbers and the definition of $\mathbb{E}[C]$,

$$\bar{R} = \lim_{n \to \infty} \frac{T_n}{n} = \frac{\mathbb{E}[C]}{K} \tag{4.5}$$

Thus, the scheduling policy that minimizes $\mathbb{E}[C]$ also maximizes $\bar{R}$. □

4.2.2 Two Server Motivating Example

To illustrate how replication can increase the effective service rate of a system of heterogeneous servers, let us consider the two server case. We compare two task scheduling policies: no replication and full replication illustrated in Fig. 4-2. This comparison provides insights into when it is better to pair the servers via replication.

(a) No replication (π_1): Assign each task to the first idle server

(b) Full replication (π_2): Assign each task to both servers and wait for one
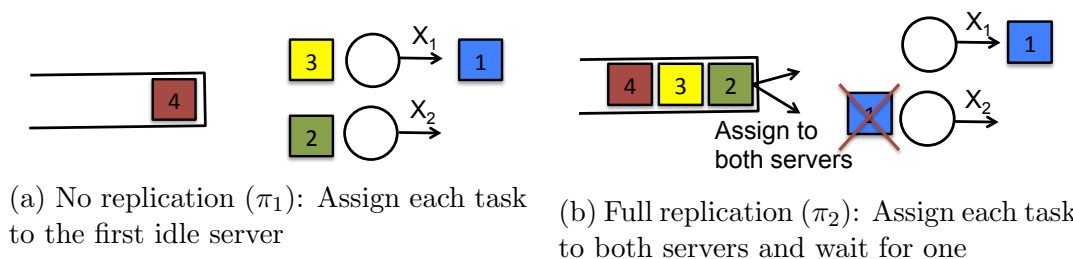
Figure 4-2: Illustration of the policies compared in Section 4.2.2

91

No Replication (π_1)

Each task is assigned to the first available idle server. For this policy, the expected computing cost is given as follows

$$\mathbb{E}\left[C(\pi_1)\right] = \Pr(\text{Assign to Server 1})\mathbb{E}\left[X_1\right] + \Pr(\text{Assign to Server 2})\mathbb{E}\left[X_2\right] \tag{4.6}$$

$$= \frac{\mathbb{E}\left[X_2\right]}{\mathbb{E}\left[X_1\right] + \mathbb{E}\left[X_2\right]}\mathbb{E}\left[X_1\right] + \frac{\mathbb{E}\left[X_1\right]}{\mathbb{E}\left[X_1\right] + \mathbb{E}\left[X_2\right]}\mathbb{E}\left[X_2\right] \tag{4.7}$$

$$= \frac{2}{1/\mathbb{E}\left[X_1\right] + 1/\mathbb{E}\left[X_2\right]} \tag{4.8}$$

Full Replication (π_2)

Each task is assigned to both servers, and as soon as one replica finishes, the other is canceled immediately.

$$\mathbb{E}\left[C(\pi_2)\right] = 2\mathbb{E}\left[\min(X_1, X_2)\right] \tag{4.9}$$

Lemma 12. *For a large number of tasks $n \to \infty$, the full replication policy π_2 results in higher throughput than the no replication policy π_1 if and only if*

$$\frac{1}{\mathbb{E}\left[\min(X_1, X_2)\right]} > \frac{1}{\mathbb{E}\left[X_1\right]} + \frac{1}{\mathbb{E}\left[X_2\right]} \tag{4.10}$$

Proof. By comparing $\mathbb{E}\left[C(\pi_1)\right]$ and $\mathbb{E}\left[C(\pi_2)\right]$ and applying Claim 3 we get Lemma 12.

$\square$

Using Lemma 12 we can compare the two policies for any arbitrary distributions of X_1 and X_2. If X_1 and X_2 are log-concave or log-convex then we get insights similar to Chapter 2. If X_1 and X_2 are exponential with rates μ_1 and μ_2 respectively then both sides of (4.10) will be equal. If X_2 is log-concave (for eg. shifted exponential), and X_1 is exponential then no replication gives higher throughput, as illustrated in Fig. 4-3. Similarly, if X_2 is log-convex then the full replication policy π_2 gives higher throughput as illustrated in Fig. 4-4.

Instead of these two extreme policies: no replication and full replication, we can
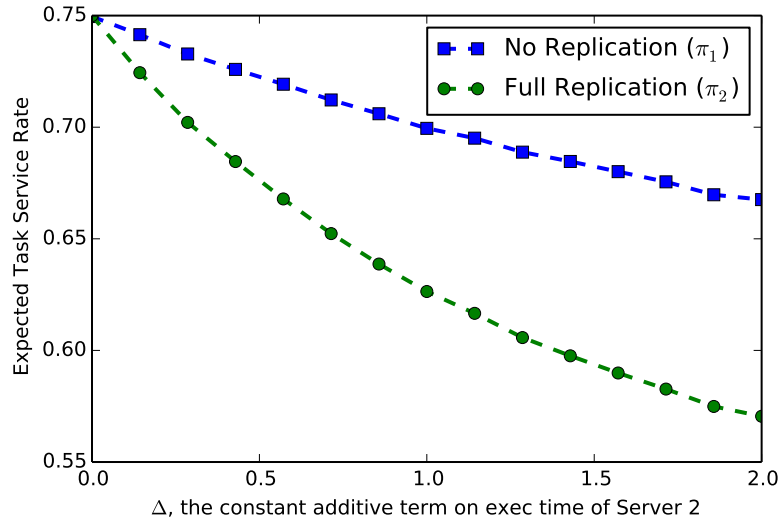
Figure 4-3: The no replication strategy gives higher throughput when $X_1 \sim \mathsf{Exp}(0.5)$ and $X_2 \sim \Delta + 0.25)$ (which is log-concave). The shift Δ increases along the $x-axis$.
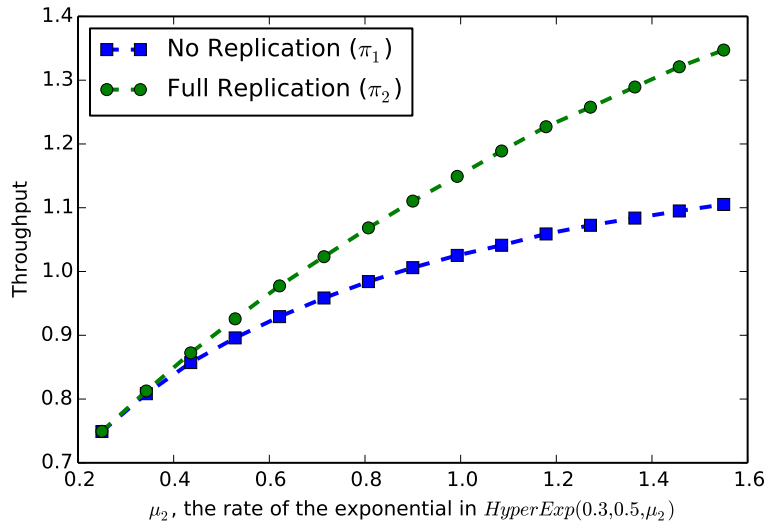
Figure 4-4: The full replication strategy gives higher throughput when $X_1 \sim \mathsf{Exp}(0.5)$ and $X_2 \sim \mathsf{HyperExp}(p = 0.3, \mu_1 = 0.5, \mu_2)$ (which is log-convex). The rate μ_2 increases along the $x-axis$.

also replicate tasks in a softer manner. For example, replicas could be added conditionally if a task does not finish in some given time. We propose the following scheduling policy that maximizes the instantaneous service rate of the system of servers.

Definition 12 (Max-rate policy π_R). *Without loss of generality, consider that server 1 finishes its assigned task and becomes idle. At this instant, suppose server 2 has spent t seconds on its current task. Let X_2' be its residual execution time, whose distribution is $\Pr(X_2' > x) = \Pr(X_2 > t + x)/\Pr(X_2 > t)$ for $x, t \geq 0$. Then if*

$$\frac{1}{\mathbb{E}\left[\min(X_1, X_2')\right]} > \frac{1}{\mathbb{E}\left[X_1\right]} + \frac{1}{\mathbb{E}\left[X_2'\right]} \tag{4.11}$$

add a replica of the task running on server 2. Otherwise, assign a new task from the centralized queue to server 1.

Generalizing this policy to $K > 2$ servers, and determine whether it is throughput-optimal (we conjecture that it is optimal) is open for future research.

4.3 Unknown Service Distribution

In this section we discuss another model generalization of particular interest: how to schedule tasks if the distribution F_X is unknown? Can we learn it online and adapt the scheduling policy? To the best of our knowledge, these questions have not been addressed in previous works.

4.3.1 Statistical Log-concavity Tests

In Chapter 2 we saw that the log-concavity of the task service time X is an important factor in determining whether replication helps. Also in Chapter 3, whether to kill or to keep the original copy of a straggling task depends on whether X is new-longer-than-used or new-shorter-than-used.

If X is not known beforehand, one way to choose the best replication strategy is via statistical tests for log-concavity. One among many such tests presented in [74] is the test for the increasing/decreasing hazard rate property. Given n empirical samples arranged in increasing order $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, we first define the normalized spacing

between order statistics defined as follows

$$D_i = (n - i + 1)(X_{(i)} - X_{(i-1)}), \text{ for } i = 1, 2, 3, n \tag{4.12}$$

If X is log-concave, then D_i's exhibit a downward trend with i. The Proschan and Pyke test [75] uses this property and declares X as log-concave if

$$P_n = \sum_{i<j} \mathbb{1}(D_i > D_j), \tag{4.13}$$

is greater than some critical value. The function $\mathbb{1}(E)$ is the indicator random variable which is 1 if event E occurs, and 0 otherwise. Conversely, by flipping the inequality sign in the indicator function we can test for log-convex distributions, which have the decreasing hazard rate property. Other tests for the increasing/decreasing hazard rate include the Epstein test [76, 77] and the Bickel and Doksum test [78]. Future work includes using such tests to determine the best replication strategy.

More broadly, strict log-concavity or log-convexity of X may not be necessary for the tests to be effective. Empirical samples may result in the erroneous classification of a 'near-log-convex' distribution as log-convex. However, a replication strategy that is optimal for log-convex distributions may still perform very well for such 'near-log-convex' distributions. We seek to develop a deeper understanding of how the error in the statistical test affects the performance of the replication strategy inferred from it.

4.3.2 Multi-arm bandits

Instead of estimating whether the distribution is log-concave or log-convex, we can try to directly adapt the optimal replication strategy based on empirical samples of service time. We now describe some initial thoughts on this approach.

Choosing the best arm r

Consider a simplified problem where we have n servers with i.i.d. task service times X, and assume that the arrival rate $\lambda \to 0$ such that the queues at the servers are

empty. Each task can be assigned to r out of the n servers. As soon as one replica finishes, the others are canceled immediately. Our objective is to find the optimal r^* that minimizes the cost $r\mathbb{E}[X_{1:r}]$.

If we launch r replicas for k_r tasks we get samples $X_{1:r}^{(1)}, X_{1:r}^{(2)}, \ldots, X_{1:r}^{(k_r)}$ and can use them to estimate $\mathbb{E}[X_{1:r}]$. Thus this can be posed as a multi-arm bandit problem with n arms. The difference from the classic multi-arm bandit problem is that the arms are correlated through the distribution X. Samples $X_{1:r}^{(1)}, X_{1:r}^{(2)}, \ldots X_{1:r}^{(k_r)}$ can be used to generate samples $X_{1:r'}^{(1)}, X_{1:r'}^{(2)}, \ldots$ from arm $r' \neq r$, albeit with more error than drawing samples from arm r' directly.

Future work includes developing an algorithm that converges to the optimal r^* with minimum number of total samples from the n arms. Accounting for the effect of queueing at the servers is a natural generalization of this model.

Heterogeneous servers

In Section 4.2 we considered the problem of scheduling task replication on heterogeneous servers with distributions $X_1, X_2, \ldots X_n$. If these distributions are unknown, we can learn them online using a multi-arm bandit approach. In this problem there will be an exploration-exploitation trade-off between finding faster servers by scheduling tasks to them, and exploiting the currently estimated fastest servers and their pairings.

Part II

Fast Content Download from

Coded Storage

Chapter 5

Background and Problem Formulation

5.1 Introduction

5.1.1 Motivation

Large-scale cloud storage systems such as Amazon Elastic Block Store (EBS) [79] and Google File System (GoogleFS) [80] have become the backbone of many applications, e.g., searching, e-commerce, and cluster computing. Content files stored on these systems may be simultaneously requested by multiple users. Content download time includes the time taken for a user to compete with the other users for access to the disks, and the time to acquire the data from the disks. Fast content download is important for delay-sensitive applications such as video streaming, VoIP, as well as collaborative tools like Dropbox [81] and Google Docs [82].

In large-scale distributed storage systems, disk failures are the norm and not an exception [7]. To protect the data from disk failures, cloud storage providers today simply replicate content throughout the storage network over multiple disks. In addition to fault tolerance, replication makes the content quickly accessible since multiple users requesting a content can be directed to different replicas. However, replication consumes a large amount of storage space. In data centers that process

massive data, using more storage space implies higher expenditure on electricity, maintenance and repair, as well as the cost of leasing physical space.

Coding, which was originally developed for reliable communication in presence of noise, offers a more efficient way to store data in distributed systems. The main idea behind coding is to add redundancy so that a content, stored on a set of disks, can be reconstructed by reading a subset of these disks. Previous work shows that coding can achieve the same reliability against failures with lower storage space used. It also allows efficient replacement of disks that have to be removed due to failure or maintenance. We show that in addition to reliability and easy repair, coding also gives faster content download because we only have to wait for content download from a subset of the disks.

5.1.2   Previous Work

Research in coding for distributed storage was galvanized by the results reported in [83]. Prior to that work, literature on distributed storage recognized that, when compared with replication, coding can offer huge storage savings for the same reliability levels. But it was also argued that the benefits of coding are limited, and are outweighed by certain disadvantages and extra complexity. Namely, to provide reliability in multi-disk storage systems, when some disks fail, it must be possible to restore either the exact lost data or an equivalent reliability with minimal download from the remaining storage. This problem of efficient recovery from disk failures was addressed in some early work [84]. But in general, the cost of repair regeneration was considered much higher in coded than in replication systems [85], until [83] established existence and advantages of new regenerating codes. This work was then followed by several related papers, for e.g., [86–88] and references therein.

Currently erasure codes are used for 'cold' that is, less frequently accessed content for which access delay is not pertinent. However for 'hot' or highly accessed content that is frequently requested by many users simultaneously, replication is prevalent. Users can be directed to any one replica of the content. Only recently [39,89,90] was it realized that, in addition to reliability, coding can guarantee the same level of content

accessibility, but with lower storage than replication. In [89], the scenario that when there are multiple requests, all except one of them are blocked and the accessibility is measured in terms of blocking probability is considered. In [90], multiple requests are placed in a queue instead of blocking and the authors propose a scheduling scheme to map requests to servers (or disks) to minimize the waiting time.

Using redundancy in coding for delay reduction has also been studied in the context of packet transmission in [36, 37, 91], and in some content retrieval scenarios [92, 93]. Although they share some common spirit, they do not consider the effect of queueing of requests in coded distributed storage systems.

5.1.3   Our Contributions

In this part we present a queueing-theoeretic approach to understand the delay in content access from coded distributed storage systems. To analyze the content access delay (waiting time plus service time) we introduce a model called the (n, k) fork-join system. In this system, each download request is assigned to queued at n servers that store coded chunks. It exits the system when any k out of n chunks are read. The (n, k) fork-join system is a fundamental generalization of the (n, n) fork-join system studied in queueing theory literature. The (n, n) fork-join system in which all n servers have to be read has been extensively studied in queueing and operations research literature [9, 10, 94]. To the best of our knowledge, we are the first to propose and analysis this queueing model in [20, 21].

This work can also be viewed as a generalization of the setup in Chapter 2. In Chapter 2 we launch r replicas of a task, wait for any one and then cancel the rest. Here, we launch n tasks of a job and wait for any k to finish. This setup, although more general, preceded the work in Chapter 2 via our papers [20, 21].

5.2   Problem Formulation

We now formally define the fork-join model and its variants, and specify the latency and resource cost metrics. Then in Chapter 6 we analyze the trade-off between
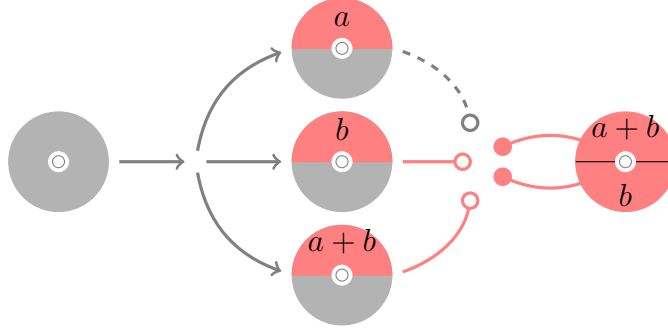
Figure 5-1: Storage is 50% higher, but response time (per server & overall) is reduced.

download latency and resource cost for these fork-join models.

5.2.1   The (n, k) fork-join model and its variants

Consider that a content F of unit size is divided into k chunks of equal size. It is encoded to $n \geq k$ chunks using an (n, k) maximum distance separable (MDS) code, and the coded chunks are stored on n different servers. MDS codes have the property that any k out of the n chunks are sufficient to reconstruct the entire file. An illustrative example with $n = 3$ servers and $k = 2$ is shown in Fig. 5-1. The content F is split into equal blocks a and b, and stored on 3 servers as a, b, and $a \oplus b$, the exclusive-or of blocks a and b. Thus each server stores content of half the size of file F. Downloads from any 2 servers jointly enable reconstruction of F. In this part we show that, in addition to error-correction, we can exploit such codes to reduce the download time of the content.

Consider that download requests arrive according to a Poisson process with rate λ. We refer to each request as a 'job'. Each job can be sent to first-come first-served queues at $r \geq k$ out of the n coded chunks. The time taken to download one coded chunk is modeled by the random variable X, with distribution F_X, and is assumed to be i.i.d. across requests and servers. Dependence across servers due to the content size can be modeled by adding a constant proportional to average job size to service time X. For example, some recent work [43, 44] on analysis of content download from Amazon S3 observed that X is shifted exponential, where Δ is proportional to the
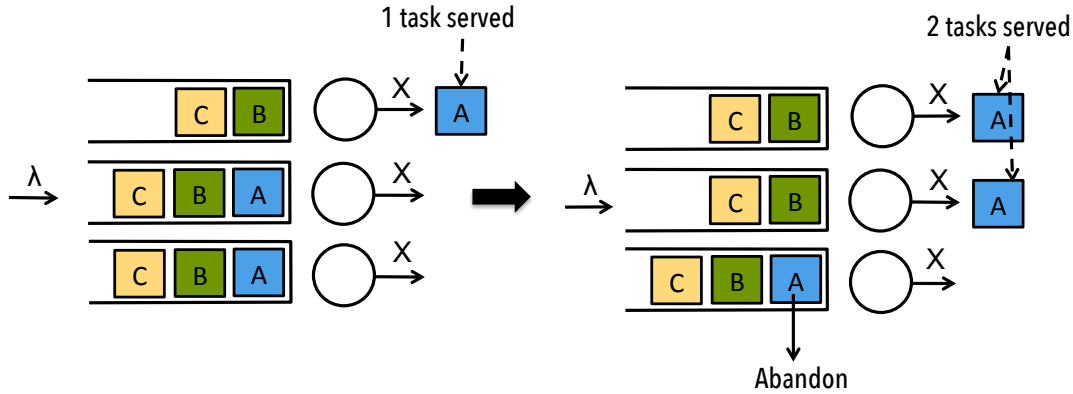
Figure 5-2: The $(3, 2)$ fork-join system. When any 2 tasks of a job finish, the third task abandons its queue.

size of the content and the exponential part is the random delay in starting the data transfer. We use $\bar{F}_X(x) = \Pr(X > x)$ to denote the tail probability function of X. We use $X_{k:n}$ is used to denote the kth smallest of n i.i.d. random variables $X_1, X_2, \ldots X_n$.

Since only k out of the n coded chunks are sufficient to recover the content, the redundant requests serve the purpose of providing diversity against queueing and service delays. We refer to each sub-request to a coded chunk as a 'task'. With this nomenclature, we can now refer to the processing of a download request as a computing job that is divided into r tasks, such that finishing any k tasks is sufficient to complete the job. Thus, going beyond the content download setup, this framework can also be used to estimate the latency of approximate computing.

Depending upon the number of redundant tasks issued and when they are canceled, we can have different queueing models as defined below.

Definition 13 ((n, k) fork-join system)**.** *Each incoming job is forked into n tasks that join a first-come first-serve queue at each of the n servers. When any k tasks finish service, all remaining tasks are canceled and abandon their queues immediately.*

Fig. 5-2 illustrates the $(3, 2)$ fork-join system. The job exits the system when any 2 out of 3 tasks are complete. The $k = 1$ case corresponds to the full replication system with cancel-on-finish studied in Chapter 2, where a job is replicated at n servers and we wait for one of the replicas to finish.
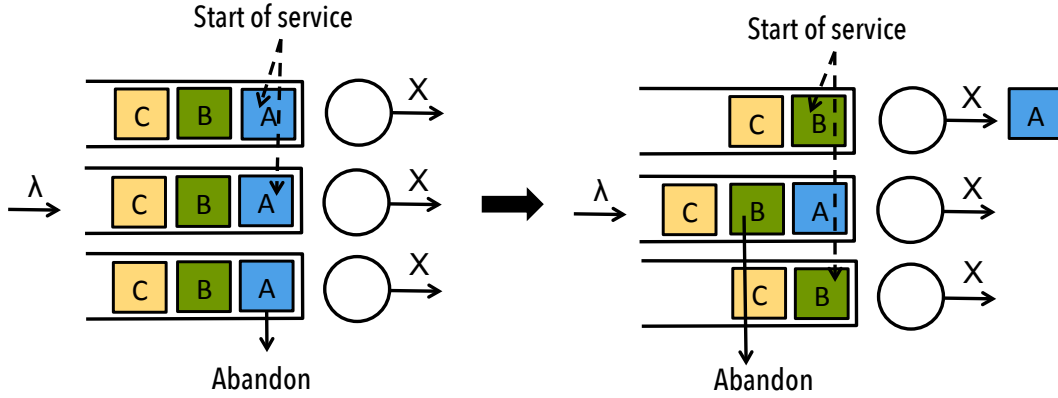
Figure 5-3: The $(3, 2)$ fork-early-cancel system. When any 2 tasks of a job start service, the third task abandons its queue. The job is complete when the 2 tasks finish.

Instead of waiting for k tasks to finish, we could cancel the redundant tasks as soon as k tasks start service. A similar idea has been proposed in systems work [35] in the context of parallel computing. This variant, called the (n, k) fork-early-cancel system is formally defined as follows.

Definition 14 ((n, k) fork-early-cancel system)**.** *An incoming job is forked into n tasks that join queues at the n servers. When any k tasks start service, we cancel the redundant tasks immediately. If more than k tasks start service simultaneously, we retain any k chosen uniformly at random. The job is complete when these k tasks finish.*

Fig. 5-3 illustrates the (n, k) fork-early-cancel system for $n = 3$ and $k = 2$. The $k = 1$ case corresponds to the full replication system with cancel-on-start studied in Chapter 2. Early cancellation of redundant tasks can possibly save computing cost (formally defined in Section 5.2.2), and also reduce queueing delay for subsequent tasks in the queues.

In another variant, we fork a job to r out of the n servers. We refer to this as the (n, r, k) partial fork-join system defined as follows.

Definition 15 ((n, r, k) partial fork-join system)**.** *Each incoming job is forked into $r \geq k$ out of the n servers. When any k tasks finish service, the redundant tasks are canceled immediately and the job exits the system.*

The r servers can be chosen according to a symmetric scheduling policy, for example uniform random, round-robin, least-work-left etc. Launching fewer redundant tasks (r close to k) compromises diversity, but we can save 1) the resource cost, and 2) queueing delay for subsequent tasks. Other variants of the fork-join system include a combination of partial forking and early cancellation explored in Chapter 7, or delaying invocation of some of the redundant tasks. Although not studied in detail here, our analysis techniques can be possibly extended to these variants.

5.2.2 Performance Metrics

Forking a job (download request) into more redundant tasks (requesting more coded chunks) generally results in additional cost of computing resources. We now define the latency and cost metrics, and analyze their trade-off afterwards in Chapter 6.

Definition 16 (Latency). *The latency $\mathbb{E}[T]$ is defined as the expected time from when a job arrives until when k of its tasks are complete.*

Setting $k = 1$ makes the latency definition equivalent to the latency metric in Chapter 2. The cost of redundancy can also be measured in the same way as Chapter 2.

Definition 17 (Computing Cost). *The computing cost $\mathbb{E}[C]$ is the expected total time spent serving the tasks of a job, not including the waiting time in queue.*

Claim 1 in Chapter 2 which shows that the maximum supported rate $\lambda_{max} = n/\mathbb{E}[C]$ also holds for this fork-join framework. By applying it, we can determine the rate of download requests that a coded storage system can support.

In addition to the cost of computing time, we consider the storage overhead, and the network cost of launching and canceling redundant requests. Unlike C, these do not depend on the service time X, but are simply functions of n, r and k.

- **Storage Overhead**: The storage overhead of a file of unit size coded using an (n, k) MDS code is $s = n/k$ units.

- **Network Cost**: The network cost is defined as the number of redundant tasks. It is equal to n for the (n, k) fork-join and fork-early-cancel systems, and r for the (n, r, k) partial-fork-join system.

5.3   Organization

The rest of this part is organized as follows. In Chapter 6, we analyze the latency of the (n, k) fork-join system, and present its trade-off with the computing, storage and network costs. In Chapter 7, we present future directions to generalize the problem setup. In particular, we discuss a generalized fork-join model called the (n, r_f, r, k) fork-join system.

Chapter 6

Latency-Cost Analysis

In this chapter we analyze the latency and cost of the fork-join system and its variants defined in Section 5.2. The latency analysis of fork-join systems is a notoriously hard problem studied in the nineties [8–10]. Even for the traditional fork-join queue ($k = n$ case in Definition 13 with exponential service time), an exact expression for latency can be found only for $n = 2$ [8]. Only bounds are known for general k with exponential F_X [9, 10]. In recent years there is a renewal of interest in fork-join systems due to their application in computing frameworks such as MapReduce.

In Section 6.1 we generalize the latency bounds in [9, 10] to the (n, k) fork-join model, and to any arbitrary service time distribution F_X. We also present the first bounds on computing cost, which help estimate the maximum arrival rate λ_{max} supported by the system. In Section 6.2 we analyze variants the early cancellation and partial forking variants of the (n, k) fork-join model. This section provides insights into the best strategy to assign download requests to the coded chunks, and when to cancel them. All proofs are deferred to Appendix E.

