Computing with Unreliable Resources: Design, Analysis and Algorithms

by

Da Wang

B.A.Sc., Electrical Engineering
University of Toronto, 2008

S.M., Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2010

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 2014

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2014

Certified by .....
Gregory W. Wornell
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Computing with Unreliable Resources:
Design, Analysis and Algorithms

by

Da Wang

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis is devoted to the study of computing with unreliable resources, a paradigm emerging in a variety of technologies, such as circuit design, cloud computing, and crowdsourcing. In circuit design, as we approach the physical limits, semiconductor fabrication has been increasingly susceptible to fabrication flaws, resulting unreliable circuit components. In cloud computing, due to co-hosting, virtualization and other factors, the response time of computing nodes are variable. This calls for computation frameworks that take this unreliable quality-of-service into account. In crowdsourcing, we humans are the unreliable computing processors due to our inherent cognitive limitations.

We investigate these three topics in the three parts of this thesis. We demonstrate that it is often necessary to introduce redundancy to achieve reliable computing, and this needs to be carried out judiciously to attain an appealing balance between reliability and resource usage. In particular, it is crucial to take the statistical properties of unreliability into account during system design, rather than to handle it as an afterthought.

In the first part, we investigate the topic of circuit design with unreliable circuit components. We first analyze the design of Flash Analog-to-Digital Converter (ADC) with imprecise comparators. Formulating this as a problem of scalar quantization with noisy partition points, we analyze fundamental limits on ADC accuracy and obtain designs that increase the yield of ADC (e.g., 5% to 10% for 6-bit Flash ADCs). Our results show that, given a fixed amount of silicon area, building more smaller and less precise comparators leads to better ADC accuracy. We then address the problem of digital circuit design with faulty components. To achieve reliability, we introduce redundant elements that can replace faulty elements via a configurable interconnect. We show that the required number of redundant elements depends on the amount of interconnect available, and propose designs that achieve near-optimal trade-off between redundancy and interconnect overhead in several design settings.

The second part of this thesis explores the problem of executing a collection of tasks in parallel on a group of computing nodes. This setting is often seen in cloud computing and crowdsourcing, where the response times of computing nodes are random due to their variability. In this case, the overall latency is determined by the response time of the slowest computing node, which is often much larger than the average response time. Task replication, which sends the same task to multiple computing nodes and obtains the earliest result, reduces latency, but in general incurs additional resource usage. We propose a theoretical framework to analyze the trade-off between latency and resource usage. We

show that, while in general there is a tension between latency and resource usage, there exist scenarios where replicating tasks judiciously reduce both latency and resource usage simultaneously. Our investigation gives insights on when and how replication helps, and provides efficient scheduling policies for a variety of computing scenarios.

Lastly, we research the problem of crowd-based ranking via pairwise comparisons, with humans as unreliable comparators. We formulate this as the problem of approximate sorting with noisy comparisons. By developing a rate-distortion theory on permutation spaces, we obtain information-theoretic lower bounds for the query complexity of approximate sorting with both noiseless and noisy comparisons. Our lower bound shows the optimality of certain existing algorithms with respect to noiseless comparisons and provides a benchmark for approximate sorting with noisy comparisons.

Thesis Supervisor: Gregory W. Wornell
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

If you are lucky enough to have studied at MIT as a young man, then wherever you go for the rest of your life, it stays with you, for MIT is a moveable feast.

Adapted from Ernest Hemingway, A Moveable Feast

It has been an unforgettable experience to pursue my Ph.D at MIT, and there are many people I am truly thankful for during this journey.

First and foremost, I want to thank my advisor Greg Wornell for his support, encouragement and guidance. The past six years have been a particularly exciting phase of my professional development, during which I develop my taste of problems, learn how to ask the right questions, and become a better researcher. None of these would be possible without his mentorship. Besides Greg, I am also privileged to have Yury Polyanskiy and Devavrat Shah as my thesis committee members. Starting from a paper reading seminar, I collaborate with Yury on the reliable circuit design problem. This fun collaboration has been a great learning experience, and eventually leads to a part of my thesis. From Devavrat, I receive many insightful comments regarding both technical results and presentation, and I just wish I have more time to learn from him!

In addition to my thesis committee, I would like to thank my collaborators during various stages of my grad school career. All these collaborations are productive and enjoyable, and I can only attribute that to my good luck and the great intellectual and emotional capacities of my collaborators: Venkat Chandar, Sae-Young Chung, Amir Ingber, Gauri Joshi, Yuval Kochman, Arya Mazumdar and Hongchao Zhou. I also want to acknowledge Ligong Wang, Frank Yaul and Anantha Chandrakasan for helpful discussions.

Being a member of the Signals, Information and Algorithms Laboratory, I am surrounded by a group of intelligent and supportive people, and it is everyone of them that makes this group awesome: Anthony Accardi, Mo Deng, Vijay Divi, Qing He, Ying-zong Huang, Gauri Joshi, Ashish Khisti, James Krieger, Emin Martinian, Urs Niesen, David Romero, Gal Shulkind, Maryam Shanechi, Ligong Wang, Atulya Yellepeddi, Xuhong Zhang, Hongchao Zhou, and of course our great administrative assistant Tricia O'Donnell. In particular, I thank Emin for the internship with him at Bain Capital as well as many helpful career advices, Anthony for the internship with him at Swipely, Maryam for sharing her perspectives, Gauri for being a great listener to my complaints, Hongchao for often being the other guy working in the office suite, and Ligong for always being available for technical and non-technical discussions.