6.1 The (n, k) Fork-join system

In this section we give bounds on the latency and computing cost of the (n, k) fork-join system. Then we discuss the practical implications of these bounds in helping understand how many users can be served by the coded storage, and how fast they
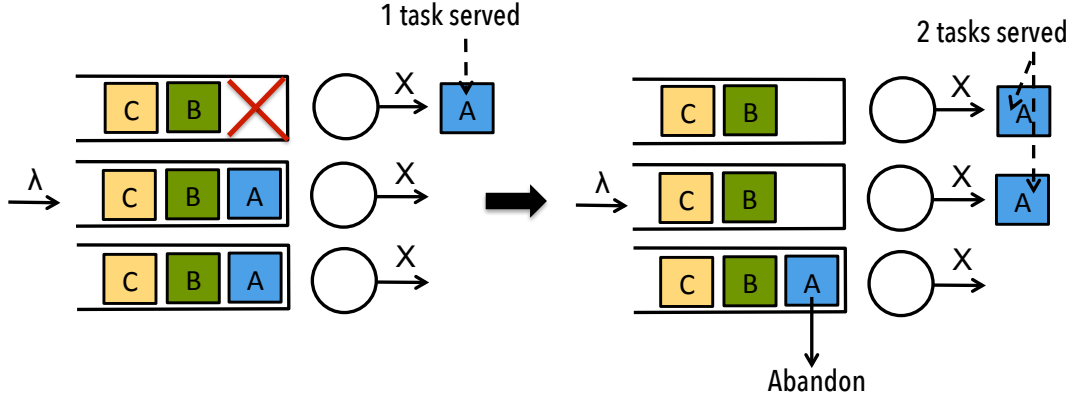
Figure 6-1: The $(3, 2)$ split-merge system. When one task finishes, that server cannot start working on the next task in queue. Only when $k = 2$ tasks are served and the third abandons, the servers can move on to the tasks of job B.

are served. The analysis also helps understand the diversity-parallelism trade-off in choosng the parameter k of the underlying (n, k) maximum-distance-separable (MDS) code.

6.1.1 Bounds on Latency

Theorem 8. *The latency* $\mathbb{E}[T]$ *of the* (n, k) *fork-join system is bounded as*

$$\mathbb{E}[T] \leq \mathbb{E}[X_{k:n}] + \frac{\lambda \mathbb{E}[X_{k:n}^2]}{2(1 - \lambda \mathbb{E}[X_{k:n}])}, \tag{6.1}$$

$$\mathbb{E}[T] \geq \mathbb{E}[X_{k:n}] + \frac{\lambda \mathbb{E}[X_{1:n}^2]}{2(1 - \lambda \mathbb{E}[X_{1:n}])}. \tag{6.2}$$

The detailed proof is given in Appendix E. To get the upper bound (6.13), we use a related queueing system called the split-merge system illustrated in Fig. 6-1. In the (n, k) split-merge system, the servers are blocked from serving subsequent jobs until at least k tasks of the current job are served. Thus, the (n, k) split-merge system gives higher latency than the (n, k) fork-join system. To get the lower bound (6.14), we use the waiting time of the $(n, 1)$ fork-join system to lower bound that of the (n, k) fork-join system.

Fig. 6-2 shows the latency bounds and simulation values vs. k for $n = 10$, $\lambda = 0.5$, and X following the Pareto distribution with $x_m = 0.5$ and $\alpha = 2.5$. For $k = n$, we
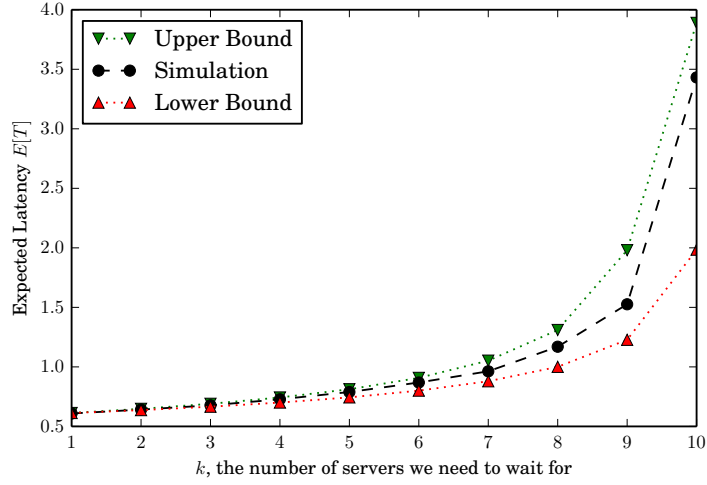
Figure 6-2: Bounds on latency $\mathbb{E}[T]$ versus k, alongside the corresponding simulation values. The service time distribution is $\mathsf{Pareto}(0.5, 2.5)$ with $n = 10$, and $\lambda = 0.5$. The $k = n$ upper bound is evaluated using Lemma 13.

can get a tighter bound than (6.13) by generalizing the approach used in [9].

Lemma 13 (Tighter Upper bound when $k = n$)**.** *For the case $k = n$, another upper bound on latency is given by,*

$$\mathbb{E}[T] \leq \mathbb{E}[\max (R_1, R_2, \cdots R_n)], \tag{6.3}$$

where R_i are i.i.d. realizations of the response time R of an $M/G/1$ queue with arrival rate λ, service distribution F_X.

Transform analysis [54, Chapter 25] can be used to determine the distribution of R, the response time of an $M/G/1$ queue in terms of $F_X(x)$. The Laplace-Stieltjes transform $R(s)$ of the probability density function of $f_R(r)$ of R is given by,

$$R(s) = \frac{sX(s)\left(1 - \frac{\lambda}{\mathbb{E}[X]}\right)}{s - \lambda(1 - X(s))}, \tag{6.4}$$

where $X(s)$ is the Laplace-Stieltjes transform of the service time distribution $f_X(x)$. The upper bound in [9] follows as a corollary to Lemma 13.

Corollary 7 (Equation (2) in [9])**.** *If $k = n$ and the service time distribution F_X is*

109
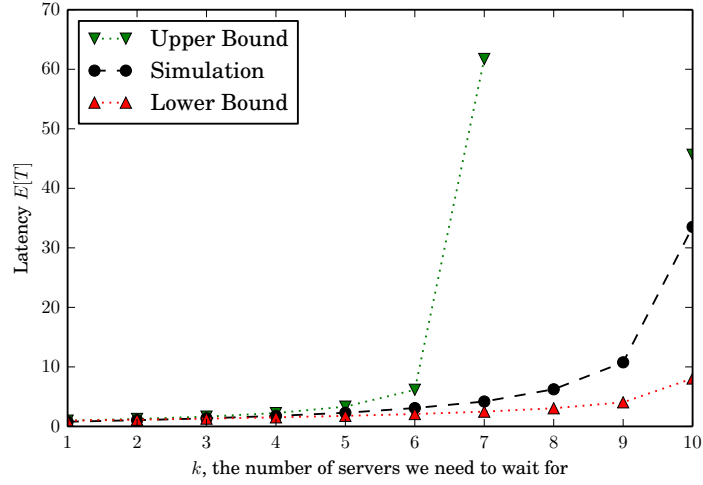
Figure 6-3: Bounds on latency $\mathbb{E}[T]$ versus k, alongside the corresponding simulation values. The service time distribution is $\mathsf{ShiftedExp}(0.5, 0.75)$ with $n = 10$, and $\lambda = 0.5$. The $k = n$ upper bound is evaluated using Lemma 13.

exponential with rate $\mu > \lambda$, the upper bound is given by

$$\mathbb{E}[T] \le H_n \frac{1}{\mu - \lambda} \tag{6.5}$$

where H_n is the nth harmonic number $\sum_{j=0}^{n} 1/j$.

The lower bound (6.14) can be improved if the service time F_X is shifted exponential.

Lemma 14 (Tighter Lower Bound for Shifted Exponential F_X). *The latency $\mathbb{E}[T]$ is lower bounded by,*

$$\mathbb{E}[T] \ge \Delta + \frac{1}{n\mu} + \frac{\lambda \left(\left(\Delta + \frac{1}{n\mu} \right)^2 + \left(\frac{1}{n\mu} \right)^2 \right)}{2 \left(1 - \lambda \left(\Delta + \frac{1}{n\mu} \right) \right)} + \sum_{j=1}^{k-1} \frac{1}{(n-j)\mu - \lambda}. \tag{6.6}$$

When F_X is exponential, the lower bound on latency which is given in [21] follows as a corollary to Lemma 14 by setting $\Delta = 0$.

Corollary 8 (Theorem 2 in [21]). *If the service time distribution F_X is exponential*
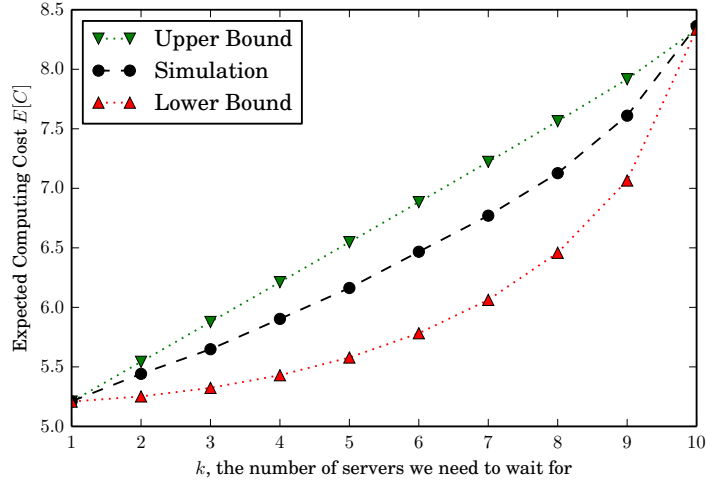
Figure 6-4: Bounds on cost $\mathbb{E}[C]$ versus k, alongside the corresponding simulation values. The service time distribution is $\mathsf{Pareto}(0.5, 2.5)$ with $n = 10$, and $\lambda = 0.5$. The bounds are tight for $k = 1$ and $k = n$.

with rate $\mu > \lambda$, the latency $\mathbb{E}[T]$ is bounded below as

$$\mathbb{E}[T] \geq \sum_{j=0}^{k-1} \frac{1}{(n-j)\mu - \lambda} \tag{6.7}$$

6.1.2   Bounds on Computing Cost

Theorem 9. *The computing cost $\mathbb{E}[C]$ of the (n, k) fork-join system is bounded as*

$$\mathbb{E}[C] \leq (k-1)\mathbb{E}[X] + (n-k+1)\mathbb{E}[X_{1:n-k+1}], \tag{6.8}$$

$$\mathbb{E}[C] \geq \sum_{i=1}^{k} \mathbb{E}[X_{i:n}] + (n-k)\mathbb{E}[X_{1:n-k+1}]. \tag{6.9}$$

The main idea behind proving Theorem 9 is our observation that for each job, some $n - k + 1$ of its tasks start service simultaneously, which allowed us to analyze them separately. The bounds are tight for $k = 1$ and $k = n$ as seen in Fig. 6-4.

Fig. 6-4 shows the bounds alongside simulation plot of the computing cost $\mathbb{E}[C]$ when F_X is $\mathsf{Pareto}(x_m, \alpha)$ with $x_m = 0.5$ and $\alpha = 2.5$. The arrival rate $\lambda = 0.5$, and $n = 10$ with k varying from 1 to 10 on the x-axis. We observe that the bounds on

111
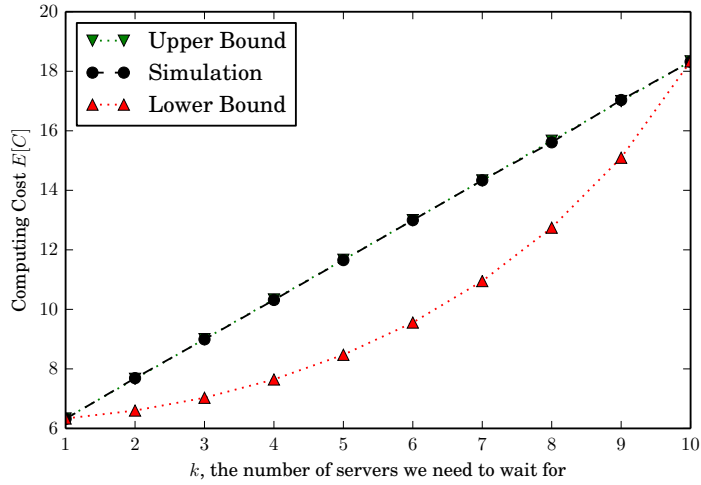
Figure 6-5: Bounds on cost $\mathbb{E}[C]$ versus k, alongside the corresponding simulation values. The service time distribution is ShiftedExp$(0.5, 0.75)$ with $n = 10$, and $\lambda = 0.5$. The upper bound is tight for all k.

$\mathbb{E}[C]$ are tight for $k = 1$ and $k = n$, which can also be inferred from (6.15) and (6.16). In Fig. 6-5 we plot the bounds for $F_X = $ ShiftedExp(Δ, μ), with $\Delta = 0.5$ and $\mu = 0.75$ and all other parameters being same as in Fig. 6-4. When F_X is shifted exponential, the upper bound is in fact tight for all k, $1 \leq k \leq n$ and equals $n\Delta + k/\mu$. This can be proved using the memoryless property of the exponential tail.

The bounds on the computing cost $\mathbb{E}[C]$ given by Theorem 9 allow us to get bounds on the maximum arrival rate $\lambda_{max} = n/\mathbb{E}[C]$ of download requests that the coded storage system is able to support. And the latency bounds in Theorem 8 help estimate the queueing plus service delay experienced by users.

6.1.3 Choosing k: The Diversity-Parallelism Trade-off

In Fig. 6-2 the expected latency $\mathbb{E}[T]$ increases with k, because we need to wait for more tasks to complete, and the service time X is independent of k. But in most computing and storage applications, the service time X is decreases as k increases because each task becomes smaller. We refer to this as the 'parallelism benefit' of splitting a job into more tasks. But as k increases, we lose the 'diversity benefit' provided by having to wait only for a subset of the tasks to finish. Thus, there is a

112

diversity-parallelism trade-off in choosing the optimal k^* that minimizes $\mathbb{E}[T]$.
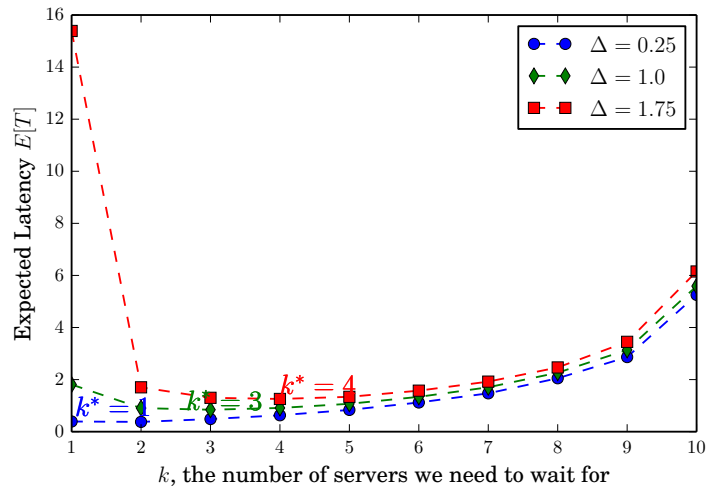
Figure 6-6: Expected latency versus k for task service time $X \sim \mathsf{ShiftedExp}(\Delta/k, 1.0)$, and arrival rate $\lambda = 0.5$. As k increases, we lose diversity but the parallelism benefit is higher because each task is smaller.
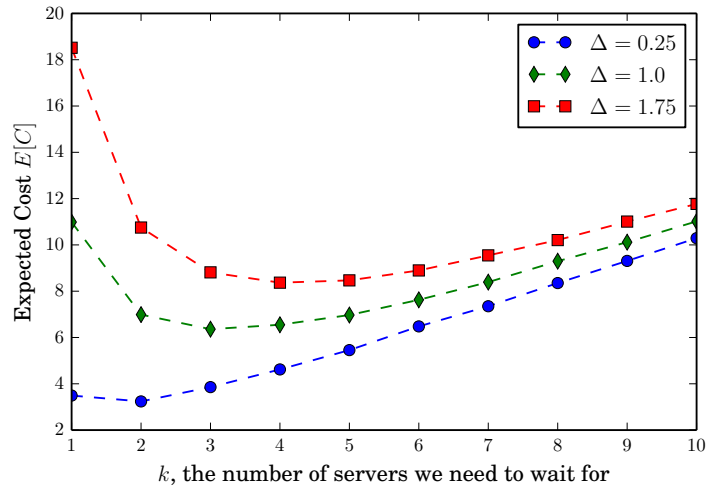
Figure 6-7: Expected cost versus k for task service time $X \sim \mathsf{ShiftedExp}(\Delta/k, 1.0)$, and arrival rate $\lambda = 0.5$. As k increases, we lose diversity but the parallelism benefit is higher because each task is smaller.

We demonstrate this diversity-parallelism trade-off in Fig. 6-6 for service time $X \sim \mathsf{ShiftedExp}(\Delta_k, \mu)$, with $\mu = 1.0$, and $\Delta_k = \Delta/k$. As k increases, we lose diversity but the parallelism benefit is higher because each task is smaller. As Δ increases, the optimal k^* shifted upward because the service distribution becomes

'less random' and so there is less diversity benefit. Fig. 6-7 shows the corresponding computing cost $\mathbb{E}[C]$ as k varies. We observe that for small k, both $\mathbb{E}[T]$ and $\mathbb{E}[C]$ decrease with k. Thus, choosing the right k can reduce both latency and cost.

We can also observe the diversity-parallelism trade-off mathematically in the low traffic regime, for $X \sim \mathsf{ShiftedExp}(\Delta/k, \mu)$. If we take $\lambda \to 0$ in (6.14) and (6.13), both bounds coincide and we get,

$$\lim_{\lambda \to \infty} \mathbb{E}[T] = \mathbb{E}[X_{k:n}] = \frac{\Delta}{k} + \frac{H_n - H_{n-k}}{\mu}, \tag{6.10}$$

where $H_n = \sum_{i=1}^{n} 1/i$, the nth harmonic number. The parallelism benefit comes from the first term in (6.10), which reduces with k. The diversity of waiting for k out of n tasks causes the second term to increase with k. The optimal k^* that minimizes (6.10) strikes a balance between these two opposing trends.
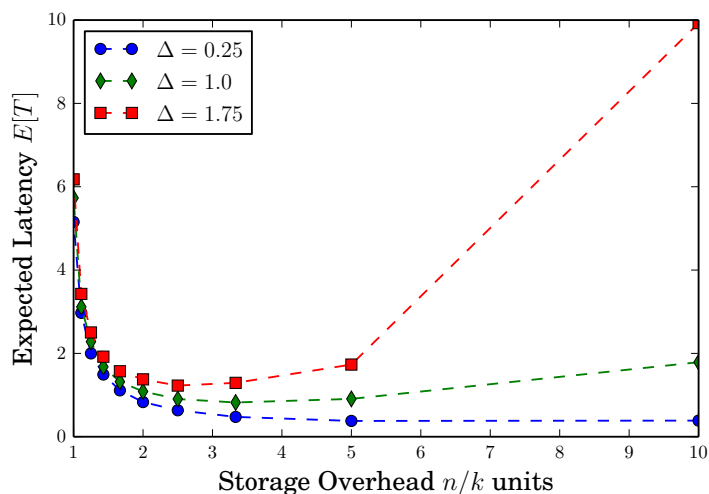
Figure 6-8: Expected latency versus storage overhead for task service time $X \sim \mathsf{ShiftedExp}(\Delta/k, 1.0)$, and arrival rate $\lambda = 0.5$. For a storage overhead of less than 2, we get a significant latency reduction.

Fig. 6-8 shows the trade-off between latency $\mathbb{E}[T]$ and storage overhead n/k. The task service time $X \sim \mathsf{ShiftedExp}(\Delta/k, 1.0)$, and arrival rate $\lambda = 0.5$. As k decreases, $\mathbb{E}[T]$ decreases because we need to wait for fewer tasks, but the storage overhead increases. When k decreases from 10 to 5, the storage overhead is 2 units. Thus, at the expense of this additional storage cost, we get a significant latency reduction.

6.2 Variants of the (n, k) fork-join system

We now analyze the latency and cost of the two fork-join variants defined in Section 5.2. Comparing the variants with the (n, k) fork-join system helps determine the best policy to issue and cancel redundant tasks.

6.2.1 The (n, k) fork-early-cancel system

Theorem 10 (Latency-Cost with Early Cancellation). *The cost $\mathbb{E}[C]$ and an upper bound expected latency $\mathbb{E}[T]$ with early cancellation is given by*

$$\mathbb{E}[C] = k\mathbb{E}[X] \tag{6.11}$$

$$\mathbb{E}[T] \leq \mathbb{E}[\max(R_1, R_2, \cdots R_k)] \tag{6.12}$$

where R_i are i.i.d. realizations of R, the reponse time of an $M/G/1$ queue with arrival rate $\lambda k/n$ and service distribution F_X.

The Laplace-Stieltjes transform of the response time R of an $M/G/1$ queue with service distribution $F_X(x)$ and arrival rate is same as (6.4), with λ replaced by $\lambda k/n$.
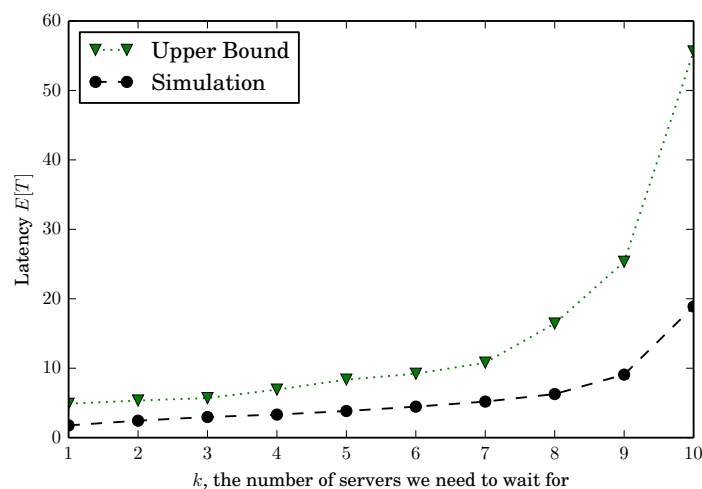
Figure 6-9: Upper bound on latency $\mathbb{E}[T]$ with early cancellation versus k, alongside the corresponding simulation values. The service time distribution is ShiftedExp$(0.5, 0.75)$ with $n = 10$, and $\lambda = 0.5$.
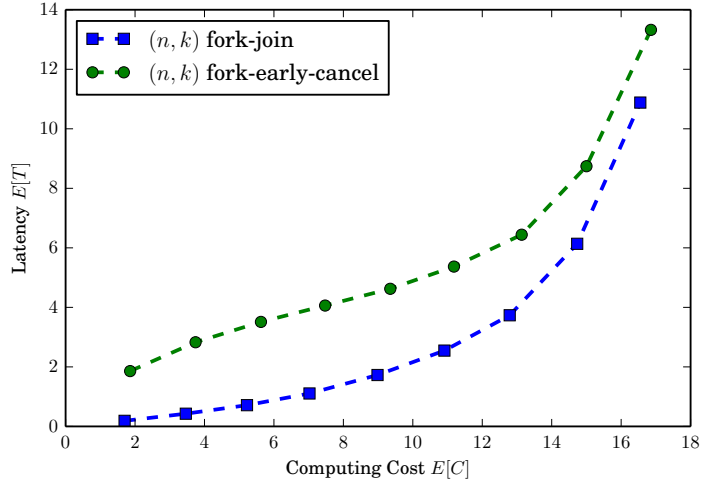
Figure 6-10: Expected latency $\mathbb{E}[T]$ versus computing cost $\mathbb{E}[C]$ as k varies. The task service time $X \sim \mathsf{HyperExp}(0.1, 1.5, 0.5)$ and arrival rate $\lambda = 0.5$. For such log-convex distributions, the (n, k) fork-join performs better for all k.

Fig. 6-9 shows the upper bound on $\mathbb{E}[T]$ with early cancellation, alongside the simulation values. The number of servers $n = 10$, the service time distribution is $\mathsf{ShiftedExp}(0.5, 0.75)$, and $\lambda = 0.5$.

By comparing the cost $\mathbb{E}[C] = k\mathbb{E}[X]$ in (6.11) to the bounds in Theorem 9 without early cancellation, we can get insights into when early cancellation is effective for a given service time distribution F_X. For example, when $\bar{F}_X$ is log-convex, the upper bound in (6.15) is smaller than $k\mathbb{E}[X]$. Thus we can infer that early cancellation is not effective when X is log-convex, as demonstrated in Fig. 6-10. This insight also matches with that in Fig. 2-8 for the $k = 1$ case.

6.2.2 The (n, r, k) partial-fork-join system

Instead of forking each download request (job) to all n coded chunks (dividing the job into n tasks), we can use partial forking. This variant is refered to as (n, r, k) partial fork-join system as defined in Definition 15. Since the latency-cost analysis of (n, k) fork-join system is hard as seen in Section 6.1.1 and Section 6.1.2, the analysis of its generalization, the (n, r, k) partial-fork-join is even harder. However it is possible for a group-based scheduling policy as given by Lemma 15 below.

Lemma 15. *Consider that the n servers are divided into n/r groups of $r \geq k$ servers each, assuming r divides n. Each incoming job is forked to the one of the groups chosen uniformly at random. Then latency and cost can be bounded as follows*

$$\mathbb{E}[T] \leq \mathbb{E}[X_{k:r}] + \frac{\lambda r \mathbb{E}[X_{k:r}^2]}{2(n - \lambda r \mathbb{E}[X_{k:r}])}, \tag{6.13}$$

$$\mathbb{E}[T] \geq \mathbb{E}[X_{k:r}] + \frac{\lambda r \mathbb{E}[X_{1:r}^2]}{2(n - \lambda r \mathbb{E}[X_{1:r}])}. \tag{6.14}$$

$$\mathbb{E}[C] \leq (k-1)\mathbb{E}[X] + (r-k+1)\mathbb{E}[X_{1:r-k+1}], \tag{6.15}$$

$$\mathbb{E}[C] \geq \sum_{i=1}^{k} \mathbb{E}[X_{i:r}] + (r-k)\mathbb{E}[X_{1:r-k+1}]. \tag{6.16}$$

Proof. Each group of r servers behaves like an independent (r, k) fork-join system with arrival rate $\lambda r/n$. Then using Theorem 8 and Theorem 9 we get the above bounds for the (n, r, k) partial-fork-join system. $\qquad\square$

These bounds can help determine the optimal r for a given X, k, and λ.

6.3   Concluding Remarks

In this chapter we presented a latency-cost analysis of the (n, k) fork-join system and its two variants: the (n, k) fork-early-cancel and (n, r, k) partial-fork-join systems, defined in Chapter 5. These bounds help understand how many users can be supported by a coded storage systems, and how fast they can be served. They also provide insights into the choice of the parameter k on the underlying (n, k) code. The insights on how the service distribution X affects the redundancy strategy are similar to Chapter 2. Log-concave distributions benefit less from redundancy (smaller k).

Chapter 7

Future Directions

In this chapter we present future research directions generalizing the problem setup in Chapter 5. In Section 7.1 we propose and analyze the (n, r_f, r, k) model, which is a combination of the (n, k) fork-early-cancel and (n, r, k) partial-fork-join systems. As an alternative to the (n, k) MDS codes studied in our work, in Section 7.2 and Section 7.3 we discuss the use of other erasure codes for fast content download.

7.1 The (n, r_f, r, k) fork-join model

We introduce a general fork-join variant that is a combination of the partial fork introduced in Section 5.2, and partial early cancellation of redundant tasks. This model helps formulate a redundancy strategy to minimize the latency, subject to computing and network cost constraints. This strategy can also be used on traces of task service time when a closed-form expressions of F_X and its order statistics are not known.

Definition 18 ((n, r_f, r, k) fork-join system)**.** *For a system of n servers and a job that requires k tasks to complete, we do the following:*

- *Fork the job to r_f out of the n servers.*

- *When any $r \leq r_f$ tasks are at the head of queues or in service already, cancel all other tasks immediately. If more than r tasks start service simultaneously,*

retain r randomly chosen ones out of them.

- *When any $k \leq r$ tasks finish, cancel all remaining tasks immediately.*

Note that k tasks may finish before some r start service, and thus we may not need to perform the partial early cancellation in the second step above.

The $r_f - r$ tasks that are canceled early, help find r queues out of the r_f with the least work left, thus reducing waiting time. From the r tasks retained, waiting for any k to finish provides diversity and hence reduces service time.

The special cases (n, n, n, k), (n, n, k, k) and (n, r, r, k) correspond to the (n, k) fork-join and (n, k) fork-early-cancel and (n, r, k) partial-fork-join systems respectively.

7.1.1 Choosing Parameters r_f and r

We propose a strategy to choose r_f and r to minimize expected latency $\mathbb{E}[T]$, subject to a computing cost constraint is $\mathbb{E}[C] \leq \gamma$, and a network cost constraint is $r_f \leq r_{max}$. We impose the second constraint because forking to more servers results in higher network cost of remote-procedure-calls (RPCs) to launch and cancel the tasks.

Claim 4 (General Redundancy Strategy). *Good heuristic choices of r_f and r to minimize $\mathbb{E}[T]$ subject to constraints $\mathbb{E}[C] \leq \gamma$ and $r_f \leq r_{max}$ are*

$$r_f^* = r_{max}, \tag{7.1}$$

$$r^* = \underset{r \in [0, r_{max}]}{\arg\min} \hat{T}(r), \quad s.t. \quad \hat{C}(r) \leq \gamma \tag{7.2}$$

where $\hat{T}(r)$ and $\hat{C}(r)$ are estimates of the expected latency $\mathbb{E}[T]$ and cost $\mathbb{E}[C]$, defined as follows:

$$\hat{T}(r) \triangleq \mathbb{E}[X_{k:r}] + \frac{\lambda r \mathbb{E}[X_{k:r}^2]}{2(n - \lambda r \mathbb{E}[X_{k:r}])}, \tag{7.3}$$

$$\hat{C}(r) \triangleq r \mathbb{E}[X_{k:r}]. \tag{7.4}$$

120

To justify the strategy above, observe that for a given r, increasing r_f gives higher diversity in finding the shortest queues and thus reduces latency. Since $r_f - r$ tasks are canceled early before starting service, r_f affects $\mathbb{E}[C]$ only mildly, through the relative task start times of r tasks that are retained. So we conjecture that it is optimal to set $r_f = r_{max}$ in (7.1), the maximum value possible under network cost constraints. Changing r on the other hand does affect both the computing cost and latency significantly. Thus to determine the optimal r, we minimize $\hat{T}(r)$ subject to constraints $\hat{C}(r) \leq \gamma$ and $r \leq r_{max}$ as given in (7.2).

The estimates $\hat{T}(r)$ and $\hat{C}(r)$ are obtained by generalizing Lemma 4 for group-based random forking to any k, and r that may not divide n. When the order statistics of F_X are hard to compute, or F_X itself is not explicitly known, $\hat{T}(r)$ and $\hat{C}(r)$ can be also be found using empirical traces of X.

The sources of inaccuracy in the estimates $\hat{T}(r)$ and $\hat{C}(r)$ are as follows.

1. For $k > 1$, the latency estimate $\hat{T}(r)$ is a generalization of the split-merge queueing upper bound in Theorem 8. Since the bound becomes loose as k increases, the error $|\hat{T}(r) - \mathbb{E}[T]|$ increases with k.

2. The estimates $\hat{T}(r)$ and $\hat{C}(r)$ are by definition independent of r_f, which is not true in practice. As explained above, for $r_f > r$, the actual $\mathbb{E}[T]$ is generally less than $\hat{T}(r)$, and $\mathbb{E}[C]$ can be slightly higher or lower than $\hat{C}(r)$.

3. Since the estimates $\hat{T}(r)$ and $\hat{C}(r)$ are based on group-based forking, they consider that all r tasks start simultaneously. Variability in relative task start times can result in actual latency and cost that are different from the estimates. For example, from Theorem 3 we can infer that when $\bar{F}_X$ is log-concave (log-convex), the actual computing cost $\mathbb{E}[C]$ is less than (greater than) $\hat{C}(r)$.

The factor (1) above is the largest source of inaccuracy, especially for larger k and λ. Since the estimate $\hat{T}_r$ is an upper bound on the actual latency, the r^* and r_f^* recommended by the strategy are smaller than or equal to their optimal values. Factors (2) and (3) only affect the relative task start times and generally result in a smaller error in estimating $\mathbb{E}[T]$ and $\mathbb{E}[C]$.

7.1.2 Simulation Results

We now present simulation results comparing the proposed strategy given in Claim 4 with the (n, r, k) partial-fork-join system with r varying from k to n. The service time distributions considered here are neither log-concave nor log-convex, thus making it hard to directly infer the best redundancy strategy using the analysis presented in the previous sections.

In Fig. 7-1 the service time $X \sim \mathsf{Pareto}(1, 2.2)$, $n = 10$, $k = 1$, and arrival rate $\lambda = 0.25$. The computing and network cost constraints are $\mathbb{E}[C] \leq 5$ and $r_f \leq 7$ respectively. We observe that the proposed strategy gives a significant latency reduction as compared to the no redundancy case ($r = k$ in the (n, r, k) partial-fork-join system). Subject to the computing and network cost constraints, $r = 4$ minimizes the latency of the (n, r, k) partial-fork-join system. We observe that the proposed strategy gives a latency-cost trade-off very close to that of the best partial-fork-join system. Using partial early cancellation ($r_f > r$) in the proposed strategy gives a slight reduction in latency in comparison with the best (n, r, k) partial-fork-join system. The cost $\mathbb{E}[C]$ increases slightly, but remains less than γ.
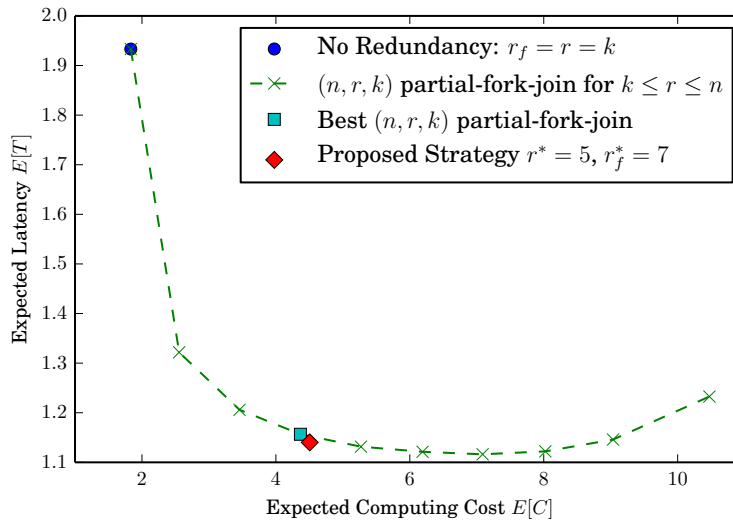
Figure 7-1: The latency-cost trade-off of the proposed redundancy strategy is close to that of the best (n, r, k) partial-fork-join system. Service time $X \sim \mathsf{Pareto}(1, 2.2)$, and the cost constraints are $\mathbb{E}[C] \leq 5$ and $r \leq r_f \leq 7$ The first constraint is active in this example.
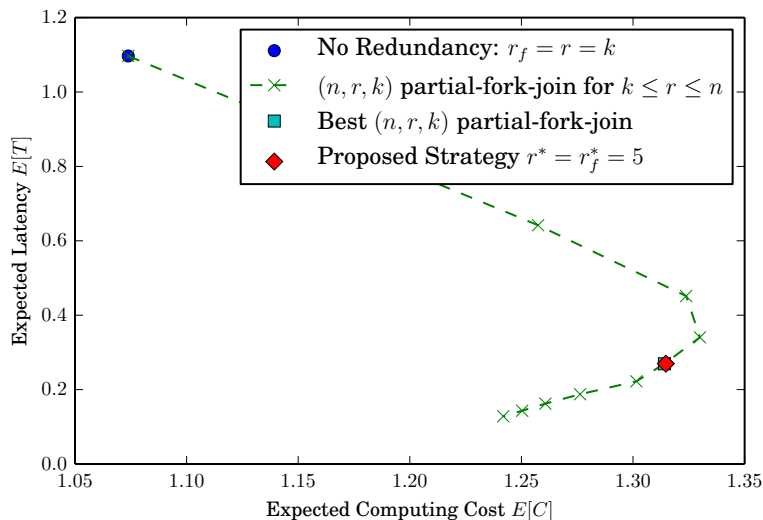
Figure 7-2: The latency-cost trade-off of the proposed redundancy strategy is close to that of the best (n, r, k) partial-fork-join system. The service time X is an equiprobable mixture of $\mathsf{Exp}(2)$ and $\mathsf{ShiftedExp}(1, 1.5)$, and the cost constraints are $\mathbb{E}[C] \leq 2$ and $r \leq r_f \leq 5$. The second constraint is active in this example.

In Fig. 7-2 we show a case where the cost $\mathbb{E}[C]$ does not always increase with the amount of redundancy r. The task service time X is a mixture of an exponential $\mathsf{Exp}(2)$ and a shifted exponential $\mathsf{ShiftedExp}(1, 1.5)$, each occurring with equal probability. All other parameters are same as in Fig. 7-1. The proposed strategy found using Claim 4 is $r^* = r_f^* = r_{max} = 5$, limited by the $r_f \leq r_{max}$ constraint rather than the $\mathbb{E}[C] \leq \gamma$ constraint. It coincides with the best (n, r, k) partial-fork-join system.

7.2   Availability Codes

In this part we considered that the content is coded using an (n, k) maximum distance separable (MDS) code. MDS codes provide reliability against failure of any $n - k$ out of the n servers. However, repairing a failed node can be expensive because k chunks need to downloaded from other servers. This can cost a prohibitive amount of network bandwidth. Regenerating codes proposed in [5, 86] allow low-cost repair with minimum amount of data communicated over the network. Locally repairable codes [95, 96] minimize the number of nodes accessed to repair failed nodes.

Due to these properties erasure codes are starting to be used at a large-scale in cloud storage systems. However, they are mostly used to store 'cold' or less frequently accessed data. As more 'hot' data is erasure-coded in cloud systems, ensuring fast content access is important, in addition to reliability and easy repair. This calls for codes that are designed for fast content download. One such class of codes is proposed in [11], building on locally repairable codes. These codes allow parallel reads from disjoint sets of nodes, while still maintaining a high code rate. A code is said to have (r, t)-availability if the content distributed across on n nodes can be recovered from t disjoint groups of r nodes each.

Recently [12] analyzed the delay of these codes using our fork-join queueing framework. To alleviate the difficulty of analysis encountered in [12], we plan to focus on maximizing the request arrival rate λ_{max} that a coded storage system can support. The analysis of λ_{max} would be more tractable than $\mathbb{E}[T]$. We believe that [11,12] are only the beginning of the construction and analysis of codes for fast content download from distributed storage.

7.3   Multiple Fountains

In the codes discussed so far, an entire chunk of the content is downloaded from k out of the n chunks. Partially downloaded chunks cannot be used to decode the file. If the content is coded using a rateless fountain code [97,98], then such decoding would be possible. For example, suppose a content file consisting of K packets. We can create M coded combinations using a fountain code such that any K (slightly more than K in practice) packets are sufficient to decode the file. The M coded combinations can be distributed across n servers. Then, downloading k_1, k_2, .. k_n packets each from the servers such that $k_1 + k_2 + \cdots + k_n \geq K$ is sufficient to decode the file. A future direction is to analyze the delay and determine the optimal way to store and request coded packets from the servers.

Part III

Erasure Coding for Smooth Streaming

Chapter 8

Effect of Block-wise Feedback in Point-to-point Streaming

8.1 Introduction

8.1.1 Motivation

A recent report [99] shows that 62% of the Internet traffic in North America comes from real-time streaming applications. Unlike traditional file transfer where only total delay matters, streaming imposes delay constraints on each individual packet. Further, many applications require in-order playback of packets at the receiver. Packets received out of order are buffered until the missing packets in the sequence are successfully decoded. In audio and video applications some packets can be dropped without affecting the streaming quality. However, other applications such as remote desktop, and collaborative tools such as Dropbox [81] and Google Docs [82] have strict order constraints on packets, where packets represent instructions that need to be executed in order at the receiver.

Thus, there is a need to develop transmission schemes that can ensure in-order packet delivery to the user, with efficient use of available bandwidth. To ensure that packets are decoded in order, the transmission scheme must give higher priority to older packets that were delayed, or received in error. However, repeating old packets

instead of transmitting new packets results in a loss in the overall rate of packet delivery to the user, i.e., the throughput. Thus there is a fundamental trade-off between throughput and in-order decoding delay.

The throughput loss incurred to achieve smooth in-order packet delivery can be significantly reduced if the source receives feedback about packet losses. Then the source can adapt its future transmission strategy to strike the right balance between old and new packets. In this chapter we study this interplay between feedback and the throughput-smoothness trade-off.

8.1.2 Previous Work

When there is immediate and error-free feedback, it is well understood that a simple Automatic-repeat-request (ARQ) scheme is both throughput and delay optimal. But only a few papers in literature have analyzed streaming codes with delayed or no feedback. Fountain codes [97] are capacity-achieving erasure codes, but they are not suitable for streaming because the decoding delay is proportional to the size of the data. Streaming codes without feedback for constrained channels such as adversarial and cyclic burst erasure channels were first proposed in [100], and also extensively explored in [101, 102]. The thesis [100] also proposed codes for more general erasure models and analyzed their decoding delay. These codes are based upon sending linear combinations of source packets; indeed, it can be shown that there is no loss in restricting the codes to be linear.

However, decoding delay does not capture *in order* packet delivery, which is required for streaming applications. This aspect is captured in the delay metrics in [103] and [104], which consider that packets are played in-order at the receiver. The authors in [103] analyze the playback delay of real-time streaming for uncoded packet transmission over a channel with long feedback delay. In [104, 105] we show that the number of interruptions in playback scales $\Theta(\log n)$ for a stream of length n.

In this chapter we use a metric called smoothness exponent to measure the quality of streaming. We aim to understand how the frequency of feedback about erasures affects the design of codes to ensure smooth point-to-point streaming. In Section 8.2

and Section 8.3 we describe the system model, and preliminary concepts respectively. Then we consider the extreme cases of immediate feedback and no feedback in Section 8.4 and Section 8.5 respectively. In Section 8.6 we propose coding schemes for the general case of block-wise feedback after every d slots. The longer proofs are deferred to Appendix F.

8.2 System Model

8.2.1 Source and Channel Model

The source has a large stream of packets $s_1, s_2, \cdots, s_n$ to be transmitted to a user over a point-to-point channel. The encoder creates a coded packet $y_n = f(s_1, \ s_2 \ .. s_n)$ in each slot n and transmits it over the channel. The encoding function f is known to the receiver. For example, if y_n is a linear combination of the source packets, the coefficients are included in the transmitted packet so that the receiver can use them to decode the source packets from the coded combination. Without loss of generality, we can assume that y_n is a linear combination of the source packets. The coefficients are chosen from a large enough field such that the coded combinations are independent with high probability.

Each coded combination is transmitted to the user over an i.i.d. erasure channel such that every transmitted packet is received successfully with probability p, and otherwise received in error and discarded. An erasure channel is a good model when encoded packets have a set of checksum bits that can be used to verify with high probability whether the received packet is error-free.

8.2.2 Packet Delivery

The application at the user requires the stream of packets to be *in order*. Packets received out of order are buffered until the missing packets in the sequence are decoded. We assume that the buffer is large enough to store all the out-of-order packets. Every time the earliest undecoded packet is decoded, a burst of in-order decoded packets is

delivered to the application. For example, suppose that s_1 has been delivered and s_3, s_4, s_6 are decoded and waiting in the buffer. If s_2 is decoded in the next slot, then s_2, s_3 and s_4 are delivered to the application.

8.2.3 Feedback Model

We consider that the source receives block-wise feedback about channel erasures after every d slots. Thus, before transmitting in slot $kd + 1$, for all integers $k \geq 1$, the source knows about the erasures in slots $(k-1)d+1$ to kd. It can use this information to adapt its transmission strategy in slot $kd + 1$. Block-wise feedback can be used to model a half-duplex communication channel where after every d slots of packet transmission, the channel is reserved to send d bits of feedback to the source about the status of decoding. The extreme case $d = 1$, corresponds to immediate feedback when the source has complete knowledge about past erasures. And when $d \to \infty$, the block-wise feedback model converges to the scenario where there is no feedback to the source. Note that the feedback can be used to estimate p, the success probablity of the erasure channel, when it is unknown to the source. Thus, the coding schemes we propose for $d < \infty$ are universal; they can be used even when the channel quality of unknown to the source.

8.3 Preliminaries

8.3.1 Notions of Packet Decoding

We now define some notions of packet decoding that aid the presentation and analysis of coding schemes in the following chapters.

Definition 19 (Innovative Packets). *A coded packet is said to be innovative if it is linear independent with respect to the coded packets received by the user until that time.*

Definition 20 (Seen Packets). *The transmitted marks a packet s_k as "seen" by a*

user when it knows that the user has successfully received a coded combination that only includes s_k and packets s_i for $1 \le i < k$.

Since the packets are required strictly in-order, the transmitter can stop including s_k in coded packets when it is seen by the user. This is because the user can decode s_k once all s_i for $i < k$ are decoded.

8.3.2 Throughput and Delay Metrics

We now define the metrics for throughput and smoothness of in-order packet delivery.

Definition 21 (Throughput)**.** *If I_n is the number of packets delivered in-order to a user until time n, the throughput is defined as,*

$$\tau = \lim_{n \to \infty} \frac{I_n}{n}. \tag{8.1}$$

The maximum possible throughput is $\tau = p$, where p is the success probability of the erasure channel. The receiver application may require a minimum level of throughput. For example, if applications with playback require τ to be greater than the playback rate. The bandwidth required is proportional to $1/\tau$.

The throughput captures the overall rate at which packets are delivered, irrespective of their delays. If the channel did not have any erasures, packet s_k would be delivered to the user in slot k. The random erasures, and absence of immediate feedback about past erasures results in variation in the time at which packets are delivered. We capture the burstiness in packet delivery using the following delay metric.

Definition 22 (Smoothness Exponent)**.** *Let D_k be in-order decoding delay of packet s_k, the earliest time at which all packets $p_1, \cdots p_k$ are decoded. The smoothness exponent $\gamma_k^{(s)}$ is defined as the asymptotic decay rate of D_k, which is given by*

$$\gamma_k^{(s)} = -\lim_{n \to \infty} \frac{\log \Pr(D_k > n)}{n} \tag{8.2}$$

The relation (8.2) can also be stated as $\Pr(D > n) \doteq e^{-n\gamma}$ where $\doteq$ stands for asymptotic equality defined in [106, Page 63]. For simplicity of analysis we define another delay exponent, the inter-delivery defined as follows. Theorem 11 shows the equivalence of the smoothness and the inter-delivery exponent for time-invariant schemes.

Definition 23 (Inter-delivery Exponent). *Let T_1 be the first inter-delivery time, that is, the first time instant when one or more packets are decoded in-order. The inter-delivery exponent λ is defined as the asymptotic decay rate of T_1, which is given by*

$$\lambda = -\lim_{n \to \infty} \frac{\log \Pr(T_1 > n)}{n} \tag{8.3}$$

In this work we focus on time-invariant transmission schemes where the coding strategy is fixed across blocks of transmission, formally defined as follows.

Definition 24 (Time-invariant schemes). *A time-invariant scheme is represented by a vector $\mathbf{x} = [x_1, \cdots x_d]$ where x_i, for $1 \le i \le d$, are non-negative integers such that $\sum_i x_i = d$. In each block we transmit x_i independent linear combinations of the i lowest-index unseen packets in the stream, for $1 \le i \le d$.*

The above class of schemes is referred to as time-invariant because the vector $\mathbf{x}$ is fixed across all blocks. Note that there is no loss of generality in restricting the length of the vector $\mathbf{x}$ to d. This is because each block can provide only up to d innovative coded packets, and hence there is no advantage in adding more than d unseen packets to the stream in a given block.

Theorem 11. *For a time-invariant scheme, the smoothness exponent $\gamma_k^{(s)}$ of packet s_k for any $k \le \infty$ is equal to λ, the inter-delivery exponent.*

The proof is given in Appendix F. As a result of this equivalence between $\gamma_k^{(s)}$ and λ, we study the trade-off between throughput τ and the inter-delivery exponent λ in the rest of this chapter.

8.4 Immediate Feedback

We first consider the extreme case of immediate feedback ($d = 1$), where the source has complete knowledge of past erasures before transmitting each packet. We can show that a simple automatic-repeat-request (ARQ) scheme is optimal in both τ and λ. In this scheme, the source transmits the lowest index unseen packet, and repeats it until the packet successfully goes through the channel.

Since a new packet is received in every successful slot, the throughput $\tau = p$, the success probability of the erasure channel. The ARQ scheme is throughput-optimal because the throughput $\tau = p$ is equal to the information-theoretic capacity of the erasure channel [106]. Moreover, it also gives the optimal the inter-delivery exponent λ because one in-order packet is decoded in every successful slot. To find λ, first observe that the tail distribution of the time T_1, the first inter-delivery time is,

$$\Pr(T_1 > n) = (1 - p)^n \tag{8.4}$$

Substituting this in Definition 23 we get the exponent $\lambda = -\log(1 - p)$. Thus, the trade-off for the immediate feedback case is $(\tau, \lambda) = (p, -\log(1 - p))$.

From this analysis of the immediate feedback case we can find limits on the range of achievable (τ, λ) for any feedback delay d. Since a scheme with immediate feedback can always simulate one with delayed feedback, the throughput and delay metrics (τ, λ) achievable for any feedback delay d must lie in the region $0 \leq \tau \leq p$, and $0 \leq \lambda \leq -\log(1 - p)$.

8.5 No Feedback

Now we consider the other extreme case ($d = \infty$), corresponding to when there is no feedback to the source. We propose a coding scheme that gives the best (τ, λ) trade-off among the class of full-rank codes, defined as follows.

Definition 25 (Full-rank Codes). *In slot n we transmit a linear combination of all packets s_1 to $s_{V[n]}$. We refer to $V[n]$ as the transmit index in slot n.*
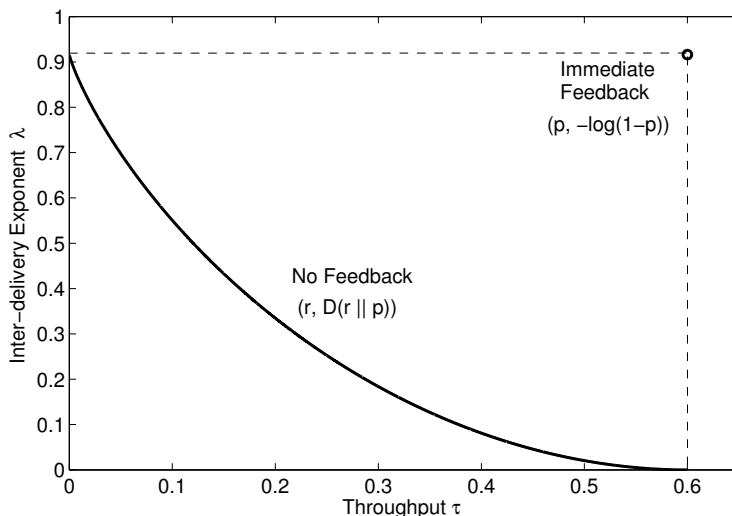
Figure 8-1: The trade-off between inter-delivery exponent λ and throughput τ with success probability $p = 0.6$ for the immediate feedback ($d = 1$) and no feedback ($d = \infty$) cases.

Conjecture 1. *Given transmit index $V[n]$, there is no loss of generality in including all packets s_1 to $s_{V[n]}$.*

We believe this conjecture is true because the packets are required in-order at the receiver. Thus, every packet s_j, $j < V[n]$ is required before packet $s_{V[n]}$ and there is no advantage in excluding s_j from the combination. Hence we believe that there is no loss of generality in restricting our attention to full-rank codes. A direct approach to verifying this conjecture would involve checking all possible channel erasure patterns.

Theorem 12. *The optimal throughput-smoothness trade-off among full-rank codes is $(\tau, \lambda) = (r, D(r\|p))$ for all $0 \leq r < p$. It is achieved by the coding scheme with $V[n] = \lceil rn \rceil$ for all n.*

The term $D(r\|p)$ is the binary information divergence function, which is defined for $0 < p, r < 1$ as

$$D(r\|p) = r \log \frac{r}{p} + (1 - r) \log \frac{1 - r}{1 - p}, \tag{8.5}$$

where $0 \log 0$ is assumed to be 0. As $r \to 0$, $D(r\|p)$ converges to $-\log(1 - p)$, which is the best possible λ as given in Section 8.4.

134

Fig. 8-1 shows the (τ, λ) trade-off for the immediate feedback and no feedback cases, with success probability $p = 0.6$. The optimal trade-off with any feedback delay d lies in between these two extreme cases.

8.6   General Block-wise Feedback

We now analyze the (τ, λ) trade-off with general block-wise feedback delay of d slots. We restrict our attention to the class of time-invariant coding schemes defined in Section 8.3.2.

Given a vector $\mathbf{x}$, define p_d, as the probability of decoding the first unseen packet during the block, and S_d as the number of innovative coded packets that are received during that block. We can express $\tau_\mathbf{x}$ and $\lambda_\mathbf{x}$ in terms of p_d and S_d as,

$$(\tau_\mathbf{x}, \lambda_\mathbf{x}) = \left(\frac{\mathbb{E}[S_d]}{d}, -\frac{1}{d} \log(1 - p_d) \right),\qquad(8.6)$$

where we get throughput $\tau_\mathbf{x}$ by normalizing the $\mathbb{E}[S_d]$ by the number of slots in the slots. We can show that the probability $\Pr(T_1 > kd)$ of no in-order packet being decoded in k blocks is equal $(1 - p_d)^k$. Substituting this in (8.3) we get $\lambda_\mathbf{x}$.

Example 1. *Consider the time-invariant scheme* $\mathbf{x} = [1, 0, 3, 0]$ *where block size* $d = 4$. *That is, we transmit* 1 *combination of the first unseen packet, and* 3 *combinations of the first* 3 *unseen packets. Fig. 8-2 illustrates this scheme for one channel realization. The probability* p_d *and* $\mathbb{E}[S_d]$ *are,*

$$p_d = p + (1 - p) \binom{3}{3} p^3 (1 - p)^0 = p + (1 - p) p^3,\qquad(8.7)$$

$$\mathbb{E}[S_d] = \sum_{i=1}^{3} i \cdot \binom{4}{i} p^i (1 - p)^{4-i} + 3p^4 = 4p - p^4,\qquad(8.8)$$

where in (8.8), *we get i innovative packets if there are i successful slots for* $1 \le i \le 3$. *But if all* 4 *slots are successful we get only* 3 *innovative packets. We can substitute* (8.7) *and* (8.8) *in* (8.6) *to get the* (τ, λ) *trade-off.*
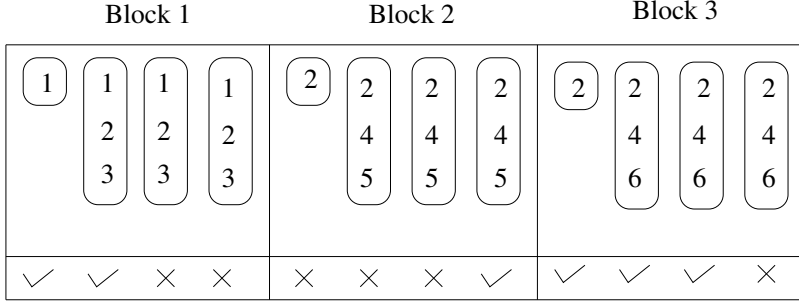
Figure 8-2: Illustration of the time-invariant scheme $\mathbf{x} = [1, 0, 3, 0]$ with block size $d = 4$. Each bubble represents a coded combination, and the numbers inside it are the indices of the source packets included in that combination. The check and cross marks denote successful and erased slots respectively. The packets that are "seen" in each block are not included in the coded packets in future blocks.

Remark 7. *Time-invariant schemes with different* $\mathbf{x}$ *can be equivalent in terms of the* (τ, λ). *In particular, given* $x_1 \geq 1$, *if any* $x_i = 0$, *and* $x_{i+1} = w \geq 1$, *then the scheme is equivalent to setting* $x_i = 1$ *and* $x_{i+1} = w - 1$, *keeping all other elements of* $\mathbf{x}$ *the same. This is because the number of independent linear combinations in the block, and the probability of decoding the first unseen is preserved by this transformation. For example,* $\mathbf{x} = [1, 1, 2, 0]$ *gives the same* (τ, λ) *as* $\mathbf{x} = [1, 0, 3, 0]$.

In Section 8.4 we saw that with immediate feedback, we can achieve $(\tau, \lambda) = (p, -\log(1 - p))$. However, with block-wise feedback we can achieve optimal τ (or λ) only at the cost of sacrificing the optimality of the other metric. We now find the best achievable τ (or λ) with optimal λ (or τ).

Claim 5 (Cost of Optimal Exponent λ). *With block-wise feedback after every d slots, and inter-delivery exponent* $\lambda = -\log(1 - p)$, *the best achievable throughput* $\tau = (1 - (1 - p)^d)/d$.

Proof. If we want to achieve $\lambda = -\log(1 - p)$, we require p_d in (8.6) to be equal to $1 - (1 - p)^d$. The only scheme that can achieve this is $\mathbf{x} = [d, 0, \cdots, 0]$, where we transmit d copies of the first unseen packet. The number of innovative packets S_d received in every block is 1 with probability $1 - (1 - p)^d$, and zero otherwise. Hence, the best achievable throughput is $\tau = (1 - (1 - p)^d)/d$ with optimal $\lambda = -\log(1 - p)$. $\quad\square$

This result gives us insight on how much bandwidth (which is proportional to

$1/\tau$) is needed for a highly delay-sensitive application that needs λ to be as large as possible.

Claim 6 (Cost of Optimal Throughput τ). *With block-wise feedback after every d slots, and throughput $\tau = p$, the best achievable inter-delivery exponent is $\lambda = -\log(1-p)/d$.*

Proof. If we want to achieve $\tau = p$, we need to guarantee an innovation packet in every successful slot. The only time invariant scheme that ensures this is $\mathbf{x} = [1, 0, \cdots, 0, d-1]$, or its equivalent vectors $\mathbf{x}$ as given by Remark 7. With $\mathbf{x} = [1, 0, \cdots, 0, d-1]$, the probability of decoding the first unseen packet is $p_d = p$. Substituting this in (8.6) we get $\lambda = -\log(1-p)/d$, the best achievable λ when $\tau = p$. $\square$

Tying back to Fig. 8-1, Claim 5 and Claim 6 correspond to moving leftwards and downwards along the dashed lines from the optimal trade-off $(p, -\log(1-p))$. From Claim 5 and Claim 6 we see that both τ and λ are $\Theta(1/d)$, keeping the other metric optimal.

Next we want to find the coding scheme that maximizes λ for any given throughput τ. We first prove that any convex combination of achievable points (τ, λ) can be achieved.

Lemma 16 (Combining of Time-invariant Schemes). *By randomizing between time-invariant schemes $\mathbf{x}^{(i)}$ for $1 \leq i \leq B$, we can achieve the throughput-smoothness trade-off given by any convex combination of the points $(\tau_{\mathbf{x}^{(i)}}, \lambda_{\mathbf{x}^{(i)}})$.*

The proof of Lemma 16 is deferred to Appendix F. The main implication of Lemma 16 is that, to find the best (τ, λ) trade-off, we only have to find the points $(\tau_{\mathbf{x}}, \lambda_{\mathbf{x}})$ that lie on the convex envelope of the achievable region spanned by all possible $\mathbf{x}$.

For general d, it is hard to search for the $(\tau_{\mathbf{x}}, \lambda_{\mathbf{x}})$ that lie on the optimal trade-off. We propose a set of time-invariant schemes that are easy to analyze and give a good (τ, λ) trade-off. In Theorem 13 we give the (τ, λ) trade-off for the proposed codes and show that for $d = 2$ and $d = 3$, it is the best trade-off among all time-invariant schemes.

Definition 26 (Proposed Codes for general d). *For general d, we propose using the time-invariant schemes with $x_1 = a$ and $x_{d-a+1} = d - a$, for $a = 1, \cdots d$.*

In other words, in every block of d slots, we transmit the first unseen packet a times, followed by $d - a$ combinations of the first $d - a + 1$ unseen packets. These schemes span the (τ, λ) trade-off as a varies from 1 to d, with a higher value of a corresponding to higher λ and lower τ. In particular, observe that the $a = d$ and $a = 1$ codes correspond to codes given in the proofs of Claim 5 and Claim 6.

Theorem 13 (Throughput-Smoothness Trade-off for General d). *The codes proposed in Definition 26 give the trade-off points*

$$(\tau, \lambda) = \left(\frac{1 - (1 - p)^a + (d - a)p}{d}, -\frac{a}{d} \log(1 - p) \right). \tag{8.9}$$

for $a = 1, \cdots d$.

Proof. To find the (τ, λ) trade-off points, we first evaluate $\mathbb{E}[S_d]$ and p_d. With probability $1 - (1 - p)^a$ we get 1 innovative packet from the first a slots in a block. The number of innovative packets received in the remaining $d - a$ slots is equal to the number of successful slots. Thus, the expected number of innovative coded packets received in the block is

$$\mathbb{E}[S_d] = 1 - (1 - p)^a + (d - a)p \tag{8.10}$$

If the first a slots in the block are erased, the first unseen packet cannot be decoded, even if all the other slots are successful. Hence, we have $p_d = 1 - (1-p)^a$. Substituting $\mathbb{E}[S_d]$ and p_d in (8.6), we get the trade-off in (8.9). $\qquad\square$

By Lemma 16, we can achieve any convex combination of the (τ, λ) points in (8.9). In Lemma 17 we show that for $d = 2$ and $d = 3$ this is the best trade-off among all time-invariant schemes.

Lemma 17. *For $d = 2$ and $d = 3$, the codes proposed in Definition 26 give the best (τ, λ) trade-off among all time-invariant schemes.*
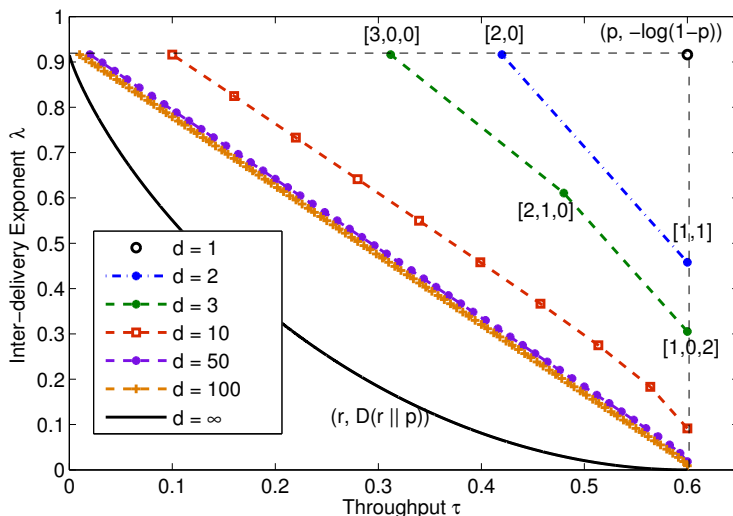
Figure 8-3: The throughput-smoothness trade-off of the suggested coding schemes in Definition 26 for $p = 0.6$ and various values of block-wise feedback delay d. The trade-off becomes significantly worse as d increases. The point labels on the $d = 2$ and $d = 3$ trade-offs are **x** vectors of the corresponding codes.

The proof is given in Appendix F. Fig. 8-3 shows the trade-off given by (8.9) for different values of d. We observe that the trade-off becomes significantly worse as d increases. Thus we can imply that frequent feedback to the source is important in delay-sensitive applications to ensure fast in-order delivery of packets. As $d \to \infty$, and $a = \alpha d$, the trade-off converges to $((1 - \alpha)p, -\alpha \log(1 - p))$ for $0 \le \alpha \le 1$, which is the line joining $(0, -\log(1 - p))$ and $(p, 0)$. It does not converge to the $(r, D(r\|p))$ curve without feedback because we consider that d goes to infinity slower than the n used to evaluate the asymptotic exponent of $\Pr(T_1 > n)$.

By Lemma 17 the proposed codes are optimal for $d = 2$ and $d = 3$ among all time-invariant schemes. Numerical results suggest that even for general d these schemes give a trade-off that is close to the best trade-off among all time-invariant schemes.

8.7 Concluding Remarks

In this chapter, we consider the problem of streaming over an erasure channel when the packets are required in-order, and investigate the trade-off between the throughput

and the smoothness of in-order packet delivery. Our analysis shows that frequent feedback about channel erasures drastically improves the smoothness, for the same throughput. We present a spectrum of coding schemes that span different points on the throughput-smoothness trade-off. Depending upon the delay-sensitivity and bandwidth limitations of the applications, one can choose a suitable operating point on this trade-off. In the next chapter we generalize to the multicast scenario, where the source wants to transmit the stream to multiple users over a shared channel. Chapter 10 we discuss other generalizations and future research directions.

Chapter 9

Multicast Streaming with Immediate Feedback

9.1 Introduction

In Chapter 8 we studied the effect of feedback delay on the throughput-smoothness trade-off in point-to-point streaming. When there are multiple users, in addition to the throughput-smoothness trade-off of each user, there is a trade-off between the different users depend upon how the source prioritizes them. Even the immediate feedback case becomes non-trivial. In this chapter we study this multicast scenario and gain understanding of how the source can strike a balance between giving priority to different users.

The use of network coding in multicast packet transmission has been studied in [107–112]. The authors in [107] use as a delay metric the number of coded packets that are successfully received, but do not allow immediate decoding of a source packet. For two users, the paper shows that a greedy coding scheme is throughput-optimal and guarantees immediate decoding in every slot. However, optimality of this scheme has not been proved for three or more users. In [108], the authors analyze decoding delay with the greedy coding scheme in the two user case. However, both these delay metrics do not capture the aspect of in-order packet delivery.

In-order packet delivery for multicast with immediate feedback is considered in

[109–111]. These works consider that packets are generated by a Poisson process and are greedily added to all future coded combinations. Another related work is [112] which also considers Poisson packet generation in a two-user multicast scenario and derives the stability condition for having finite delivery delay. However, in practice the source can use feedback about past erasures to decide which packets to add to the coded combinations, instead of just greedy coding over all generated packets.

In Section 9.2 we generalize the system model in Chapter 8 to the multicast scenario. In Section 9.3 we identify the structure of coding schemes required to simultaneously satisfy the requirements of multiple users. In Section 9.4 we use this structure to find the best coding scheme for the two user case, where one user is always given higher priority. In Section 9.5 we generalize this scheme to allow tuning the level of priority given to each user. The analysis of both these cases is based on a new Markov chain model of packet decoding. We propose coding schemes which allow us to tune the level of priority given to each user and hence achieve different points on its throughput-smoothness trade-off. The longer proofs are deferred to Appendix G.

9.2   System Model

The system model is a generalized version of that in Section 8.2. Instead of a single user, the source is transmitting a common packet stream $s_1, s_2, \cdots, s_n$ to K users $U_1, U_2, \cdots, U_K$. We consider an i.i.d. erasure channel to each user such that every transmitted packet is received successfully at user U_k with probability p_k, and otherwise received in error and discarded. The erasure events are independent across the users.

The throughput and smoothness metrics are same as in Section 8.3. We denote the throughput and smoothness exponent of user U_k by τ_k and λ_k respectively. In addition to the notions of packet decoding defined in Section 8.3 we define one additional notion of the 'required' packet of each user.

Definition 27 (Required packet). *The required packet of U_i is its earliest undecoded packet. Its index is denoted by r_i.*

In other words, s_{r_i} is the first unseen packet of user U_i. For example, if packets s_1, s_3 and s_4 have been decoded at user U_i, its required packet s_{r_i} is s_2.

Instead of considering block-wise feedback to the source, we focus on the immediate feedback ($d = 1$) case. This feedback can be used to estimate p_i for $1 \leq i \leq K$, the success probablities of the erasure channels, when they are unknown to the source. The case of general feedback delay d is hard to analyze and open for future work.

Remark 8. *For the no feedback case ($d = \infty$) we can extend Theorem 12 to show that the optimal throughput-smoothness trade-off for user U_k, $k = 1, 2, \cdots K$ among full-rank codes is $(\tau_k, \lambda_k) = (r, D(r\|p_k))$, if $0 \leq r \leq p_k$. If $r > p_k$ then $\lambda_k = 0$ for user U_k. Since we are transmitting a common stream, the rate r of adding new packets is same for all users.*

9.3 Structure of Coding Schemes

The best possible trade-off is $(\tau_i, \lambda_i) = (p_i, -\log(1 - p_i))$, and it can be achieved when there is only one user, and the source uses a simple Automatic-repeat-request (ARQ) protocol where it keeps retransmitting the earliest undecoded packet until that packet is decoded. In this chapter our objective is to design coding strategies to maximize τ and λ for the two user case. For two or more users we can show that it is impossible to achieve the optimal trade-off $(\tau, \lambda) = (p_i, -\log(1 - p_i))$ simultaneously for all users. We now present code structures that maximize throughput and inter-delivery exponent of the users.

Claim 7 (Include only Required Packets). *In a given slot, it is sufficient for the source to transmit a combination of packets s_{r_i} for $i \in \mathcal{I}$ where $\mathcal{I}$ is some subset of $\{1, 2, \cdots K\}$.*

Proof. Consider a candidate packet s_c where $c \neq r_i$ for any $1 \leq i \leq K$. If $c < r_i$ for all i, then s_c has been decoded by all users, and it need not be included in the combination. For all other values of c, there exists a required packet s_{r_i} for some $i \in \{1, 2, \cdots K\}$ which, if included instead of s_c, will allow more users to decode their

required packets. Hence, including that packet instead of s_c gives a higher smoothness exponent λ. $\qquad\square$

Claim 8 (Include only Decodable Packets). *If a coded combination already includes packets s_{r_i} with $i \in \mathcal{I}$, and U_j, $j \notin I$ has not decoded all s_{r_i} for $i \in \mathcal{I}$, then a scheme that does not include s_{r_j} in the combination gives a better throughput-smoothness trade-off than a scheme that does.*

Proof. If U_j has not decoded all s_{r_i} for $i \in \mathcal{I}$, the combination is innovative but does not help decoding an in-order packet, irrespective of whether s_{r_j} is included in the combination. However, if we do not include packet s_{r_j}, U_j may be able to decode one of the packets s_{r_i}, $i \in \mathcal{I}$, which can save it from out-of-order packet decoding in a future slot. Hence excluding s_{r_j} gives a better throughput-smoothness trade-off. $\qquad\square$

Example 2. *Suppose we have three users U_1, U_2, and U_3. User U_1 has decoded packets s_1, s_2, s_3 and s_5, user U_2 has decoded s_1, s_3, and s_4, and user U_3 has decoded s_1, s_2, and s_5. The required packets of the three users are s_4, s_2 and s_3 respectively. By Claim 7, the optimal scheme should transmit a linear combination of one or more of these packets. Suppose we construct combination of s_4 and s_2 and want to decide whether to include s_3 or not. Since user U_3 has not decoded s_4, we should not include s_3 as implied by Claim 8.*

The choice of the initial packets in the combination is governed by a priority given to each user in that slot. Claims 7 and 8 imply the following code structure for the two user case.

Proposition 1 (Code Structure for the Two User Case). *Every achievable trade-off between throughput and inter-delivery exponent can be obtained by a coding scheme where the source transmits s_{r_1}, s_{r_2} or the exclusive-or, $s_{r_1} \oplus s_{r_2}$ in each slot. It transmits $s_{r_1} \oplus s_{r_2}$ if and only if $r_1 \neq r_2$, and U_1 has decoded s_{r_2} or U_2 has decoded s_{r_1}.*

In the rest of this chapter we analyze the two user case and focus on coding schemes as given by Proposition 1.

Time	Sent	U_1	U_2
1	s_1	s_1	✗
2	s_2	✗	s_2
3	$s_1 \oplus s_2$	s_2	s_1
4	s_3	s_3	✗
5	s_4	s_4	s_4

Figure 9-1: Illustration of the optimal coding scheme when the source always give priority to user U_1. The third and fourth columns show the packets decoded at the two users. Cross marks indicate erased slots for the corresponding user.

9.4 Optimal Performance for One of Two Users

In this section we consider that the source always gives priority to one user, called the primary user. We determine the best achievable throughput-smoothness trade-off for a secondary user that is "piggybacking" on such a primary user. For simplicity of notation, let $a \triangleq p_1 p_2$, $b \triangleq p_1(1 - p_2)$, $c \triangleq (1 - p_1)p_2$ and $d \triangleq (1 - p_1)(1 - p_2)$, the probabilities of the four possible erasure patterns.

Without loss of generality, suppose that U_1 is the primary user, and U_2 is the secondary user. Recall that ensuring optimal performance for U_1 implies achieving $(\tau_1, \lambda_1) = (p_1, -\log(1 - p_1))$. While ensuring this, the best throughput-smoothness trade-off for user U_2 is achieved by the coding scheme given by Claim 9 below.

Claim 9 (Optimal Coding Scheme). *A coding scheme where the source transmits $s_{r_1} \oplus s_{r_2}$ if U_2 has already decoded s_{r_1}, and otherwise transmits s_{r_1}, gives the best achievable (τ_2, λ_2) trade-off while ensuring optimal (τ_1, λ_1).*

Proof. Since U_1 is the primary user, the source must include its required packet s_{r_1} in every coded combination. By Proposition 1, if the source transmits $s_{r_1} \oplus s_{r_2}$ if U_2 has already decoded s_{r_1}, and transmits s_{r_1} otherwise, we get the best achievable throughput-smoothness trade-off for U_2. $\square$

Fig. 9-1 illustrates this scheme for one channel realization.

Packet decoding at the two users with the scheme given by Claim 9 can be modeled by the Markov chain shown in Fig. 9-2. The state index i can be expressed in terms of the number of gaps in decoding of the users, defined as follows.
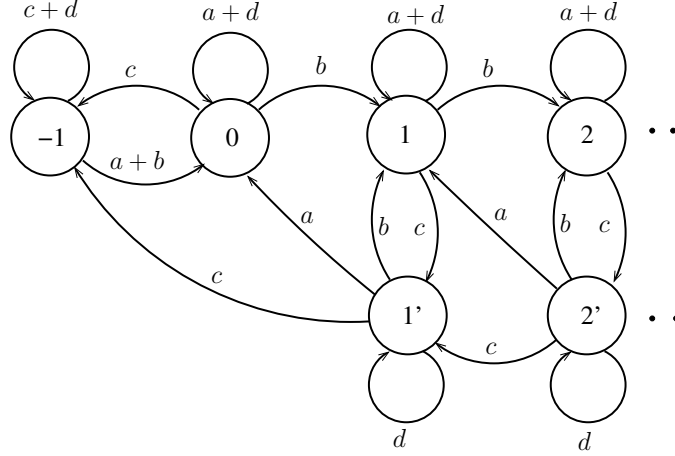
Figure 9-2: Markov chain model of packet decoding with the coding scheme given by Claim 9, where U_1 is the primary user. The state index i represents the number of gaps in decoding of U_2 minus that for U_1. The states i' are the advantage states where U_2 gets a chance to decode its required packet.

Definition 28 (Number of Gaps in Decoding). *The number of gaps in U_i's decoding is the number of undecoded packets of U_i with indices less than $r_{\max} = \max_i r_i$.*

In other words, the number of gaps is the amount by which a user U_i lags behind the user that is leading the in-order packet decoding. The state index i, for $i \geq -1$ is equal to the number of gaps in decoding at U_2, minus that for U_1. Since the source gives priority to U_1, it always has zero gaps in decoding, except when there is a $c = p_2(1 - p_1)$ probability erasure in state 0, which causes the system goes to state -1. The states i' for $i \geq 1$ are called "advantage" states and are defined as follows.

Definition 29 (Advantage State). *The system is in an advantage state when $r_1 \neq r_2$, and U_2 has decoded s_{r_1} but U_1 has not.*

By Claim 9, the source transmits $s_{r_1} \oplus s_{r_2}$ when the system is in an advantage state i', and it transmits s_{r_1} when the system is in state i for $i \geq -1$. We now describe the state transitions of this Markov chain. First observe that with probability $d = (1 - p_1)(1 - p_2)$, both users experience erasures and the system transitions from any state to itself. When the system is in state -1, the source transmits s_{r_1}. Since s_{r_1} has been already decoded by U_2, the probability $c = p_2(1 - p_1)$ erasure also keeps the system in the same state. If the channel is successful for U_1, which occurs with

146

probability $p_1 = a + b$, it fills its decoding gap and the system goes to state 0.

The source transmits s_{r_1} in any state i, $i \geq 1$. With probability $a = p_1 p_2$, both users decode s_{r_1}, and hence the state index i remains the same. With probability $b = p_1(1 - p_2)$, U_1 receives s_{r_1} but U_2 does not, causing a transition to state $i + 1$. With probability $c = (1 - p_1)p_2$, U_2 receives s_{r_1} and U_1 experiences an erasure due to which the system moves to the advantage state i'. When the system is an advantage state, having decoded s_{r_1} gives U_2 an advantage because it can use $s_{r_1} \oplus s_{r_2}$ transmitted in the next slot to decode s_{r_2}. From state i', with probability a, U_1 decodes s_{r_1} and U_2 decodes s_{r_2}, and the state transitions to $i - 1$. With probability c, U_2 decodes s_{r_2}, but U_1 does not decode s_{r_1}. Thus, the system goes to state $(i - 1)'$, except when $i = 1$, where it goes to state 0.

Claim 10. *The Markov chain in Fig. 9-2 is positive-recurrent and has unique steady-state distribution if and only if $b < c$, which is equivalent to $p_1 < p_2$.*

Lemma 18 (Trade-off for the Piggybacking user). *When the source always gives priority to user U_1 it achieves the optimal trade-off $(\tau_1, \lambda_1) = (p_1, -\log(1 - p_1))$. The scheme in Claim 9 gives the best achievable (τ_2, λ_2) trade-off for piggybacking user U_2. The throughput τ_2 is given by*

$$\tau_2 = p_1 \quad \text{if } p_2 > p_1. \tag{9.1}$$

If $p_2 \leq p_1$, τ_2 cannot be evaluated using our Markov chain analysis. The inter-delivery exponent of U_2 for any p_1 and p_2 is given by

$$\lambda_2 = -\log\left(\max\left(\frac{1 - c + d + \sqrt{(1 - c + d)^2 + 4(bc + cd - d)}}{2}, 1 - p_1\right)\right). \tag{9.2}$$

In Fig. 9-3 we plot the inter-delivery exponent λ_2 versus p_2, which increases from p_1 to 1 along each curve. The inter-delivery exponent λ_2 increases with p_2, but saturates at $-\log(1 - p_1)$, the inter-delivery exponent λ_1 of the primary user. Since U_2 is the
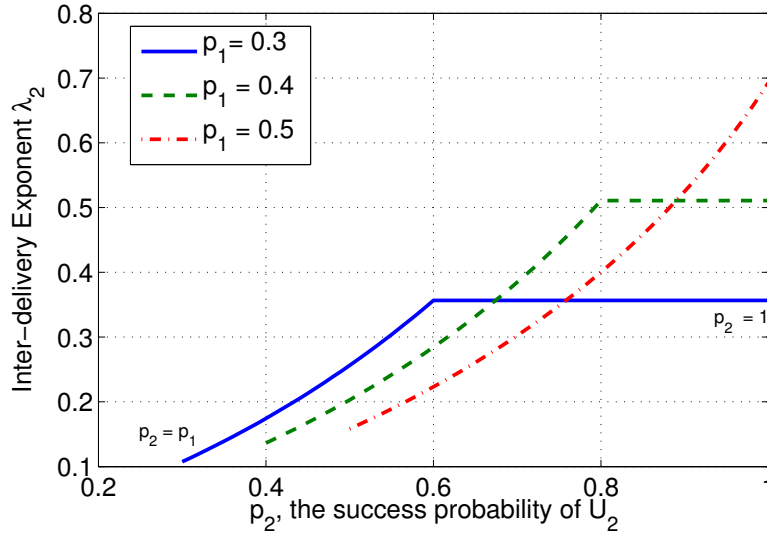
Figure 9-3: Plots of the inter-delivery exponent λ_2 of the piggybacking user U_2, versus the success probability p_2 throughput τ_2. The value of p_2 varies from p_1 to 1 on each curve. The exponent saturates at $-\log(1 - p_1)$, which is equal to λ_1, the exponent of the primary user U_1.

secondary user it cannot achieve faster in-order delivery than the primary user U_1.

9.5 General Throughput-Smoothness Trade-offs

For the general case, we propose coding schemes that can be combined to tune the priority given to each of the two users and achieve different points on their throughput-smoothness trade-offs.

Let $r_{\max} = \max(r_1, r_2)$ and $r_{\min} = \min(r_1, r_2)$, where r_1 and r_2 are the indices of the required packets of the two users. We refer to the user with the higher index r_i as the leader(s) and the other user as the lagger. Thus, U_1 is the leader and U_2 is the lagger when $r_1 > r_2$. If $r_1 = r_2$, without loss of generality we consider U_1 as the leader.

Definition 30 (Priority-(q_1, q_2) Codes)**.** *If the lagger U_i has not decoded packet $s_{r_{\max}}$, the source transmits $s_{r_{\min}}$ with probability q_i and $s_{r_{\max}}$ otherwise. If the lagger has decoded $s_{r_{\max}}$, the source transmits $s_{r_{\max}} \oplus s_{r_{\min}}$.*

Note that the code given in Claim 9, where the source always gives priority to user
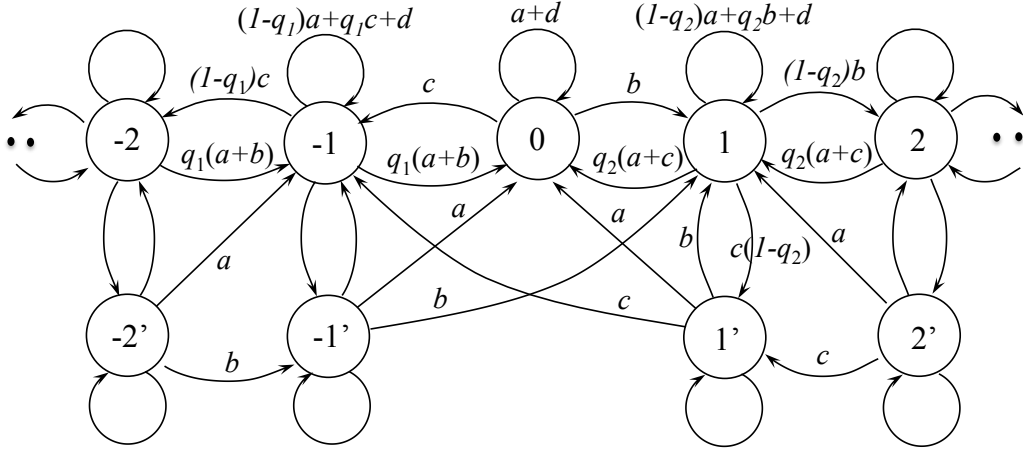
Figure 9-4: Markov chain model of packet decoding with the priority-(q_1, q_2) coding scheme given by Definition 30. The state index i represents the number of gaps in decoding if U_2 compared to U_1 and q_i is the probability of giving priority to the U_i when it is the lagger, by transmitting its required packet s_{r_i}. and

U_1 is a special case of priority-(q_1, q_2) codes with $(q_1, q_2) = (1, 0)$. Another special case is $(q_1, q_2) = (0, 0)$ which is a greedy coding scheme that always favors the user which is ahead in in-order delivery. The greedy coding scheme ensures throughput optimality to both users, i.e. $\tau_1 = p_1$ and $\tau_2 = p_2$.

Remark 9. *A generalization of priority-(q_1, q_2) codes is to consider priorities $q_1^{(i)}$ and $q_2^{(i)}$ that depend on the state i of the Markov chain. A special case of this is $q_1^{(i)} = 1$ for all states $i \geq -M$, and $q_2^{(i)} = 1$ for all states $j \leq N$ for integers $M, N > 0$. This scheme corresponds to putting hard boundaries on both sides of the Markov chain. It was analyzed in [24].*

The Markov model of packet decoding with a priority-(q_1, q_2) code is as shown in Fig. 9-4, which is a two-sided version of the Markov chain in Fig. 9-2. Same as in Fig. 9-2, the index i of a state i of the Markov chain is the number of gaps in decoding of U_2 minus that for U_1. User U_1 is the leader when the system is in state $i \geq 1$ and U_2 is the leader when $i \leq -1$, and both users are leaders when $i = 0$. The system is in the advantage state i' if packet is decoded by the lagger but not the leader. For simplicity of representation we define the notation $\bar{d} \triangleq 1 - d$, $\bar{q}_1 \triangleq 1 - q_1$ and $\bar{q}_2 \triangleq 1 - q_2$.
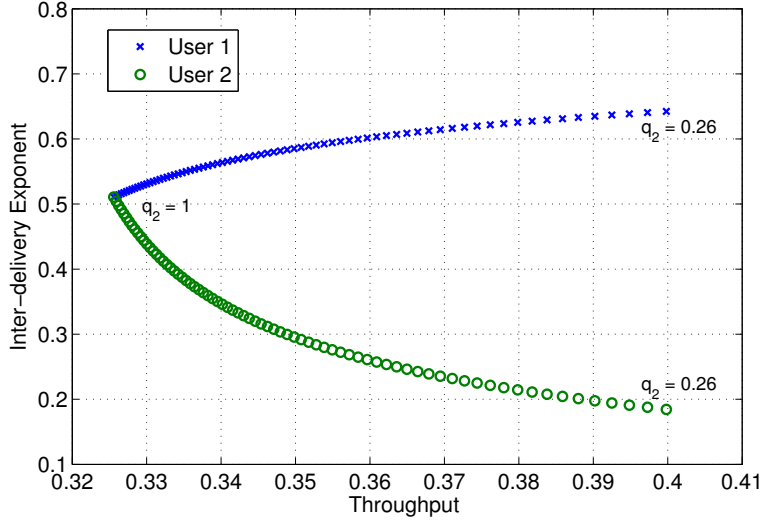
Figure 9-5: Plot of the throughput-smoothness trade-off for $q_1 = 1$ and as q_2 varies. The success probabilities $p_1 = 0.5$ and $p_2 = 0.4$.

Lemma 19 ((τ, λ) Trade-offs with Priority-(q_1, q_2) codes). *Let* $\mu = \pi_{i-1}/\pi_i$ *for* $i \leq -1$. *Then the priority-(q_1, q_2) codes given by Definition 30 give the following throughput for* U_2.

$$\tau_2 = p_2 \left(1 - \frac{q_1 \pi_{-1}}{1 - \mu} \right) \quad \text{if } \mu < 1 \tag{9.3}$$

If $\mu > 1$ *then* τ_2 *cannot be evaluated using our Markov chain analysis. On the other hand, the inter-delivery exponent can be evaluated for any* μ *as given by*

$$\lambda_2 = -\log \max \left(d + q_1 c + \bar{q}_1 b, \right.$$
$$\left. \frac{\left(2d + \bar{q}_2 a + b + \sqrt{(2d + \bar{q}_2 a + b)^2 - 4(d(b + d) + \bar{q}_2(da - bc))} \right)}{2} \right) \tag{9.4}$$

Similarly, let $\rho = \pi_{i+1}/\pi_i$ *for* $i \geq 1$. *If* $\rho < 1$, *the expressions for throughput* τ_1 *and inter-delivery exponent* λ_1 *of* U_1 *are same as (9.3) and (9.4) with* b *and* c, *and* q_1 *and* q_2 *interchanged,* π_{-1} *replaced by* π_1, *and* μ *replaced by* ρ.

Fig. 9-5 shows the throughput-smoothness trade-offs of the two users as q_2 varies,
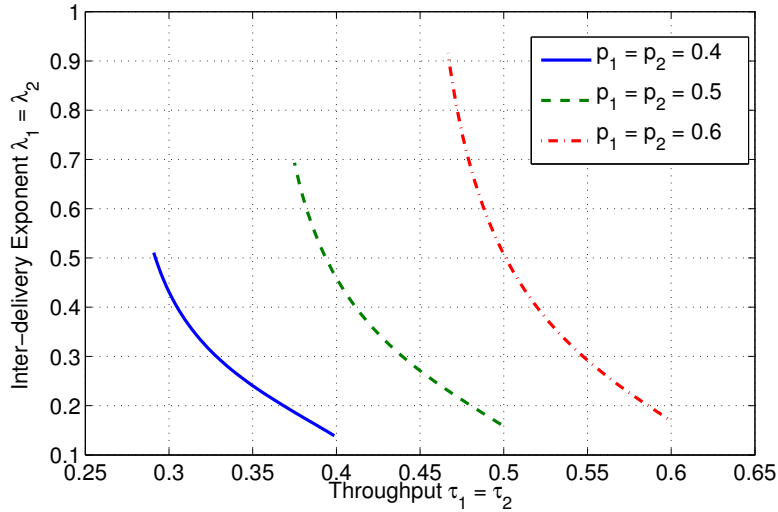
Figure 9-6: Plot of the throughput-smoothness trade-off for different values $p_1 = p_2$. On each curve, $q_1 = q_2$ varies from 0 from 1.

when $q_1 = 1$, $p_1 = 0.5$ and $p_2 = 0.4$. To stabilize the right-side of the Markov chain in Fig. 9-4 for these parameters we require q_2 to be at least 0.25. As q_2 increases from 0.26 to 1 in Fig. 9-5 we observe that U_2 gains in smoothness, at the cost of the smoothness of U_1. Also, both users lose throughput when q_2 increases.

In Fig. 9-6 we show the effect of increasing q_1 and q_2 simultaneously for different values of $p_1 = p_2$. As $q_1 = q_2$ increases, we get a better inter-delivery exponent for both users, but at the cost of loss of throughput.

9.6 Concluding Remarks

In this chapter we consider the problem where multiple users request a packet stream from the source over a shared channel with immediate feedback. While different users decode different sets of packets, the source has to simultaneously ensure smooth decoding at all of them. We study the inter-dependence of the throughput-smoothness trade-offs for the two user case, and develop coding schemes to tune the priority given to each user. A future research direction is to generalize the analysis beyond two users, and also consider the effect of delayed feedback about erasures.

Chapter 10

Future Directions

In Chapter 8 and Chapter 9 we studied the interplay between throughput and smoothness of streaming communication, and developed erasure codes that achieve a good trade-off. In this chapter we discuss generalizations of the system model and future research directions.

10.1   Model Generalizations

10.1.1   Finite Buffer

There can be two types of buffers at the receiver: the decoding buffer and the playback buffer. The decoding buffer collects coded combinations received over the erasure channel. Once enough combinations are received, the packets are decoded and sent to the playback buffer. The playback buffer is drained at a constant rate at which packets are played. Interruptions in playback occur when the playback buffer becomes empty. In Chapter 8 and Chapter 9 we do not have a playback buffer: bursts of packets are delivered to the application as soon as they are decoded in order. A playback buffer is considered in [104, 105, 113]. Our analysis on the growth of playback delay gives insights into the required size of playback buffer.

Interesting future directions include considering one or both of these buffers being finite in size. In [113] we considered the case of a finite playback buffer, where we

proposed a dynamic bandwidth scheme called the buffer refill scheme. However, determining the optimal packet transmission scheme is open for future work.

10.1.2 Dynamic Bandwidth

Instead of fixed bandwidth b packets per slot, if the bandwidth can be dynamically adjusted, we can potentially better throughput-smoothness performance. Dynamic bandwidth allows the source to better adapt future transmissions to feedback about erasures. For example, when a large number of consecutive packets are erased, the source can use more bandwidth to fill those gaps in decoding. Moreover, if the channel has memory such that erasures occur in bursts, dynamic bandwidth can be used to develop a water-pouring [2] type transmission strategy. Dynamic bandwidth was also considered in [103] for uncoded streaming transmission. Analyzing and developing erasure codes with dynamic bandwidth is open for future research.

10.1.3 Packet Dropping

In audio or video streaming, there is significant correlation between subsequent packets. Thus the quality perceived by the user remains unaffected if some packets are dropped. If packet dropping is permitted, then the source can use the limited available bandwidth to transmit new packets instead of attempting to fill older gaps in decoding. Thus, careful packet dropping can result in higher throughput for the same smoothness perceived by the user. Choice of which packets to drop such that the quality of streaming is maintained is an interesting future direction.

10.1.4 Streaming from Distributed Sources

In practical systems often there are multiple sources, for example caches in a peer-to-peer network, that store the streaming data fully or in parts. The sources may have different bandwidth limitations, and erasure probabilities while transmitting coded packets to the user in each slot. Using the diversity of sources we can improve the delay in decoding each individual packet. The throughput improvement of such coded

caching strategies has been explored in a line of recent works [114–116]. However the analysis of smoothness of packet delivery from such distributed systems remains unexplored. We aim extend the work on point-to-point streaming to consider how to best store and combine the information received from each of the sources.

10.2 Alternate Smoothness Metrics

To capture the smoothness of packet delivery to the users, we considered smoothness exponent λ, defined as the exponent of the in-order decoding delay D_k. We found it particular tricky to come up with the best suited delay metric to evaluate the quality of streaming. We now discuss some alternate metrics that are also meaningful for streaming applications. In some regimes these metrics can be expressed in terms of the smoothness exponent λ.

10.2.1 Playback Delay

When packets are being generated in real-time at the source and played back at a constant rate at the receiver, playback delay (defined as the number of interrupted slots) is a good metric of the streaming quality. The authors in [103] analyze the playback delay of real-time streaming for uncoded packet transmission over a channel with long feedback delay.

In [104, 105] we show that the number of interruptions in playback scales $\Theta(\log n)$ for a stream of length n. We then proposed codes that minimize the pre-log term for the no feedback and immediate feedback cases. For the no feedback case, the playback delay grows as $(1/\lambda) \cdot \log n$, where $\lambda = D(r\|p)$, the smoothness exponent. Recent work [113] shows that if the stream is available beforehand (non real-time), it is possible to guarantee constant number of interruptions, independent of n.

When the feedback to the source is delayed by d slots, the analysis of playback delay becomes complicated, and is open for future research. In [105] we present some preliminary insights on this problem. We express the playback delay in terms of a random walk between two boundaries at a distance d from each other.

10.2.2 In-order Delivery Delay

For applications such as Dropbox and remote desktop, the packets are required in-order, but are not played back at a constant rate. For example, a burst of in-order instructions delivered to a remote desktop can be executed almost instantaneously. Instead of playback delay, in-order delivery delay D_k is a good metric for such applications. In-order delivery delay is analyzed in [14] for the no feedback case. In [13] the authors present bounds on the mean and variance of the in-order delivery delay when the source receives delayed feedback about packet erasures.

In Chapter 8 and Chapter 9 we consider the exponent of the in-order delivery delay D_k as the smoothness metric. Instead, we can also analyze the expected value of in-order delivery delay. We refer to this as the 'smoothness index' and analyze it for multicast streaming with immediate feedback in [24].

10.2.3 Probability of Interruption

In [117] the authors consider the probability of interruption in playback as the delay metric. They study the trade-off between the throughput and the probability of interruption due to buffer underflow. For our system model, the probability of buffer underflow is same the probability of the inter-delivery delay T exceeding the size of the buffer. Thus the probability of interruption is proportional to $e^{-\lambda B}$, where B is the playback buffer size and λ is the smoothness exponent.

Chapter 11

Concluding Remarks

11.1 Summary

The cloud provides flexible and scalable access to computing via thousands on shared servers in data centers. Fast and fluid response to users is one of the key metrics of performance. With applications becoming more interactive, we as users are becoming sensitive to even sub-second delays in response. Guaranteeing such low delays is challenging because service times can vary due to factors such as virtualization, outages, packet loss etc. And it is difficult to set up centralized monitoring of the servers to accurately estimate their state. In this thesis we use *redundancy* to reduce delay in a stateless and cost-efficient manner.

In Part I we study the simple idea of replicating computing tasks and waiting for the earliest to finish. Task replication can be a powerful and effective way to speed-up computing, without requiring constant monitoring of the servers. However, it can cost additional computing resources and increase queueing load at the servers. In Chapter 2 we presented a framework to analyze queues with redundancy. This analysis uncovered the surprising insight that for *log-convex* service distributions, replication not only reduces latency but also makes the system more efficient. Chapter 3 explored straggler replication strategies for computing jobs with many parallel tasks.

In Part II we generalized the task replication setup and analyze content download from erasure coded storage. If a content is erasure coded with an (n, k) maximum-

distance-separable code, than any k out of n chunks are sufficient to recover it. Thus, the delay in content access can be reduced by requesting all n chunks and waiting for k. This idea led to the formulation and analysis of the (n, k) fork-join queueing system and its variants. We can show that coding can significantly reduce latency, without much computing and storage overhead.

Unlike traditional content download where only total delay matters, streaming requires the packets of the content to be delivered fast and in-order. In Part III we developed erasure codes that combine old and new packets effectively to ensure smooth playback with limited available bandwidth.

A unified view of the three parts of this thesis is that we study techniques to combat variability in the system and reduce delay, subject to the order and resource constraints. Part I and Part II focus on the overall latency, without order constraints tasks executed or the content downloaded. Part III accounts for these order constraints by considering in-order packet decoding. There are several unexplored intersections of these three areas. For instance, order constraints on tasks are common in cloud computing. In MapReduce the *map* tasks need to executed before *reduce* tasks. Optimal scheduling of tasks on cloud servers by accounting for these dependencies between tasks is a possible future direction. There are also interesting problems at the intersection on Part II and Part III. A complementary solution to our work on streaming codes is to access content at multiple caches to improve streaming quality.

11.2   Broader Future Directions

More broadly, it is well-known that the amount of data in the cloud is growing at an alarming rate. Thus there is an urgent need to build infrastructure to store and process this data in a fast, efficient and reliable fashion. This thesis develops a novel mathematical framework to understand how redundancy can help combat random service delays. Armed with this framework, we can approach problems of stochastic variability in a wide range of applications. Some areas of particular interest are described below.

- **Large-scale Machine Learning:** Machine learning algorithms constitute a large fraction of the computing performed in the cloud today. These algorithms require training of large neural networks using millions of training samples. For speed and scalability, training can be parallelized by creating model replicas that communicate with a central parameter server [73]. Such distributed learning frameworks suffer from service variability and stragglers, and can benefit from strategies similar to our work in Part I.

- **Crowdsourcing:** Humans are increasing becoming part of the cloud infrastructure. Tasks that require human intelligence can be assigned to workers via platforms like Amazon Mechanical Turk. Although it is mostly used for small tasks (e.g. image annotation) today, crowdsourcing has great potential in transforming the way humans accomplish large projects. Redundancy techniques based on this thesis can be applied to reduce the variability in waiting for workers to attempt and complete tasks.

- **Noisy Logic Devices**: The building blocks of all cloud infrastructure are logic gates and memory devices. Some emerging devices such as spintronic gates have a curious accuracy-speed trade-off; their accuracy improves if we allow them more time to settle. We are interested in developing techniques to guide circuit design that strikes a balance between accuracy and speed.

- **Energy Systems:** In the electricity grid, matching the supply and demand is crucial. This matching is becoming harder to achieve due to the increased integration of stochastically varying renewable sources. At the same time, data from smart meters provides novel opportunities to shift demand towards surplus supply. The probabilistic modeling and analysis techniques used in this thesis can be applied to construct data-driven scheduling to improve energy efficiency.

11.3 Final Remarks

In this thesis we explored the use of redundancy to reduce latency in cloud systems. Redundancy in the form of repetition, and more generally coding has been used in communication, network routing etc. for several decades. However its use was largely unexplored in the context of scheduling in cloud systems. This thesis presents a theoretical framework to study queueing systems with redundancy, and reduce delay in a cost-efficient manner.

The central idea behind this work is to build a reliable system from a set of unreliable and unpredictable components. From the diversity in the future directions discussed above, it is apparent that imperfect components arise in almost every practical system. And redundancy can help deal with those imperfections. We are curious to explore many more applications of this simple, yet powerful idea.

Appendix A

Properties of Log-concavity

In this section we present some properties and examples of log-concave and log-convex random variables that are relevant to this work. For more properties please see [52].

Property 1 (Jensen's Inequality)**.** *If $\bar{F}_X$ is log-concave, then for $0 < \theta < 1$ and for all $x, y \in [0, \infty)$,*

$$\Pr(X > \theta x + (1 - \theta)y) \geq \Pr(X > x)^\theta \Pr(X > y)^{1-\theta}. \tag{A.1}$$

The inequality is reversed if $\bar{F}_X$ is log-convex.

Proof. Since $\bar{F}_X$ is log-concave, $\log \bar{F}_X$ is concave. Taking log on both sides on (A.1) we get the Jensen's inequality which holds for concave functions. □

In past literature saying X is log-concave usually means that f is log-concave. This implies that F and $\bar{F}$. However log-convex f, does not always imply log-convexity of F and $\bar{F}$.

Property 2 (Scaling)**.** *If $\bar{F}_X$ is log-concave, for $0 < \theta < 1$,*

$$\Pr(X > x) \leq \Pr(X > \theta x)^{1/\theta} \tag{A.2}$$

The inequality is reversed if $\bar{F}_X$ is log-convex.

Proof. We can derive (A.2) by setting $y = 0$ in (A.1).

$$\Pr(X > \theta x + (1 - \theta)0) \geq \Pr(X > x)^\theta \Pr(X > 0)^{1-\theta}, \tag{A.3}$$

$$\Pr(X > \theta x) \geq \Pr(X > x)^\theta. \tag{A.4}$$

To get (A.4) we observe that if $\bar{F}_X$ is log-concave, then $\Pr(X > 0)$ has to be 1. Otherwise log-concavity is violated at $x = 0$. Raising both sides of (A.4) to power $1/\theta$ we get (A.2). The reverse inequality of log-convex $\bar{F}_X$ can be proved similarly. □

Property 3 (Sub-multiplicativity). *If $\bar{F}_X$ is log-concave, the conditional tail probability of X satisfies for all $t, x > 0$,*

$$\Pr(X > x + t | X > t) \leq \Pr(X > x) \tag{A.5}$$

$$\Leftrightarrow \Pr(X > x + t) \leq \Pr(X > x) \Pr(X > t) \tag{A.6}$$

The inequalities above are reversed if $\bar{F}_X$ is log-convex.

Proof.

$$\Pr(X > x) \Pr(X > t) \tag{A.7}$$

$$= \Pr\left(X > \frac{x}{x+t}(x+t)\right) \Pr\left(X > \frac{t}{x+t}(x+t)\right), \tag{A.8}$$

$$\geq \Pr(X > x + t)^{\frac{x}{x+t}} \Pr(X > x + t)^{\frac{t}{x+t}}, \tag{A.9}$$

where we apply Property 2 to (A.8) to get (A.9). Equation (A.5) follows from (A.9).

□

Note that for exponential F_X which is memoryless, (A.5) holds with equality. Thus log-concave distributions can be thought to have 'optimistic memory', because the conditional tail probability decreases over time. On the other hand, log-convex distributions have 'pessimistic memory' because the conditional tail probability increases over time. The definition of the notions 'new-better-than-used' in [46] is same as (A.5). By Property 3 log-concavity of $\bar{F}_X$ implies that X is new-better-than-used.

New-better-than-used distributions are referred to as 'light-everywhere' in [48] and 'new-longer-than-used' in [49].

Property 4 (Mean Residual Life). *If $\bar{F}_X$ is log-concave (log-convex), $\mathbb{E}[X - t|X > t]$, the mean residual life after time $t > 0$ has elapsed is non-increasing (non-decreasing) in t.*

of Lemma 1. Lemma 1 is true for log-concave $\bar{F}_X$ if $r\mathbb{E}[X_{1:r}] \leq (r + 1)\mathbb{E}[X_{1:r+1}]$ for all integers $r \geq 1$. This inequality can be simplified as follows.

$$r\mathbb{E}[X_{1:r}] \leq (r + 1)\mathbb{E}[X_{1:r+1}] \tag{A.10}$$

$$\Leftrightarrow r \int_0^\infty \Pr(X_{1:r} > x)dx \leq \int_0^\infty (r + 1)\Pr(X_{1:r+1} > x)dx, \tag{A.11}$$

$$\Leftrightarrow r \int_0^\infty \Pr(X > x)^r dx \leq \int_0^\infty (r + 1)\Pr(X > x)^{r+1}dx, \tag{A.12}$$

$$\Leftrightarrow \int_0^\infty \Pr\left(X > \frac{x'}{r}\right)^r dx' \leq \int_0^\infty \Pr\left(X > \frac{x'}{r + 1}\right)^{r+1} dx', \tag{A.13}$$

We get (A.11) using the fact that the expected value of a non-negative random variable is equal to the integral of its tail distribution. To get (A.12) observe that since $X_{1:r} = \min(X_1, X_2, \cdots, X_r)$ for i.i.d. X_i, we have $\Pr(X_{1:r} > x) = \Pr(X > x)^r$ for all $x > 0$. Similarly $\Pr(X_{1:r+1} > x) = \Pr(X > x)^{r+1}$. Next we perform a change of variables on both sides of (A.12) to get (A.13).

Now we use Property 2 to compare the two integrands in (A.13). Setting $\theta = r/r + 1$ and $x = x'/r$ in Property 2, we get

$$\Pr\left(X > \frac{x'}{r}\right)^r \leq \Pr\left(X > \frac{x'}{r + 1}\right)^{r+1} \qquad \text{for all } x' \geq 0. \tag{A.14}$$

Hence, by (A.14) and the equivalences in (A.10)-(A.13) it follows that for log-concave $\bar{F}_X$ if $r\mathbb{E}[X_{1:r}]$ is non-decreasing in r. For log-convex $\bar{F}_X$, we can show that $r\mathbb{E}[X_{1:r}]$ is non-increasing in r by reversing all inequalities above.

$\square$

Remark 10. *If X is new-better-than-used (a weaker notion implied by log-concavity*

of X), it can be shown that

$$\mathbb{E}[X] \leq r\mathbb{E}[X_{1:r}] \text{ for all integers } r \geq 1 \qquad (A.15)$$

This is weaker than Lemma 1 which shows the monotonicity of $r\mathbb{E}[X_{1:r}]$ for log-concave (log-convex) X.

Property 5 (Hazard Rates)**.** *If $\bar{F}_X$ is log-concave (log-convex), then the hazard rate $h(x)$, which is defined by $-\bar{F}'_X(x)/\bar{F}_X(x)$, is non-decreasing (non-increasing) in x.*

Property 6 (Coefficient of Variation)**.** *The coefficient of variation $C_v = \sigma/\mu$ is the ratio of the standard deviation σ and mean μ of random variable X. For log-concave (log-convex) X, $C_v \leq 1$ ($C_v \geq 1$), and $C_v = 1$ when X is pure exponential.*

Property 7 (Examples of Log-concave $\bar{F}_X$)**.** *The following random variables have log-concave $\bar{F}_X$:*

- *Shifted Exponential (Exponential plus constant $\Delta > 0$)*

- *Uniform over any convex set*

- *Weibull with shape parameter $c \geq 1$*

- *Gamma with shape parameter $c \geq 1$*

- *Chi-squared with degrees of freedom $c \geq 2$*

Property 8 (Examples of Log-convex $\bar{F}_X$)**.** *The following random variables have log-convex $\bar{F}_X$:*

- *Exponential*

- *Hyper Exponential (Mixture of exponentials)*

- *Weibull with shape parameter $0 < c < 1$*

- *Gamma with shape parameter $0 < c < 1$*

Appendix B

Proof of Theorem 3

Proof of Theorem 3. Using (2.4), we can express the cost C in terms of the relative task start times t_i, and S as follows. Since only r tasks are invoked, the relative start times $t_{r+1}, \ldots, t_n$ are equal to ∞.

$$C = S + (S - t_2)^+ + \cdots + (S - t_r)^+ , \tag{B.1}$$

where S is the time between the start of service of the earliest task, and when any 1 of the r tasks finishes. The tail distribution of S is given by

$$\Pr(S > s) = \prod_{i=1}^{r} \Pr(X > s - t_i). \tag{B.2}$$

By taking expectation on both sides of (B.1) and simplifying we get,

$$\mathbb{E}\left[C\right] = \sum_{u=1}^{r} \int_{t_u}^{\infty} \Pr(S > s) ds, \tag{B.3}$$

$$= \sum_{u=1}^{r} u \int_{t_u}^{t_{u+1}} \Pr(S > s) ds, \tag{B.4}$$

$$= \sum_{u=1}^{r} u \int_{0}^{t_{u+1} - t_u} \Pr(S > t_u + x) dx, \tag{B.5}$$

$$= \sum_{u=1}^{r} u \int_{0}^{t_{u+1} - t_u} \prod_{i=1}^{u} \Pr(X > x + t_u - t_i) dx. \tag{B.6}$$

We now prove that for log-concave $\bar{F}_X$, $\mathbb{E}[C] \geq \mathbb{E}[X]$. The proof that $\mathbb{E}[C] \leq \mathbb{E}[X]$ when $\bar{F}_X$ is log-convex follows similarly with all inequalities below reversed. We express the integral in (B.6) as,

$$\mathbb{E}[C] = \sum_{u=1}^{r} u \left(\int_0^\infty \prod_{i=1}^{u} \Pr(X > x + t_u - t_i)dx - \int_0^\infty \prod_{i=1}^{u} \Pr(X > x + t_{u+1} - t_i)dx \right),$$
(B.7)

$$= \sum_{u=1}^{r} \left(\int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i \right) dx' - \right.$$
$$\left. \int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_{u+1} - t_i \right) dx' \right),$$
(B.8)

$$= \mathbb{E}[X] + \sum_{u=2}^{r} \int_0^\infty \left(\prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i \right) - \right.$$
$$\left. \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'}{u - 1} + t_u - t_i \right) \right) dx',$$
(B.9)

$$\geq \mathbb{E}[X],$$
(B.10)

where in (B.7) we express each integral in (B.6) as a difference of two integrals from 0 to ∞. In (B.8) we perform a change of variables $x = x'/u$. In (B.9) we rearrange the grouping of the terms in the sum; the uth negative integral is put in the $u+1$ term of the summation. Then the first term of the summation is simply $\int_0^\infty \Pr(X > x)dx$ which is equal to $\mathbb{E}[X]$. In (B.9) we use the fact that each term in the summation in (B.8) is positive when $\bar{F}_X$ is log-concave. This is shown in Lemma 20 below.

Next we prove that for log-concave $\bar{F}_X$, $\mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}]$. Again, the proof of $\mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}]$ when $\bar{F}_X$ is log-convex follows with all the inequalities below reversed.

$$\mathbb{E}\left[C\right] \le \sum_{u=1}^{r} u \int_{0}^{t_{u+1}-t_u} \prod_{i=1}^{u} \Pr\left(X > \frac{u(x+t_u-t_i)}{r}\right)^{r/u} dx, \tag{B.11}$$

$$= \sum_{u=1}^{r} \left(\int_{0}^{\infty} \prod_{i=1}^{u} \Pr\left(X > \frac{x'+u(t_u-t_i)}{r}\right)^{r/u} dx' - \right.$$
$$\left. \int_{0}^{\infty} \prod_{i=1}^{u} \Pr\left(X > \frac{x'+u(t_{u+1}-t_i)}{r}\right)^{r/u} dx' \right), \tag{B.12}$$

$$= \int_{0}^{\infty} \Pr\left(X > \frac{x'}{r}\right)^{r} dx' + \sum_{u=2}^{r} \left(\int_{0}^{\infty} \prod_{i=1}^{u} \Pr\left(X > \frac{x'+u(t_u-t_i)}{r}\right)^{r/u} dx' - \right.$$
$$\left. \int_{0}^{\infty} \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'+(u-1)(t_u-t_i)}{r}\right)^{\frac{r}{u-1}} dx' \right), \tag{B.13}$$

$$\le r\mathbb{E}\left[X_{1:r}\right], \tag{B.14}$$

where we get (B.11) by applying Property 2 to (B.6). In (B.12) we express the integral as a difference of two integrals from 0 to ∞, and perform a change of variables $x = x'/u$. In (B.13) we rearrange the grouping of the terms in the sum; the uth negative integral is put in the $u+1$ term of the summation. The first term is equal to $r\mathbb{E}\left[X_{1:r}\right]$. We use Lemma 21 to show that each term in the summation in (B.13) is negative when $\bar{F}_X$ is log-concave. $\square$

Lemma 20. *If $\bar{F}_X$ is log-concave,*

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) \ge \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'}{u-1} + t_u - t_i\right). \tag{B.15}$$

The inequality is reversed for log-convex $\bar{F}_X$.

Proof of Lemma 20. We bound the left hand side expression as follows.

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x}{u} + t_u - t_i\right)$$
$$= \Pr(S > t_u) \prod_{i=1}^{u} \Pr\left(X > \frac{x}{u} + t_u - t_i | X > t_u - t_i\right), \tag{B.16}$$

167

$$= \Pr(S > t_u) \Pr\left(X > \frac{x}{u}\right)^{\frac{u-1}{u-1}} \times \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u} + t_u - t_i | X > t_u - t_i\right), \quad \text{(B.17)}$$

$$\geq \Pr(S > t_u) \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u} + t_u - t_i | X > t_u - t_i\right)^{\frac{u}{u-1}}, \quad \text{(B.18)}$$

$$\geq \Pr(S > t_u) \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u-1} + t_u - t_i | X > t_u - t_i\right), \quad \text{(B.19)}$$

$$= \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u-1} + t_u - t_i\right) \quad \text{(B.20)}$$

where we use Property 3 to get (B.18). The inequality in (B.19) follows from applying Property 2 to the conditional distribution $\Pr(Y > x'/u) = \Pr(X > x'/u + t_u - t_i | X > t_u - t_i)$, which is also log-concave.

For log-convex $\bar{F}_X$ all the inequalities can be reversed. $\qquad\square$

Lemma 21. *If $\bar{F}_X$ is log-concave,*

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{r/u} \leq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + (u-1)(t_u - t_i)}{r}\right)^{\frac{r}{u-1}} \quad \text{(B.21)}$$

The inequality is reversed for log-convex $\bar{F}_X$.

Proof of Lemma 21. We start by simplifying the left-hand side expression, raised to the power $(u-1)/r$.

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{\frac{u-1}{u}} = \Pr\left(X > \frac{x}{r}\right)^{\frac{u-1}{u}} \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{\frac{u-1}{u}}$$

$$\text{(B.22)}$$

$$= \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{r}\right)^{\frac{1}{u}} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{\frac{u-1}{u}}$$

$$\text{(B.23)}$$

$$\leq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + (u-1)(t_u - t_i)}{r}\right) \quad \text{(B.24)}$$

where (B.24) follows from the log-concavity of $\Pr(X > x)$, and the Jensen's equality. The inequality is reversed for log-convex $\bar{F}_X$. $\qquad\square$

Appendix C

Results from Order Statistics

C.1 Central order statistics

For an order statistic $X_{k:n}$, we called it a central order statistic if $k \approx np$ for some $p \in (0,1)$. In this case, $X_{k:n}$ is asymptotically normal, concentrated around the p-th quantile of X, as indicated by the following result called the Central Value Theorem (Theorem 10.3 in [118]).

Theorem 14 (Central Value Theorem)**.** *Given* $X_1, X_2, \ldots, X_n \stackrel{i.i.d.}{\sim} F$, *if* $0 < p < 1$ *and* $0 < f(x_p) < \infty$, *where* $x_p = F^{-1}(p)$, *then for* $k = k(n)$ *such that* $k = np + o(\sqrt{n})$,

$$X_{k:n} \stackrel{P}{\to} N\left(x_p, \frac{p(1-p)}{n f^2(x_p)}\right)$$

where $f(\cdot)$ *is the p.d.f. corresponds to* F *and* $\stackrel{P}{\to}$ *denotes convergence in probability as* $n \to \infty$.

C.2 Extreme order statistics

Extreme value theory (EVT) is an asymptotic theory of extremes, i.e., minima and maxima. It shows that if a distribution belongs to one of three families of distributions Theorem 15), then its maxima can be well characterized asymptotically as given

by Theorem 16, which is also referred to as the Fisher-Tippett-Gnedenko Theorem (Theorem 1.1.3 in [119]).

Theorem 15 (Domains of attraction). *A distribution function F_X has one of the following domains of attraction if it satisfies the conditions of the extreme value distribution $G(x)$ if and only if*

1. *$F_X \in \mathrm{DA}\,(\Lambda)$ if and only if there exists $\eta(x) > 0$ such that*

$$\lim_{x \to \omega(F)^-} \frac{\bar{F}(x + t\eta(x))}{\bar{F}(x)} = e^{-t};$$

2. *$F_X \in \mathrm{DA}\,(\Phi_\xi)$ if and only if $\omega\,(F) = \infty$ and*

$$\lim_{x \to \infty} \frac{\bar{F}(tx)}{\bar{F}(x)} = t^{-\xi}, \quad t > 0;$$

3. *$F_X \in \mathrm{DA}\,(\Psi_\xi)$ if and only if $\omega\,(F) < \infty$ and*

$$\lim_{x \to 0^+} \frac{\bar{F}(\omega\,(F) - tx)}{\bar{F}(\omega\,(F) - x)} = t^{\xi}, \quad t > 0;$$

where $\omega\,(x) = \sup\{x : F_X(x) < 1\}$, the upper end point of the distribution F_X.

Intuitively, $F \in \mathrm{DA}\,(\Lambda)$ corresponds to the case that $\bar{F}$ has an exponentially decaying tail, $F \in \mathrm{DA}\,(\Phi_\xi)$ corresponds to the case that $\bar{F}$ has heavy tail (such as polynomially decaying), and $F \in \mathrm{DA}\,(\Psi_\xi)$ corresponds to the case that $\bar{F}$ has a short tail with finite upper bound.

Theorem 16 (Extreme Value Theorem). *Given $X_1, \ldots, X_n \overset{i.i.d.}{\sim} F$, if there exist sequences of constants $a_n > 0$ and $b_n \in \mathbb{R}$ such that*

$$\mathbb{P}\left[(X_{n:n} - b_n)/a_n \le x\right] \to G(x) \tag{C.1}$$

as $n \to \infty$ and $G(\cdot)$ is a non-degenerate distribution. The extreme value distribution $G(x)$ and the values of a_n and b_n depend on the domain of attraction (and hence the tail behavior) of F_X given by Theorem 15.

1. *For $F_X \in \mathrm{DA}(\Lambda)$,*

$$a_n = \eta \left(F^{-1}(1 - 1/n) \right), \qquad (\mathrm{C.2})$$

$$b_n = F^{-1}(1 - 1/n) \qquad (\mathrm{C.3})$$

$$G(x) = \Lambda(x) = \exp\left\{ -\exp\left(-x \right) \right\} \qquad (\mathrm{C.4})$$

where $\Lambda(x)$ is called the Gumbel distribution.

2. *For $F_X \in \mathrm{DA}(\Phi_\xi)$,*

$$a_n = F^{-1}(1 - 1/n), \qquad (\mathrm{C.5})$$

$$b_n = 0, \qquad (\mathrm{C.6})$$

$$G(x) = \Phi_\xi(x) = \begin{cases} 0 & x \leq 0 \\ \exp\left\{ -x^{-\xi} \right\} & x > 0 \end{cases}. \qquad (\mathrm{C.7})$$

where $\Phi_\xi(x)$ is called the Fréchet distribution.

3. *For $F_X \in \mathrm{DA}(\Psi_\xi)$,*

$$a_n = \omega\left(F \right) - F^{-1}(1 - 1/n), \qquad (\mathrm{C.8})$$

$$b_n = \omega\left(F \right), \qquad (\mathrm{C.9})$$

$$G(x) = \Psi_\xi(x) = \begin{cases} \exp\left\{ -(-x)^\xi \right\} & x < 0, \\ 1 & x \geq 0. \end{cases} \qquad (\mathrm{C.10})$$

where $\Psi_\xi(x)$ is called the reversed-Weibull distribution.

Based on Theorem 16, we can derive the expected value of extreme values, as shown in Lemma 22.

Lemma 22 (Expected Extreme Values).

$$\mathbb{E}\left[\Lambda\right] = \gamma_{\mathrm{EM}},$$

$$\mathbb{E}\left[\Phi_\xi\right] = \begin{cases} \Gamma\left(1 - 1/\xi\right) & \xi > 1 \\ +\infty & \text{otherwise}, \end{cases}$$

$$\mathbb{E}\left[\Psi_\xi\right] = -\Gamma\left(1 + 1/\xi\right),$$

where γ_{EM} is the Euler-Mascheroni constant and $\Gamma(\cdot)$ is the Gamma function, i.e.,

$$\Gamma(t) \triangleq \int_0^\infty x^{t-1} e^{-x} \, dx.$$

We can also characterize the limit distribution of the sample extreme $X_{1:n}$ analogously via Theorem 16 by

$$X_{1:n} = \min\left\{X_1, \ldots, X_n\right\} = -\max\left\{-X_1, \ldots, -X_n\right\}.$$

It is worth noting that the distribution function for $-X$ may be in a different domain of attraction from that of X.

Appendix D

Proofs of Chapter 3

D.1 Latency and cost for general F_X

Proof of Lemma 9. First consider π_{kill} where we relaunch the original copy, and add r replicas for each of the pn straggling tasks. Thus, there are $r+1$ identical replicas of each task after forking. The residual execution time distribution F_Y (after time $T^{(1)}$ when the replicas are added) of each task is the minimum of $r+1$ i.i.d. random variables with distribution F_X. Hence,

$$\Pr(Y > y) = \Pr(\min(X_1, X_2, \ldots X_{r+1}) > y), \tag{D.1}$$

$$\bar{F}_Y(y) = \bar{F}_X(y)^{r+1} \quad \text{for } \pi_{\text{kill}}. \tag{D.2}$$

For π_{kill}, there is 1 original replica and r new replicas of each of the straggling tasks. Thus, the tail distribution $\bar{F}_Y(y) = 1 - F_Y(y)$ is given by

$$\Pr(Y > y) = \Pr(X_1 > y + T^{(1)} | X_1 > T^{(1)}) \cdot \Pr(\min(X_2, \ldots X_{r+1}) > y), \tag{D.3}$$

$$\bar{F}_Y(y) = \frac{\bar{F}_X(y + T^{(1)})}{\bar{F}_X(T^{(1)})} \bar{F}_X(y)^r. \tag{D.4}$$

As the number of tasks $n \to \infty$ by Theorem 14 we have $T^{(1)} \to F_X^{-1}(1-p)$. Hence,

$$\bar{F}_Y(y) = \frac{\bar{F}_X\left(y + F_X^{-1}(1-p)\right)}{p} \bar{F}_X(y)^r \quad \text{for } \pi_{\text{keep}}. \tag{D.5}$$

Proof of Theorem 5. The expected latency $\mathbb{E}[T]$ can be divided into two parts: before and after replication.

$$\begin{aligned}
\mathbb{E}[T] &= \mathbb{E}\left[T^{(1)}\right] + \mathbb{E}\left[T^{(2)}\right], \\
&= \mathbb{E}\left[X_{(1-p)n:n}\right] + \mathbb{E}\left[\max_{j=1,2,\dots,pn} Y_j\right], \\
&= F_X^{-1}(1-p) + \mathbb{E}\left[Y_{pn:pn}\right]. \tag{D.6}
\end{aligned}$$

The time before forking $T^{(1)}$ is the time until $(1-p)n$ of the n tasks launched at time 0 finish. Thus, its expected value $\mathbb{E}\left[T^{(1)}\right]$ is the expectation of the $(1-p)n^{th}$ order statistic $X_{(1-p)n:n}$ of n i.i.d. random variables with distribution F_X. By the Central Value Theorem stated as Theorem 14, for $n \to \infty$, this term goes to inverse CDF value $F_X^{-1}(1-p)$. At this forking point, the scheduler introduces replicas of the pn straggling tasks. The distribution F_Y of the residual execution time (minimum over the $r+1$ replicas) of each straggling task is given by Lemma 9. Thus the term $\mathbb{E}\left[T^{(2)}\right]$ in (D.6) is the expected value of the maximum of pn i.i.d. random variables with distribution F_Y.

Recall from Definition 9 that the expected cost $\mathbb{E}[C]$ is the sum of the running times of all machines, normalized by the number of tasks n. We can analyze $\mathbb{E}[C]$ by dividing it into sum of machine runtimes before and after forking.

$$\mathbb{E}[C] = \mathbb{E}\left[C^{(1)}\right] + \mathbb{E}\left[C^{(2)}\right], \tag{D.7}$$

$$\mathbb{E}\left[C^{(1)}\right] = \frac{1}{n}\sum_{i=1}^{(1-p)n} \mathbb{E}[X_{i:n}] + \frac{np}{n}\mathbb{E}\left[T^{(1)}\right], \tag{D.8}$$

$$= \frac{1}{n}\sum_{i=1}^{(1-p)n} F_X^{-1}\left(\frac{i}{n}\right) + pF_X^{-1}(1-p), \tag{D.9}$$

$$= \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p). \tag{D.10}$$

174

$$\mathbb{E}\left[C^{(2)}\right] = \frac{1}{n}\sum_{j=1}^{pn}(r+1)\mathbb{E}\left[Y_j\right], \tag{D.11}$$

$$= (r+1)p \cdot \mathbb{E}\left[Y\right]. \tag{D.12}$$

The cost before forking $\mathbb{E}\left[C^{(1)}\right]$ consists of the cost for the $(1-p)n$ tasks that finish first, plus the cost for the pn straggling tasks. The first term in (D.8) is the sum of the expected values of the smallest $(1-p)n$ execution times. Using Theorem 14, we can show that the ith term in the summation goes to $F_X^{-1}(i/n)$ as $n \to \infty$. Expressing the sum as an integral over $h = i/n$ we get the first term in (D.10). The second term in (D.8), is the normalized running time of the pn straggling tasks before forking. Substituting $\mathbb{E}\left[T^{(1)}\right]$ from (D.6) and simplifying, we get (D.10).

The cost after forking, $\mathbb{E}\left[C^{(2)}\right]$ is the normalized sum of the runtimes of the $r+1$ replicas of each of the pn straggling tasks. By Lemma 9, the residual execution time of the jth straggling task is $Y_j \sim F_Y$. Since the scheduler kills all replicas as soon as one replica finishes, the expected runtime for the jth straggling task is $(r+1)\mathbb{E}\left[Y_j\right]$. Thus, the cost in (D.11) is the sum of $(r+1)\mathbb{E}\left[Y_j\right]$ over the pn tasks, normalized by n. Since Y_j are i.i.d, we can reduce this to (D.12). $\qquad\square$

To prove Lemma 10, we characterize the expected maximum of a large number of random variables using Theorem 16. First, we state Lemma 23 which implies that the domain of attraction (see Theorem 15) of F_Y is same as that of F_X.

Lemma 23 (Domain of attraction for F_Y). *Given a single fork policy $\pi\left(p, r; n\right)$ with $0 < p < 1$,*

1. *if $F_X \in \mathrm{DA}\left(\Lambda\right)$, then $F_Y \in \mathrm{DA}\left(\Lambda\right)$;*

2. *if $F_X \in \mathrm{DA}\left(\Phi_\xi\right)$, then $F_Y \in \mathrm{DA}\left(\Phi_{(r+1)\xi}\right)$;*

3. *if $F_X \in \mathrm{DA}\left(\Psi_\xi\right)$, then $F_Y \in \mathrm{DA}\left(\Psi_{(r+1)\xi}\right)$ for $\pi_{\mathrm{kill}}\left(p, r\right)$ and $F_Y \in \mathrm{DA}\left(\Psi_\xi\right)$ for $\pi_{\mathrm{keep}}\left(p, r\right)$.*

The proof follows directly from Lemma 9 and Theorem 15, and hence is omitted here.

Proof of Lemma 10. We can use Lemma 23 to find the domain of attraction of F_Y. Then from (C.1) we have

$$\mathbb{E}[Y_{n:n}] = \tilde{a}_n \mathbb{E}[G(y)] + \tilde{b}_n,$$

where $\mathbb{E}[G(y)]$ can be found using Theorem 16 and Lemma 22. □

Proof of Lemma 11. When we kill the original copy, the residual execution time

$$Y_{\text{kill}} = \min\{X_{1:r}, X_O\},$$

where X_O is the additional time needed for the original copy to finish, and satisfies

$$\mathbb{P}[X_O > x] = \mathbb{P}[X > x + t \mid X > t],$$

where t is the forking time. When we keep the original copy, the residual execution time satisfies

$$Y_{\text{keep}} = \min\{X_{1:r}, X\}.$$

Then the proof follows from the properties of stochastic dominance of X_O over X and vice-versa depending on whether X is new-longer-than-used or new-shorter-than-used. □

D.2   Latency and Cost for Pareto F_X

We prove Theorem 7, which evaluates the latency $\mathbb{E}[T]$ and computing cost $\mathbb{E}[C]$ metrics when the task execution time distribution F_X is the Pareto, as defined in (3.16).

Proof of Theorem 7. From Theorem 5 we have

$$\mathbb{E}\left[T\right] = F_X^{-1}(1-p) + \mathbb{E}\left[Y_{pn:pn}\right],$$

$$= x_m p^{-1/\alpha} + \tilde{a}_{pn}\mathbb{E}\left[\Phi_{(r+1)\alpha}\right], \tag{D.13}$$

$$= x_m p^{-1/\alpha} + \tilde{a}_{pn}\Gamma\left(1 - 1\frac{1}{(r+1)\alpha}\right). \tag{D.14}$$

$$\mathbb{E}\left[C\right] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + (r+1)p \cdot \mathbb{E}\left[Y\right], \tag{D.15}$$

$$= x_m \int_0^{1-p} (1-h)^{-1/\alpha}dh + px_m p^{-1/\alpha} + (r+1)p \cdot \mathbb{E}\left[Y\right],$$

$$= x_m \frac{\alpha}{\alpha - 1}[1 - p^{1-1/\alpha}] + x_m p^{1-1/\alpha} + (r+1)p \cdot \mathbb{E}\left[Y\right],$$

$$= x_m \frac{\alpha}{\alpha - 1} - x_m \frac{p^{1-1/\alpha}}{\alpha - 1} + (r+1)p \cdot \mathbb{E}\left[Y\right]. \tag{D.16}$$

To obtain (D.13) we first observe that since F_X is Pareto, by Theorem 15 it falls into the Fréchet domain of attraction, i.e. $F_X \in \mathrm{DA}\left(\Phi_\alpha\right)$. Then using Lemma 23 we can show that $F_Y \in \mathrm{DA}\left(\Phi_{(r+1)\alpha}\right)$. Subsequently, using Theorem 16 and Lemma 22 we get (D.14). To derive the expected cost (D.16) we substitute $F_X^{-1}(h) = x_m(1-h)^{-1/\alpha}$ in the first and second terms in (D.15) and simplify the expression. To find $\tilde{a}_{pn}$ and $\mathbb{E}\left[Y\right]$ in (D.14) and (D.16) respectively we consider the cases of killing the original task (π_{kill}) and keeping the original task (π_{keep}) separately.

Case 1: Killing the original task(π_{kill})

For a single-fork policy that kills the original task, the scheduler waits for $(1-p)n$ tasks to finish and then relaunches each of the pn straggler tasks on a new machine.

$$Y = \min(X_1, X_2, \ldots X_{r+1}),$$

$$Y \sim \mathsf{Pareto}\left((r+1)\alpha, x_m\right). \tag{D.17}$$

From (C.5) in Theorem 16 we can evaluate $\tilde{a}_{pn}$ as follows

$$\tilde{a}_{pn} = F_Y^{-1}\left(1 - \frac{1}{pn}\right) = x_m(pn)^{1/\alpha}.$$

And $\mathbb{E}[Y]$ of (D.17) can be evaluated as

$$\mathbb{E}[Y] = \frac{(r+1)\alpha}{(r+1)\alpha - 1} x_m. \tag{D.18}$$

Case 2: Keeping the original task (π_{keep})

For a single-fork policy that keeps the original task, the scheduler keeps the original copy, and adds r additional replicas for each straggling task. Using Lemma 9 we can show that

$$\bar{F}_Y(y) = \frac{1}{p} \left(\frac{x_m}{y} \right)^{\alpha r} \left(\frac{x_m}{y + x_m p^{-1/\alpha}} \right)^{\alpha}. \tag{D.19}$$

From (C.5) in Theorem 16, $\tilde{a}_{pn} = \bar{F}_Y^{-1}\left(\frac{1}{pn}\right)$, which simplifies to

$$(pn)^{1/\alpha} = \left(1 + \frac{\tilde{a}_{pn}}{x_m p^{-1/\alpha}} \right) \left(\frac{\tilde{a}_{pn}}{x_m} \right)^r,$$

which simplifies to (3.21). The expected value of Y can be found by numerically integrating $\bar{F}_Y(y)$ in (D.19) over its support. $\qquad\square$

Proof of Corollary 6. For the case of killing the original task, it follows directly from (3.17) and (3.19). For the case of keeping the original task, note that $\tilde{a}_{pn}$ grows with n, and hence when n is large enough, from (3.21) we have

$$\tilde{a}_{pn}^{r+1} \leq n^{1/\alpha} x_m^{r+1} \leq 2\tilde{a}_{pn}^{r+1}, \tag{D.20}$$

and then the result holds again following (3.17). $\qquad\square$

D.3   Latency and Cost for Shifted Exponential F_X

Now we prove Theorem 6, which gives the latency-cost trade-off when the distribution of the execution time X is a shifted exponential given by (3.13).

Proof of Theorem 6.

$$\mathbb{E}[T] = F_X^{-1}(1-p) + \mathbb{E}[Y_{pn:pn}],$$

$$= \Delta - \frac{1}{\mu}\ln p + \tilde{a}_{pn}\mathbb{E}[\Lambda] + \tilde{b}_{pn}, \tag{D.21}$$

$$= \Delta - \frac{1}{\mu}\ln p + \tilde{a}_{pn}\gamma_{\text{EM}} + \tilde{b}_{pn}. \tag{D.22}$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + (r+1)p \cdot \mathbb{E}[Y], \tag{D.23}$$

$$= \int_0^{1-p} \left(\Delta - \frac{1}{\mu}\ln(1-h)\right) dh + p\left(\Delta - \frac{1}{\mu}\ln p\right),$$

$$+ (r+1)p \cdot \mathbb{E}[Y], \tag{D.24}$$

$$= \Delta + \frac{1}{\mu}(p\ln p + (1-p)) + p\Delta - \frac{p}{\mu}\ln p,$$

$$+ (r+1)p \cdot \mathbb{E}[Y], \tag{D.25}$$

$$= \Delta(1+p) + \frac{1-p}{\mu} + (r+1)p \cdot \mathbb{E}[Y]. \tag{D.26}$$

To find $\mathbb{E}[Y]$, $\tilde{a}_{pn}$ and $\tilde{b}_{pn}$ we consider the cases of relaunching ($l = 0$) and no relaunching ($l = 1$) separately.

Case 1: Killing the original task (π_{kill})

$$Y = \min\{X_1, X_2, \cdots X_{r+1}\} \tag{D.27}$$

$$\sim \mathsf{ShiftedExp}\left(\Delta, (r+1)\mu\right) \tag{D.28}$$

$$\mathbb{E}[Y] = \Delta + \frac{1}{(r+1)\mu} \tag{D.29}$$

Based on Theorem 15, for $\eta(y) = 1/((r+1)\mu)$ we have

$$\lim_{y\to\omega(F_Y)} \frac{\bar{F}_Y(y + u\eta(y))}{\bar{F}_Y(y)} = e^{-u}. \tag{D.30}$$

By Theorem 16 and Theorem 15, the maximum of shifted exponential belongs to the

Gumbel family with

$$\tilde{a}_{pn} = \frac{1}{\mu(1+r)},$$

$$\tilde{b}_{pn} = \bar{F}_Y^{-1}(1/n) = \Delta + \frac{\ln(pn)}{\mu(r+1)}.$$

Case 2: Keeping the original task (π_{keep})

In the case of no relaunching,

$$Y = \min\left\{\mathsf{Exp}(\mu), \Delta + \mathsf{Exp}(r\mu)\right\}.$$

Note that the first term does not include Δ because for large n the original task would have run for at least Δ seconds. Thus the tail distribution of Y is given by

$$\bar{F}_Y(y) = \begin{cases} e^{-\mu y} & 0 < y < \Delta, \\ e^{\mu r \Delta} e^{-\mu(r+1)y} & y \geq \Delta. \end{cases} \tag{D.31}$$

The expected value $\mathbb{E}[Y]$ is the integration of $\bar{F}_Y(y)$ over its support.

$$\mathbb{E}[Y] = \int_0^\Delta e^{-\mu y} dy + \int_\Delta^\infty e^{\mu r \Delta} e^{-\mu(r+1)y},$$

$$= \frac{1 - e^{-\mu\Delta}}{\mu} + \frac{e^{-\mu\Delta}}{\mu(r+1)}.$$

By Theorem 16 and Theorem 15 similar to the relaunching case we have

$$\tilde{a}_{pn} = 1/\left[\mu(1+r)\right],$$

$$\tilde{b}_{pn} = \bar{F}_Y^{-1}(1/n) = \frac{r}{r+1}\Delta + \frac{\ln(pn)}{\mu(r+1)}.$$

$\square$

Appendix E

Proofs of Chapter 6

Proof of Theorem 8. To find the upper bound on latency, we consider a related queueing system called the split-merge queueing system. In the split-merge system all the queues are blocked and cannot serve the next tasks in queue until k tasks of the current job finish. This phenomenon is illustrated in Fig. 6-1. Thus the latency of the split-merge system serves as an upper bound on that of the fork-join system. In the split-merge system we observe that jobs are served one-by-one, and no two jobs are served simultaneously. So it is equivalent to an $M/G/1$ queue with Poisson arrival rate λ, and service time $X_{k:n}$, the kth order statistic of the i.i.d. service times $X_1, X_2, \cdots X_n$. The expected latency of an $M/G/1$ queue is given by the Pollaczek-Khinchine formula [120, Chapter 5], and it reduces to the upper bound in (6.13).

To find the lower bound we consider a genie system where the job requires k out of n tasks to complete, but all jobs arriving before it require only 1 task to finish. Then the service time is $\mathbb{E}\left[X_{k:n}\right]$. The expected waiting time in queue is equal to the second term in (6.13) with k set to 1. Adding the expected service and the lower bound on expected waiting time, we get the lower bound (6.14) on the expected latency. □

Proof of Lemma 13. The bound above is a generalization of the bound for the (n, n) fork-join system with exponential service time presented in [9]. To find the bound, we first observe that the response times of the n queues form a set of associated random variables [121]. Then we use the property of associated random variables that their

expected maximum is less than that for independent variables with the same marginal distributions. Unfortunately, this approach cannot be directly extended to the (n, k) fork-join system with $k < n$ because this property of associated variables does not hold for the kth order statistic for $k < n$. $\qquad \square$

Proof of Lemma 14. First we derive the lower bound for the case when service time is a pure exponential with rate μ. The lower bound in (6.6) is a generalization of the bound for the (n, n) fork-join system derived in [10]. The bound for the (n, n) system is derived by considering that a job goes through n stages of processing. A job is said to be in the jth stage if j out of n tasks have been served by their respective nodes for $0 \le j \le n - 1$. The job waits for the remaining $n - j$ tasks to be served, after which it departs the system. For the (n, k) fork-join system, since we only need k tasks to finish service, each job now goes through k stages of processing. In the jth stage, where $0 \le j \le k - 1$, j tasks have been served and the job will depart when $k - j$ more tasks to finish service.

We now show that the service rate of a job in the jth stage of processing is *at most* $(n - j)\mu$. Consider two jobs B_1 and B_2 in the ith and j^{th} stages of processing respectively. Let $i > j$, that is, B_1 has completed more tasks than B_2. Job B_2 moves to the $(j + 1)^{th}$ stage when one of its $n - j$ remaining tasks complete. If all these tasks are at the heads of their respective queues, the service rate for job B_2 is exactly $(n - j)\mu$. However since $i > j$, B_1's task could be ahead of B_2's in one of the $n - j$ pending queues, due to which that task of B_2 cannot be immediately served. Hence, we have shown that the service rate of in the jth stage of processing is at most $(n-j)\mu$. Thus, for pure exponential service time,

$$\mathbb{E}\left[T\right] \ge \sum_{j=0}^{k-1} \frac{1}{(n - j)\mu - \lambda} \tag{E.1}$$

For the shifted exponential distribution, we can a closed-form expression for a lower bound on latency. The first term gives the time taken to serve the first of the k chunks of the file. The last term is the same as the last $k - 1$ terms of the summation in (E.1). It is the expected sum of the residual response times, without considering

the Δ shift of the distribution. $\qquad\qquad\square$

Proof of Theorem 9. A central idea that is used in proving both the bounds is that at least $n - k + 1$ out of the n tasks of a job i start service at the same time. This is because when the kth task of job $i - 1$ finishes, the remaining $n - k$ tasks are canceled immediately. These $n - k + 1$ queues start working on the tasks of job i at the same time. Note that the result holds trivially if job i arrives when all n queues are idle.

To prove the upper bound we divide the n tasks into two groups, the $k - 1$ tasks that can start early, and the $n - k + 1$ which start at the same time after the last tasks of the previous job are terminated. We consider that all the $k - 1$ tasks in the first group and 1 of the remaining $n - k + 1$ tasks needs to be served for completion of the job. This gives an upper bound on the computing cost because we are not taking into account the case where more than one tasks from the second group can finish service before the $k - 1$ tasks in the first group. For the $n - k + 1$ tasks in the second group, the computing cost is equal to $n - k + 1$ times the time taken for one of them to complete. The computing time spent on the first $k - 1$ tasks is at most $(k - 1)\mathbb{E}[X]$. Adding this to the second group's cost, we get the upper bound (6.15).

We observe that the expected computing cost for the k tasks that finish is at least $\sum_{i=1}^{k} \mathbb{E}[X_{i:n}]$, which takes into account full diversity of the redundant tasks. Since we need k tasks to complete in total, at least 1 of the remaining $n - k + 1$ tasks needs to be served. Thus, the computing cost of the $(n - k)$ redundant tasks is at least $(n - k)\mathbb{E}[X_{1:n-k+1}]$. Adding this to the lower bound on the first group's cost, we get (6.16). $\qquad\qquad\square$

Proof of Theorem 10. Since exactly k tasks are served, and others are cancelled before they start service, it follows that the expected computing cost $\mathbb{E}[C] = k\mathbb{E}[X]$.

To find an upper bound on the latency, consider a partial fork system without redundancy where each job has k tasks that assigned to k out of n queues, chosen uniformly at random. The job exits the system when all k tasks are complete. Although only k tasks enter service in the (n, k) fork-early-cancel, it gives lower latency because having the $n - k$ redundant tasks provides diversity and helps find the k

shortest queues.

Now let us upper bound the latency $\mathbb{E}\left[T^{(pf)}\right]$ of the partial fork-join system. Each queue has arrival rate $\lambda k/n$, and service distribution F_X. Using the approach in [9] we can show that the response times (waiting plus service time) R_i, $1 \leq i \leq k$ of the k queues serving each job form a set of associated random variables. Using the property that the expected maximum of k associated random variables is less than the expected maximum of k independent variables with the same marginal distributions we can show that,

$$\mathbb{E}\left[T\right] \leq \mathbb{E}\left[T^{(pf)}\right] \tag{E.2}$$

$$\leq \mathbb{E}\left[\max\left(R_1, R_2, \cdots R_k\right)\right] \tag{E.3}$$

where R_i are i.i.d. random variables with distribution same as the response time of each queue. It is a standard result [54, Chapter 25] that the Laplace-Stieltjes transform of the response time of an $M/G/1$ queue with service distribution $F_X(x)$ is (6.4). $\qquad \square$

Appendix F

Proofs of Chapter 8

Proof of Theorem 11. The in-order decoding delay D_k of packet s_k can be expressed as a sum of inter-delivery times as follows.

$$D_k = T_1 + T_2 + \dots T_W \tag{F.1}$$

where W is the number of in-order delivery instants until packets $s_1, \dots s_k$ are decoded. The random variable W can take values $1 \le W \le k$, since multiple packets may be decoded at one in-order delivery instants. Note that successive inter-delivery times $T_1, T_2, \dots, T_W$ are not i.i.d. The tail probability $\Pr(T_1 > t)$ of the first inter-delivery time is,

$$\Pr(T_1 > t) \ge \Pr(T_i > t) \text{ for all integers } i, t \ge 0. \tag{F.2}$$

This is because during the first inter-delivery time T_1 we start with no prior information. During time T_1, the receiver may collect coded combinations that it is not able to decode. For a time-invariant scheme, these coded combinations can result in faster in-order decoding and hence a smaller T_i for $i > 1$.

We now find lower and upper bounds on $\Pr(D_k \ge n)$ to find the decay rate of D_k.

The lower bound can be derived as follows.

$$\Pr(D_k \geq n) = \mathbb{E}_W\left[\Pr(T_1 + T_2 + ..T_W \geq n)\right] \tag{F.3}$$

$$\geq \Pr(T_1 \geq n) \tag{F.4}$$

$$\doteq e^{-\lambda n} \tag{F.5}$$

where (F.5) follows from Definition 23. Now we derive an upper bound on $\Pr(D_k > n)$.

$$\Pr(D_k \geq n) = \mathbb{E}_W\left[\Pr(T_1 + T_2 + \cdots + T_W \geq n)\right] \tag{F.6}$$

$$\leq \mathbb{E}_W\left[\Pr(T_1^{(1)} + T_1^{(2)} + \cdots + T_1^{(W)} \geq n)\right] \tag{F.7}$$

$$\leq \mathbb{E}_W\left[\sum_{n_i:\sum_{i=1}^W n_i=n} \prod_{i=1}^W \Pr(T_1^{(i)} > n_i)\right] \tag{F.8}$$

$$\doteq \mathbb{E}_W\left[\sum_{n_i:\sum_{i=1}^W n_i=n} e^{-\lambda(n_1+\cdots+n_W)}\right] \tag{F.9}$$

$$= \mathbb{E}_W\left[\binom{n+W-1}{W-1} e^{-\lambda n}\right] \tag{F.10}$$

$$\doteq \mathbb{E}_W\left[e^{-\lambda n}\right] \tag{F.11}$$

$$= e^{-\lambda n} \tag{F.12}$$

where in (F.7) $T_1^{(i)}$ are i.i.d. samples from the probability distribution of the first inter-delivery time T_1. By (F.2), replacing T_i by $T_1^{(i)}$ gives an upper bound on the probability $\Pr(D_k \geq n)$. In (F.8) we upper bound the tail probability in (F.7) by product of tail probabilities of each of the random variables $T_1^{(i)}$, with n_i being non-negative integers that sum to n. The product in (F.8) double-counts certain events and thus serves as a upper bound to $\Pr(T_1^{(1)} + T_1^{(2)} + \cdots + T_1^{(W)} \geq n)$. By (8.3), each term in the product in (F.8) asymptotically decays at rate λ. Thus we get (F.9) and (F.10). Since $W \leq k << n$, the binomial coefficient decays subexponentially, and we get (F.12).

From (F.5) and (F.12) we can conclude that the asymptotic decay rate $\lambda_k^{(s)}$ for

any k is equal to the inter-delivery exponent λ. □

Proof of Theorem 12. We first show that the scheme with transmit index $V[n] = \lceil rn \rceil$ in time slot n achieves the trade-off $(\tau, \lambda) = (r, D(r\|p))$. Then we prove the converse by showing that no other full-rank scheme gives a better trade-off.

Achievability Proof: Consider the scheme with transmit index $V[n] = \lceil rn \rceil$, where r represents the rate of adding new packets to the transmitted stream. The rate of adding packets is below the capacity of the erasure channel. Thus it is easy to see that the throughput $\tau = r$. Let $E[n]$ be the number of combinations, or equations received until time n. It follows the binomial distribution with parameter p. All packets $s_1 \cdots s_{V[n]}$ are decoded when $E[n] \geq V[n]$. Define the event $G_n = \{E[j] < V[j] \text{ for all } 1 \leq j \leq n\}$, that there is no packet decoding until slot n. The tail distribution of the first inter-delivery time T_1 is,

$$\Pr(T_1 > n) = \sum_{k=0}^{\lceil nr \rceil - 1} \Pr(E[n] = k) \Pr(G_n | E[n] = k),$$

$$= \sum_{k=0}^{\lceil nr \rceil - 1} \binom{n}{k} p^k (1-p)^{n-k} \Pr(G_n | E[n] = k),$$

where $\Pr(G_n | E[n] = k) = 1 - k/n$ as given by the Generalized Ballot theorem in [122, Chapter 4]. Hence it is sub-exponential and does not affect the exponent of $\Pr(T_1 > n)$ and we have

$$\Pr(T_1 > n) \doteq \sum_{k=0}^{\lceil nr \rceil - 1} \binom{n}{k} p^k (1-p)^{n-k}, \tag{F.13}$$

$$\doteq \binom{n}{\lceil nr \rceil - 1} p^{\lceil nr \rceil - 1} (1-p)^{n - \lceil nr \rceil + 1}, \tag{F.14}$$

$$\doteq e^{-nD(r\|p)}, \tag{F.15}$$

where in (F.13) we take the asymptotic equality $\doteq$ to find the exponent of $\Pr(T_1 > n)$, and remove the $\Pr(G_n | E[n] = k)$ term because it is sub-exponential. In (F.14), we only retain the $k = \lceil nr \rceil - 1$ term from the summation because for $r \leq p$, that term

asymptotically dominates other terms. Finally, we use the Stirlings approximation $\binom{n}{k} \approx e^{nH(k/n)}$ to obtain (F.15).

Converse Proof: First we show that the transmit index $V[n]$ of the optimal full-rank scheme should be non-decreasing in n. Given any scheme, we can permute the order of transmitting the coded packets such that $V[n]$ is non-decreasing in n. This does not affect the throughput τ, but it can improve the inter-delivery exponent λ because decoding can occur sooner when the initial coded packets include fewer source packets.

We now show that it is optimal to have $V[n] = \lceil rn \rceil$, where we add new packets to the transmitted stream at a constant rate r. Suppose a full-rank scheme uses rate r_i for n_i slots for all $1 \leq i \leq L$, such that $\sum_{i=0}^{L} n_i = n$ and $\sum_{i=1}^{L} n_i r_i = nr$. Then, the tail distribution of T_1 is,

$$\Pr(T_1 > n) = \sum_{k=0}^{\lceil \sum_{i=1}^{L} n_i r_i \rceil - 1} \Pr(E[n] = k) \Pr(G_n | E[n] = k), \tag{F.16}$$

$$\doteq \sum_{k=0}^{\lceil nr \rceil - 1} \binom{n}{k} p^k (1-p)^{n-k}, \tag{F.17}$$

$$\doteq e^{-nD(r \| p)}. \tag{F.18}$$

Varying the rate of adding packets affects the term $\Pr(G_n | E[n] = k)$ in (F.16), but it is still $\omega(1/n)$ and we can eliminate it when we take the asymptotic equality in (F.17). As a result, the in-order delay exponent is same as that if we had a constant rate r of adding new packets to the transmitted stream. Hence we have proved that no other full-rank scheme can achieve a better (τ, λ) trade-off than $V[n] = \lceil nr \rceil$ for all n. $\qquad \square$

Proof of Lemma 16. Here we prove the result for $B = 2$, that is randomizing between two schemes. It can be extended to general B using induction. Given two time-invariant schemes $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ that achieve the throughput-delay trade-offs $(\tau_{\mathbf{x}^{(1)}}, \lambda_{\mathbf{x}^{(1)}})$ and $(\tau_{\mathbf{x}^{(2)}}, \lambda_{\mathbf{x}^{(2)}})$ respectively, consider a randomized strategy where, in each block we use the scheme $\mathbf{x}^{(1)}$ with probability μ and scheme $\mathbf{x}^{(2)}$ otherwise. Then,

it is easy to see that the throughput on the new scheme is $\tau = \mu\tau_{\mathbf{x}^{(1)}} + (1-\mu)\tau_{\mathbf{x}^{(2)}}$.

Now we prove the inter-delivery exponent λ is also a convex combinations of $\lambda_{\mathbf{x}^{(1)}}$ and $\lambda_{\mathbf{x}^{(2)}}$. Let p_{d_1} and p_{d_2} be the probabilities of decoding the first unseen packet in a block using scheme $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ respectively. Suppose in an interval with k blocks, we use scheme $\mathbf{x}^{(1)}$ for h blocks, and scheme $\mathbf{x}^{(2)}$ in the remaining blocks, we have

$$\Pr(T_1 > kd) = (1 - p_{d_1})^h (1 - p_{d_2})^{k-h}. \tag{F.19}$$

Using this we can evaluate λ as,

$$\lambda = \lambda_{\mathbf{x}^{(1)}} \lim_{k\to\infty} \frac{h}{k} + \lambda_{\mathbf{x}^{(2)}} \lim_{k\to\infty} \frac{k-h}{k} \tag{F.20}$$

$$= \mu\lambda_{\mathbf{x}^{(1)}} + (1-\mu)\lambda_{\mathbf{x}^{(2)}} \tag{F.21}$$

where we get (F.20) using (8.6). As $k \to \infty$, by the weak law of large numbers, the fraction h/k converges to μ. □

Proof of Lemma 17. When $d = 2$ there are only two possible time-invariant schemes $\mathbf{x} = [2, 0]$ and $[1, 1]$ that give unique (τ, λ). By Remark 7, all other $\mathbf{x}$ are equivalent to one of these vectors in terms (τ, λ). The vectors $\mathbf{x} = [2, 0]$ and $[1, 1]$ correspond to the $a = 1$ and $a = 2$ codes proposed in Definition 26. Hence, the line joining their corresponding (τ, λ) points, as shown in Fig. 8-3, is the best trade-off for $d = 2$.

When $d = 3$ there are four time-invariant schemes $\mathbf{x}^{(1)} = [1, 0, 2]$, $\mathbf{x}^{(2)} = [2, 1, 0]$, $\mathbf{x}^{(3)} = [1, 2, 0]$ and $\mathbf{x}^{(4)} = [3, 0, 0]$ that give unique (τ, λ), according to Definition 24 and Remark 7. The vectors $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(4)}$ correspond to the codes with $a = 1, 2, 3$ in Definition 26. The throughput-delay trade-offs $(\tau_{\mathbf{x}^{(i)}}, \lambda_{\mathbf{x}^{(i)}})$ for $i = 1, 2, 4$ achieved by these schemes are given by (8.9). From Claim 5 and Claim 6 we know that $(\tau_{\mathbf{x}^{(1)}}, \lambda_{\mathbf{x}^{(1)}})$ and $(\tau_{\mathbf{x}^{(4)}}, \lambda_{\mathbf{x}^{(4)}})$ have to be on the optimal trade-off. By comparing the slopes of the lines joining these points we can show that the point $(\tau_{\mathbf{x}^{(2)}}, \lambda_{\mathbf{x}^{(2)}})$ lies above the line joining $(\tau_{\mathbf{x}^{(1)}}, \lambda_{\mathbf{x}^{(1)}})$ and $(\tau_{\mathbf{x}^{(4)}}, \lambda_{\mathbf{x}^{(4)}})$ for all p. Fig. 8-3 illustrates this

for $p = 0.6$. For the scheme with $\mathbf{x}^{(3)} = [1, 2, 0]$, we have

$$\left(\tau_{\mathbf{x}^{(3)}}, \lambda_{\mathbf{x}^{(3)}}\right) = \left((3p - p^3)/3, -(\log(1 - p)^2(1 + p))/3\right).$$

Again, by comparing the slopes of the lines joining $\left(\tau_{\mathbf{x}^{(i)}}, \lambda_{\mathbf{x}^{(i)}}\right)$ for $i = 1, \cdots 4$ we can show that for all p, $\left(\tau_{\mathbf{x}^{(3)}}, \lambda_{\mathbf{x}^{(3)}}\right)$ lies below the piecewise linear curve joining $\left(\tau_{\mathbf{x}^{(i)}}, \lambda_{\mathbf{x}^{(i)}}\right)$ for $i = 1, 2, 4$. $\qquad\square$

Appendix G

Proofs of Chapter 9

Proof on Claim 10. We now solve for the steady-state distribution of this Markov chain. Let π_i and π'_i be the steady-state probabilities of states i for $i \geq -1$ and advantages states i' for all $i \geq 0$ respectively. The steady-state transition equations are given by

$$(1 - a - d)\pi_i = b(\pi_{i-1} + \pi'_i) + a\pi'_{i+1} \quad \text{for } i \geq 1, \tag{G.1}$$

$$(1 - d)\pi'_i = c(\pi_i + \pi'_{i+1}) \quad \text{for } i \geq 1, \tag{G.2}$$

$$(1 - c - d)\pi_{-1} = c(\pi_0 + \pi'_1), \tag{G.3}$$

$$(1 - a - d)\pi_0 = a\pi'_1 + (a + b)\pi_{-1}. \tag{G.4}$$

By rearranging the terms in (G.1)-(G.4), we get the following recurrence relation,

$$\pi_i = \frac{(1 - a - d)}{c}\pi_{i-1} - \frac{b}{c}\pi_{i-2} \quad \text{for } i \geq 2. \tag{G.5}$$

Solving the recurrence in (G.5) and simplifying (G.1)-(G.4) further, we can express π_i, π'_i for $i \geq 2$ in terms of π_1 as follows,

$$\frac{\pi_i}{\pi_{i-1}} = \frac{b}{c}, \tag{G.6}$$

$$\frac{\pi'_i}{\pi_i} = \frac{c}{a + c}. \tag{G.7}$$

From (G.6) we see that the Markov chain is positive-recurrent and a unique steady-state distribution if and only if $b < c$, which is equivalent to $p_1 < p_2$. If $p_1 \geq p_2$, the expected recurrence time to state 0, that is the time taken for U_2 to catch up with U_1 is infinity. When the Markov chain is positive recurrent, we can use (G.6) and (G.7) to solve for all the steady state probabilities. $\qquad\square$

Proof of Lemma 18. Since we always give priority to the primary user U_1, we have $(\tau_1, \lambda_1) = (p_1, -\log(1 - p_1))$. When $p_1 < p_2$, we can express the throughput τ_2 in terms of the steady state probabilities of the Markov chain in Fig 9-2. User U_2 experiences a throughput loss when it is in state -1 and the next slot is successful. Thus, when $p_2 > p_1$,

$$\tau_2 = p_2(1 - \pi_{-1}), \tag{G.8}$$

$$= p_2 \left(1 - \frac{c - b}{a + c}\right) = p_1. \tag{G.9}$$

If $p_1 \geq p_2$, the system drifts infinitely to the right side. There is a non-zero probability that in-order decoding via advantage states is not able to catch up and fill all gaps in decoding of U_2. Thus, we cannot evaluate τ_2 using this Markov chain analysis.

To determine λ_2, first observe that U_2 decodes an in-order packet when the system is in state 0 or states i', for $i \geq 1$, and the next slot is successful. As given by Definition 23, the inter-delivery exponent λ_2 is the asymptotic decay rate of $\Pr(T_1 > t)$, the probability that no in-order packet is decoded by U_2 for t consecutive slots. To determine λ, we add an absorbing state F to the Markov chain as shown in Fig. G-1, such that the system transitions to F when an in-order packet is decoded by U_2.

In Fig. G-1, all the states i and i' for $i \geq 1$ are fused into states I and I' because this does not affect the probability distribution of the time to reach the absorbing state F. The inter-delivery exponent λ_2 is equal to the rate of convergence of this Markov chain to its steady state, which is known to be (see [120, Chapter 4]) $\lambda_2 = -\log \xi_2$ where ξ_2 is the second largest eigenvalue of the state transition matrix of the Markov
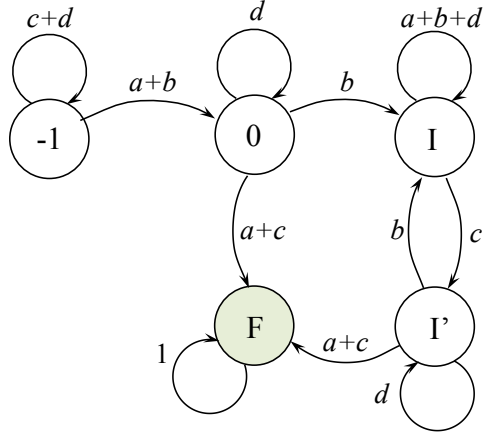
Figure G-1: Markov model used to determine the inter-delivery exponent λ_2 of user U_2. The absorbing state F is reached when an in-order packet is decoded by U_2. The exponent of the distribution of the time taken to reach this state is λ_2.

chain,

$$
A = \begin{pmatrix}
d & b & 0 & 0 & a+c \\
0 & a+b+d & c & 0 & 0 \\
0 & b & d & 0 & a+c \\
a+b & 0 & 0 & c+d & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}. \tag{G.10}
$$

Solving for the second largest eigen-value of A, we can show that

$$
\xi_2 = \max\left(1 - p_1, \frac{\left(1 - c + d + \sqrt{(1-c+d)^2 + 4(bc+cd-d)}\right)}{2}\right). \tag{G.11}
$$

Hence the inter-delivery exponent $\lambda_2 = -\log \xi_2$ is as given by (9.2). $\qquad \square$

Proof of Lemma 19. The state-transition equations of the Markov chain are as fol-

lows.

$$(\bar{d} - a)\pi_0 = a(\pi_1' + \pi_{-1}') + q_1(a + b)\pi_{-1} + q_2(a + c)\pi_1 \tag{G.12}$$

$$(\bar{d} - \bar{q}_2 a - q_2 b)\pi_i = q_2(a + c)\pi_{i+1} + \bar{q}_2 b\pi_{i-1} + b\pi_i' + a\pi_{i+1}' \qquad \text{for } i \geq 2 \tag{G.13}$$

$$(\bar{d} - \bar{q}_1 a - q_1 c)\pi_i = q_1(a + b)\pi_{i+1} + \bar{q}_1 c\pi_{i+1} + c\pi_i' + a\pi_{i-1}' \qquad \text{for } i \leq -2 \tag{G.14}$$

$$(\bar{d} - \bar{q}_2 a - q_2 b)\pi_1 = q_2(a + c)\pi_2 + b(\pi_{-1}' + \pi_0 + \pi_1') + a\pi_2' \tag{G.15}$$

$$(\bar{d} - \bar{q}_1 a - q_1 c)\pi_{-1} = q_1(a + b)\pi_{-2} + c(\pi_{-1}' + \pi_0 + \pi_1') + a\pi_{-2}' \tag{G.16}$$

$$\bar{d}\pi_i' = \bar{q}_2 c\pi_i + c\pi_{i+1}' \qquad \text{for } i \geq 1 \tag{G.17}$$

$$\bar{d}\pi_i' = \bar{q}_2 c\pi_i + b\pi_{i+1}' \qquad \text{for } i \leq -1 \tag{G.18}$$

Rearranging the terms, we get the following recurrence in the steady-state probabilities on the right-side of the chain,

$$\alpha_3 \pi_{i+3} + \alpha_2 \pi_{i+2} + \alpha_1 \pi_{i+1} + \alpha_0 \pi_i = 0 \qquad \text{for } i \geq 1 \tag{G.19}$$

where,

$$\alpha_3 = c(a + c)q_2 \tag{G.20}$$

$$\alpha_2 = -c\bar{d} + bcq_2 - (a + c)q_2\bar{d} \tag{G.21}$$

$$\alpha_1 = \bar{d}(\bar{d} - bq_2 - a\bar{q}_2) \tag{G.22}$$

$$\alpha_0 = -\bar{d}b\bar{q}_2 \tag{G.23}$$

The characteristic equation of this recurrence has the roots 1, ρ and ρ'. We can show that both ρ and ρ' are positive and at least one of them is greater than 1. The expression for the smaller root is,

$$\rho = -\frac{\alpha_3 + \alpha_2}{2\alpha_3} - \frac{\sqrt{(\alpha_3 + \alpha_2)^2 + 4\alpha_3\alpha_0}}{2\alpha_3} \tag{G.24}$$

The right-side of the Markov chain is stable if and only if $\rho < 1$. Thus, when $\rho < 1$,

194

the steady-state probabilities π_i and π'_i for $i \geq 1$ are related by the recurrences,

$$\frac{\pi_{i+1}}{\pi_i} = \rho \text{ and } \frac{\pi'_i}{\pi_i} = \frac{c(1-q_2)}{\bar{d}-c\rho} \qquad \text{(G.25)}$$

Similarly, for the left side of the chain we have the recurrences,

$$\frac{\pi_{i+1}}{\pi_i} = \mu \text{ and } \frac{\pi'_i}{\pi_i} = \frac{b(1-q_1)}{\bar{d}-b\mu} \qquad \text{(G.26)}$$

where

$$\mu = -\frac{\beta_3+\beta_2}{2\beta_3} - \frac{\sqrt{(\beta_3+\beta_2)^2+4\beta_3\beta_0}}{2\beta_3} \qquad \text{(G.27)}$$

with the expressions β_k for $k = 0, 1, 2, 3$ being the same as α_k with b and c interchanged, and q_2 replaced by q_1. We can use these recurrences we can express all steady-state probabilities π_i and π'_i for $i \geq 1$ in terms of π_1, and the steady-state probabilities π_i and π'_i for $i \leq -1$ in terms of π_{-1}. Then using the states transition equation (G.12), and the fact that all the steady state probabilities sum to 1, we can solve for all the steady-state probabilities of the Markov chain.

User U_1 receives an innovative in every successful slot except when the source (with probability q_2) gives priority to U_2 in states i, $i \geq 1$. Thus, if $\rho < 1$ its throughput is given by

$$\tau_1 = p_1 \left(1 - q_2 \sum_{i=1}^{\infty} \pi_i\right) = p_1 \left(1 - \frac{q_2\pi_1}{1-\rho}\right) \qquad \text{(G.28)}$$

Similarly if $\mu < 1$ we have,

$$\tau_2 = p_2 \left(1 - q_1 \sum_{i=-\infty}^{-1} \pi_i\right) = p_2 \left(1 - \frac{q_1\pi_{-1}}{1-\mu}\right) \qquad \text{(G.29)}$$

Similar to the proof of Lemma 18, we determine the inter-delivery exponent λ_2 of user U_2 by adding an absorbing state F to the Markov chain as shown in Fig. G-2, such that the system transitions to F when an in-order packet is decoded by U_2. In
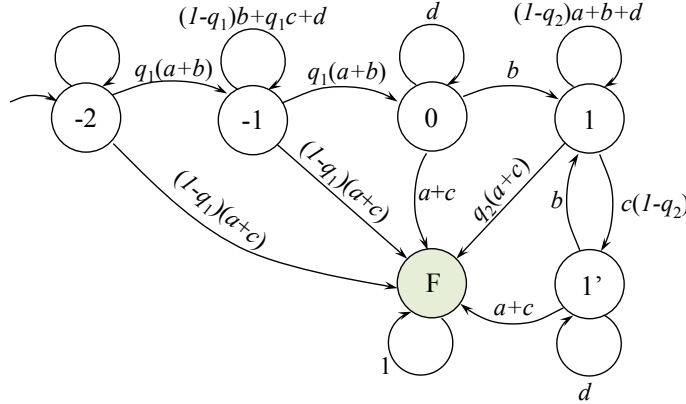
Figure G-2: Markov model used to determine the inter-delivery exponent λ_2 of user U_2. The absorbing state F is reached when an in-order packet is decoded by U_2. The exponent of the distribution of the time taken to reach this state is λ_2.

Fig. G-2, all the states i and i' for $i \geq 1$ are fused into states I and I' because this does not affect the probability distribution of the time to reach the absorbing state F. The inter-delivery exponent $\lambda_2 = -\log \xi_2$ where ξ_2 is the second largest eigenvalue of its state transition matrix of this Markov chain which is given by,

$$A = \begin{pmatrix} d & b & 0 & 0 & a+c \\ 0 & \bar{q}_2 a + b + d & c\bar{q}_2 & 0 & q_2(a+c) \\ 0 & b & d & 0 & a+c \\ q_1(a+b) & 0 & 0 & d + q_1 c + \bar{q}_1 b & \bar{q}_1(a+c) \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (\text{G.30})$$

Solving for the second largest eigen-value ξ_2 of A, we get

$$\xi_2 = \max\left(d + q_1 c + \bar{q}_1 b, \frac{\left(2d + \bar{q}_2 a + b + \sqrt{(2d + \bar{q}_2 a + b)^2 - 4(d(b+d) + \bar{q}_2(da - bc))}\right)}{2} \right).$$
$$(\text{G.31})$$

The inter-delivery exponent $\lambda_2 = -\log \xi_2$ and is given by (9.4). The expression for the inter-delivery exponent λ_1 of user U_1 is same as (9.4) with b and c, and q_1 and q_2 interchanged.

$\square$

Bibliography

[1] Nasuni, "The State of the Cloud Industry Report." `http://www.ciosummits.com/2013_Nasuni_CSP_Report.pdf`, 2013.

[2] R. G. Gallager, *Information theory and reliable communication.* Wiley, 1968.

[3] E. Berkelamp, *Algebraic coding theory.* New York, USA: McGraw-Hill, 1968.

[4] D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *Proceedings of the ACM SIGMOD*, vol. 17, pp. 109–116, June 1988.

[5] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, pp. 4539–4551, Sept. 2010.

[6] N. Shah, K. V. Rashmi, P. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Transactions on Information Theory*, vol. 58, pp. 2134–2158, Apr. 2012.

[7] J. Dean and L. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[8] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands I," *SIAM Journal on Applied Mathematics*, vol. 44, no. 5, pp. 1041–1053, 1984.

[9] R. Nelson and A. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE Transactions on Computers*, vol. 37, pp. 739–743, Jun. 1988.

[10] E. Varki, A. Merchant, and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks," *unpublished, available online*, 2008.

[11] A. Rawat, D. Papailiopoulos, A. Dimakis, and S. Vishwanath, "Locality and availability in distributed storage," in *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, pp. 681–685, June 2014.

[12] S. Kadhe, E. Soljanin, and A. Sprintson, "Analyzing the download time of availability codes," Jun 2015.

[13] J. Cloud, D. J. Leith, and M. Médard, "A Coded Generalization of Selective Repeat ARQ," *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015.

[14] M. Karzand, D. J. Leith, J. Cloud, and M. Médard, "Low Delay Random Linear Coding Over a Stream," *arXiv [cs.it] 1509.00167*, Sept. 2015.

[15] F. Wu, Y. Sun, Y. Yang, K. Srinivasan, and N. Shroff, "Constant-delay and constant-feedback moving window network coding for wireless multicast: Design and asymptotic analysis," *IEEE Journal on Selected Areas in Communications*, vol. 33, pp. 127–140, Feb. 2015.

[16] G. Joshi, E. Soljanin, and G. Wornell, "Efficient replication of queued tasks for latency reduction in cloud systems," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, Oct. 2015.

[17] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *arXiv:1508.03599*, Aug. 2015.

[18] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *Proceedings of ACM SIGMETRICS*, June 2014.

[19] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," in *Proceedings of the ACM SIGMETRICS Distributed Cloud Computing Workshop*, June 2015.

[20] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, pp. 326–333, Oct. 2012.

[21] G. Joshi, Y. Liu, and E. Soljanin, "On the Delay-storage Trade-off in Content Download from Coded Distributed Storage," *IEEE Journal on Selected Areas on Communications*, May 2014.

[22] G. Joshi, E. Soljanin, and G. Wornell, "Queues with redundancy: Latency-cost analysis," in *Proceedings of the ACM SIGMETRICS Workshop on Mathematical Modeling and Analysis*, June 2015.

[23] G. Joshi, Y. Kochman, G. Wornell, "The Effect of Block-Wise Feedback on the Throughput-Delay Trade-off in Streaming," Apr. 2014.

[24] G. Joshi, Y. Kochman, G. Wornell, "Throughput-Smoothness Trade-offs in Multicasting of an Ordered Packet Stream," in *Proceedings of the International Symposium on Network Coding*, June 2014.

[25] G. Joshi, Y. Kochman, G. Wornell, "Throughput-Smoothness Trade-offs in Streaming Communication," *arXiv:1511.08143*, Nov. 2015.

[26] D. Menasce, D. Saha, S. Porto, V. Almeida, and S. Tripathi, "Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures," *Journal of Parallel and Distributed Computing*, vol. 28, pp. 1–18, July 1995.

[27] M. Maheswaran, S. Ali, H. J. Siegal, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW)*, pp. 30–44, 1999.

[28] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM*, vol. 46, pp. 720–748, Sept. 1999.

[29] M. Mitzenmacher, *The power of two choices in randomized load balancing.* PhD thesis, University of California Berkeley, CA, 1996.

[30] M. Mitzenmacher, B. Prabhakar, and D. Shah, "Load balancing with memory," in *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[31] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wijckoff, "Charlotte: Metacomputing on the web," *Journal on Future Generation Computing Systems - Special issue on metacomputing*, vol. 15, pp. 559–570, Oct. 1999.

[32] W. Cirne, F. Brasileiro, D. Paranhos, L. Fabrício W. Góes, and W. Voorsluys, "On the efficacy, efficiency and emergent behavior of task replication in large distributed systems," *Parallel Computing*, vol. 33, no. 3, pp. 213–234, 2007.

[33] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun. Mag.*, vol. 51, pp. 107–113, Jan. 2008.

[34] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, pp. 185–198, Apr. 2013.

[35] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pp. 69–84, 2013.

[36] N. F. Maxemchuk, "Dispersity routing," *Proceedings of the International Conference on Communications (ICC)*, pp. 10–13, Jun. 1975.

[37] G. Kabatiansky, K. E., and S. S., *Error correcting coding and security for data networks: analysis of the superchannel concept.* Wiley, 1st ed., Mar. 2005.

[38] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 283–294, 2013.

[39] N. Shah, K. Lee, and K. Ramachandran, "The MDS queue: Analyzing the Latency Performance of Erasure Codes," in *Proceedings on the IEEE International Symposium on Information Theory*, July 2014.

[40] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Reducing latency via redundant requests: Exact analysis," in *Proceedings of the ACM SIGMETRICS*, Jun. 2015.

[41] A. Kumar, R. Tandon, and T. C. Clancy, "On the latency of heterogeneous mds queue," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pp. 2375–2380, Dec. 2014.

[42] Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *SIGMETRICS Performance Evaluation Review*, vol. 42, pp. 3–14, Sept. 2014.

[43] G. Liang and U. Kozat, "TOFEC: Achieving Optimal Throughput-Delay Tradeoff of Cloud Storage Using Erasure Codes," Apr. 2014.

[44] S. Chen, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, Y. Sun, and N. B. Shroff, "When Queueing Meets Coding: Optimal-Latency Data Retrieving Scheme in Storage Clouds," Apr. 2014.

[45] J. Cao and Y. Wang, "The NBUC and NWUC classes of Life Distributions," *Journal of Applied Probability*, pp. 473–479, 1991.

[46] G. Koole and R. Righter, "Resource allocation in grid computing," *Journal of Scheduling*, vol. 11, pp. 163–173, June 2008.

[47] Y. Kim, R. Righter, and R. Wolff, "Job replication on multiserver systems," *Advances in Applied Probability*, vol. 41, pp. pp. 546–575, June 2009.

[48] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, Oct. 2013.

[49] Y. Sun, Z. Zheng, C. E. Koksal, K. Kim, and N. B. Shroff, "Provably delay efficient data retrieving in storage clouds," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015.

[50] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing (extended version)," *arXiv:1503.03128 [cs.dc]*, Mar. 2015.

[51] C. Reiss, A. Tumanov, G. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, 2012.

[52] M. Bagnoli and T. Bergstrom, "Log-concave probability and its applications," *Economic Theory*, vol. 26, no. 2, pp. pp. 445–469, 2005.

[53] A. M. Lee and P. A. Longton, "Queueing process associated with airline passenger check-in," *Operations Research Quarterly*, pp. 56–71, 1959.

[54] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[55] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, p. 10, 2010.

[56] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[57] W. Neiswanger, C. Wang, and E. Xing, "Asymptotically exact, embarrassingly parallel MCMC," *arXiv:1311.4780 [cs, stat]*, Nov. 2013.

[58] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pp. 15–28, 2012.

[59] Apache Software Foundation, "Apache spark configuration - scheduling (version 1.5.2)." https://spark.apache.org/docs/1.5.2/configuration.html. Accessed: 2016-01-03.

[60] D. Wang, *Computing with Unreliable Resources: Design, Analysis and Algorithms*. PhD thesis, Massachusetts Institute of Technology, 2014.

[61] S. Kochar and D. Wiens, "Partial orderings of life distributions with respect to their aging properties," *Naval Research Logistics*, vol. 34, no. 6, pp. 823–829, 1987.

[62] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster data trace document (version 2)." http://code.google.com/p/googleclusterdata/wiki/TraceVersion2. Accessed: 2014-03-01.

[63] B. Efron and R. Tibshirani, "Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy," *Statistical science*, pp. 54–75, 1986.

[64] M. J. D. Powell, "A view of algorithms for optimization without derivatives," *Cambridge University Technical Report*, no. DAMTP 2007/NA03, 2007.

[65] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[66] K. Gardner, S. Zbarsky, M. Harchol-Balter, and A. Scheller-Wolf, "Analyzing response time in the redundancy-d system," in *CMU-CS-15-141 archive*, Dec. 2015.

[67] Y. Sun, C. E. Koksal, and N. B. Shroff, "On delay-optimal scheduling in queueing systems with replications," *arXiv:1603.07322*, Mar. 2016.

[68] K. Lee, R. Pedarsani, and K. Ramchandran, "On Scheduling Redundant Requests with Cancellation Overheads," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, Oct. 2015.

[69] F. Poloczek and F. Ciucu, "Contrasting effects of replication in parallel systems: From overload to underload and back," *arXiv:1602.07978*, Feb. 2016.

[70] Q. Xie and Y. Lu, "Priority algorithm for near-data scheduling: Throughput and heavy-traffic optimality," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015.

[71] S. Li, M. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, Oct. 2015.

[72] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, July 2016.

[73] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25*, pp. 1223–1231, 2012.

[74] M. Y. An, "Log-concave probability distributions: Theory and statistical testing," tech. rep., Duke University, Department of Economics, Nov. 1995.

[75] R. Proschan and R. Pyke, "Tests for Monotone Failure Rate," in *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, pp. 292–312, 1967.

[76] B. Epstein, "Tests for the Validity of the Assumption That the Underlying Distribution of Life Is Exponential Part I," *Technometrics*, vol. 2, pp. 83–101, Feb. 1960.

[77] B. Epstein, "Tests for the Validity of the Assumption That the Underlying Distribution of Life Is Exponential Part II," *Technometrics*, vol. 2, pp. 167–183, May 1960.

[78] P. Bickel and K. Doksum, "Tests on Monotone Failure Rate Based on Normalized Spacings," *The Annals of Mathematical Statistics*, vol. 40, pp. 1216–1235, 1969.

[79] Amazon EBS. `http://aws.amazon.com/ebs/`.

[80] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29–43, 2003.

[81] Dropbox. `http://www.dropbox.com/`.

[82] Google Docs. `http://docs.google.com/`.

[83] A. G. Dimakis, P. B. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Proceedings of IEEE INFOCOM*, pp. 2000–2008, May 2007.

[84] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

[85] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Proceedings of the International Workshop on Peer-to-Peer Systems*, pp. 226–239, Feb. 2005.

[86] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," *Proceedings of the Allerton Conference on Communication, Control and Computing*, pp. 1243 – 1249, Sep. 2009.

[87] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: necessity and code constructions," vol. 58, pp. 2134–2158, Apr. 2012.

[88] I. Tamo, Z. Wang and J. Bruck, "Zigzag Codes: MDS Array Codes With Optimal Rebuilding," *IEEE Transactions on Information Theory*, vol. 59, no. 3, pp. 1597–1616, 2013.

[89] U. Ferner, M. Médard, and E. Soljanin, "Toward sustainable networking: Storage area networks with network coding," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, pp. 517–524, Oct. 2012.

[90] L. Huang, S. Pawar, H. Zhang, and Kannan Ramchandran, "Codes can reduce queueing delay in data centers," in *Proceedings of the IEEE International Conference on Information Theory (ISIT)*, pp. 2766–2770, July 2012.

[91] Y. Liu, J. Yang, and S. C. Draper, "Exploiting route diversity in multi-packet transmission using mutual information accumulation," *Allerton Conference on Communication, Control and Computing*, pp. 1793–1800, Sep. 2011.

[92] L. Xu, *Highly Available Distributed Storage Systems.* PhD thesis, California Institute of Technology, 1998.

[93] E. Soljanin, "Reducing delay with coding in (mobile) multi-agent information transfer," *Allerton Conference on Communication, Control and Computing*, pp. 1428–1433, Sep. 2010.

[94] C. Kim and A. K. Agrawala, "Analysis of the Fork-Join Queue," *IEEE Transactions on Computers*, vol. 38, pp. 250–255, Feb. 1989.

[95] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," in *IEEE Transactions on Information theory*, vol. 58, pp. 6925–6934, Nov. 2012.

[96] D. Papailiopoulos and A. Dimakis, "Locally repairable codes," *IEEE Transactions on Information Theory*, vol. 60, pp. 5843–5855, oct 2014.

[97] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *ACM symposium on Theory of computing*, (New York, NY, USA), pp. 150–159, ACM, 1997.

[98] D. Mackay, "Fountain codes," in *Proceedings of IEE Communications*, vol. 152, pp. 1062–1068, Dec. 2005.

[99] Sandvine Intelligent Networks, "Global internet phenomena report." `http://www.sandvine.com/downloads/documents/Phenomena_1H_2013/Sandvine_Global_Internet_Phenomena_Report_1H_2013.pdf`, Mar. 2013.

[100] E. Martinian, *Dynamic Information and Constraints in Source and Channel Coding*. PhD thesis, MIT, Cambridge, USA, Sept. 2004.

[101] A. Badr, A. Khisti, W. Tan and J. Apostoupoulos, "Robust Streaming Erasure Codes based on Deterministic Channel Approximations," in *Proceedings of the International Symposium on Information Theory (ISIT)*, July 2013.

[102] P. Patil, A. Badr, A. Khisti and W. Tan, "Delay-Optimal Streaming Codes under Source-Channel Rate Mismatch," in *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, Nov. 2013.

[103] H. Yao, Y. Kochman and G. Wornell, "A Multi-Burst Transmission Strategy for Streaming over Blockage Channels with Long Feedback Delay," *IEEE Journal on Selected Areas in Communications*, Dec. 2011.

[104] G. Joshi, Y. Kochman, G. Wornell, "On Playback Delay in Streaming Communication," in *Proceedings of the IEEE International Symposium on Information Theory*, July 2012.

[105] G. Joshi, *On Playback Delay in Streaming Communication.* Masters thesis, Massachusetts Institute of Technology, 2012.

[106] T. Cover and J. Thomas, *Elements of information theory.* New York, NY, USA: Wiley-Interscience, 2nd ed., 1991.

[107] L. Keller, E. Drinea and C. Fragouli, "Online Broadcasting with Network Coding," in *Proceedings of the IEEE Network Coding Theory and Applications*, pp. 1–6, Jan. 2008.

[108] J. Barros, R. Costa, D. Munaretto, and J. Widmer, "Effective Delay Control in Online Network Coding," in *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pp. 208–216, Apr. 2009.

[109] A. Fu, P. Sadeghi, and M. Medard, "Delivery delay analysis of network coded wireless broadcast schemes," in *Proceedings of the Wireless Communications and Networking Conference (WCNC)*, pp. 2236–2241, 2012.

[110] J. Sundararajan, P. Sadeghi, and M. Médard, "A feedback-based adaptive broadcast coding scheme for reducing in-order delivery delay," in *IEEE Workshop on Network Coding, Theory, and Applications*, pp. 1–6, 2009.

[111] J. Sundararajan, D. Shah and M. Médard, "Online network coding for optimal throughput and delay: the three-receiver case," in *International Symposium on Information Theory and its Applications*, Dec. 2008.

[112] Y. E. Sagduyu and A. Ephremides, "On broadcast stability region in random access through network coding," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, pp. 143–150, 2006.

[113] K. Mahadaviani, A. Khisti, G. Joshi, and G. Wornell, "Playback Delay in On-Demand Streaming Communication with Feedback," in *Proceedings of the International Symposium on Information Theory (ISIT)*, July 2015.

[114] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, pp. 2856–2867, May 2014.

[115] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. Diggavi, "Hierarchical coded caching," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pp. 2142–2146, June 2014.

[116] U. Niesen and M. A. Maddah-Ali, "Coded caching for delay-sensitive content," in *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 5559–5564, 2015.

[117] A. ParandehGheibi, M. Medard, A. Ozdaglar, and S. Shakkottai, "Avoiding Interruptions – A QoE Reliability Function for Streaming Media Applications," *IEEE Journal on Selected Areas in Communications*, vol. 29, pp. 1064–1074, May 2011.

[118] H. A. David and H. N. Nagaraja, *Order statistics*. Hoboken, N.J.: John Wiley, 2003.

[119] L. de Haan and A. Ferreira, *Extreme value theory an introduction*. New York: Springer, 2006.

[120] R. Gallager, *Discrete Stochastic Processes*. Kluwer Academic Publishers, 2nd ed., 2013.

[121] J. Esary, F. Proschan, and D. Walkup, "Association of random variables, with applications," *Annals of Mathematics and Statistics*, vol. 38, pp. 1466–1474, Oct. 1967.

[122] R. Durrett, *Probability: Theory and Examples*. Cambridge University Press, 4th ed., 2010.