I also benefit greatly from my interaction with the larger Research Laboratory of Electronics community, especially the RLE_5^6. Among all the wonderful members, I want to especially thank Vivek Goyal for his semi-annual and superbowl parties, Lizhong Zheng for inspiring me to run the first marathon in my life, Al Oppenheim for sharing his wisdom, and John Sun for valuable advices and helpful discussions. Beyond RLE, I have the great fortune to interact with countless bright young men and women in the MIT community. Among them, I want to especially thank Vincent Tan for his belief in me as an information theorist, and Mario Lok and Weifei Zeng for being delightful roommates.

All these MIT experiences are fascinating, but they would not be possible without those who introduced me to the world of intellectual discovery at the University of Toronto: Frank Kschischang, Ben Liang, Danilo Silva and Wei Yu. It was my great fortunate to meet them, and I am always grateful for their encouragement and guidance.

Finally, I want to thank my parents for their unconditional love and support, and Hengni for her patience and the energy she brings to my life. This journey is at its end, and I look forward to the next one with all of you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

If the world were perfect, it wouldn't be.

Yogi Berra

In modern computing, we increasingly face the challenge of processing enormous amount of data. Handling a large amount of data often requires a large amount of computing resources, and at a large scale, it is often economically inviable or even technologically infeasible to maintain the reliability of each individual computing unit. These constraints pose the challenge of *computing with unreliable resources*, where computing units could be unreliable in a variety of ways.

The paradigm of computing with unreliable resources can be observed in many technologies or applications, such as VLSI circuit fabrication, cloud computing, and crowdsourcing. In VLSI circuit fabrication, as the sizes of transistors shrink, the issue of fabrication flaws become more severe. As a result, fabrication yield, the percentage of chips that meet the fabrication specifications, decreases. In cloud computing, due to co-hosting, virtualization and other factors, the response times of computing nodes are variable, even when they all have the same configuration. Furthermore, computer failures are frequent due to the scales of data centers. This calls for computation frameworks that take this varying quality-of-service into account to achieve robust and efficient computation. Finally, in crowdsourcing, we are essentially using humans as unreliable processors to solve a relatively complex problem by asking questions with answers in simple forms. This poses challenges to both question design and answer aggregation.

All these scenarios involve building a reliable and efficient system with unreliable components, a characteristic shared by the problem of communication in noisy environments. For communication, information theory provides the key insight that by taking the statistical property of the noisy channel into account, we can introduce redundancy (coding) efficiently and achieve reliable communication, as pointed out by Shannon in his seminal paper [1]. In this thesis, we investigate how to introduce redundancy for a few computing systems, and seek to answer the following two questions:

1. What is the fundamental trade-off between redundancy and performance (accuracy, reliability, etc.)?

2. What is the proper way to introduce redundancy so that we can handle the unreliable components in the system efficiently?

In particular, we analyze three applications in the three parts of this thesis. The first one is *reliable circuit design with fabricators flaws*. We investigate the problem of designing a Flash ADC with imprecise comparators, which provides an example of handling the

issue of process variation in VLSI circuit fabrication. In addition, we analyze the problem of designing a reliable digital circuit with faulty components, where faulty components come from fabrication defects. The second application is *scheduling parallel tasks with variable response times*. This type of scheduling problem occurs when we distribute a collection of computation tasks among a group of computing nodes, a phenomena that happens in both data centers and crowd sourcing. For this problem, we aim to reduce the latency of the computation without too much additional resource usage. The third one is *approximate sorting with noisy comparisons*, where we aim to sort a list of items by comparing them in a pairwise fashion so that the end result is close to their ordering. This problem is motivated by crowd-based ranking, a canonical example of crowdsourcing. Our investigations of these three applications show that, by taking the statistical property of unreliability into account, we can introduce redundancy *efficiently* to increase fabrication yield, reduce computation latency and improve sorting accuracy.

A few common tools emerge in our analysis for these three different applications. First, rate-distortion theory provides the framework for evaluating both the ADC design problem in Chapter 3 and the approximate sorting algorithm in Part III. Second, order statistics analysis plays a pivotal role in the ADC performance characterization in Chapter 3, and also the scheduling problem in Part II. Finally, combinatorics and probabilistic methods provide many useful techniques for the analysis in Chapter 4, Chapter 7, and Chapter 9.

The three parts of the thesis are largely independent and can be read separately. Furthermore, within Part I, Chapter 3 and Chapter 4 are largely independent, and Chapter 9 in Part III is mostly self-contained.

1.1 Common notation

In this section we introduce the notation that is used throughout the thesis. Additional problem-specific notation is introduced separately in each part of this thesis.

We use $\mathbb{R}$ to denote all real numbers, $\mathbb{R}_+$ all the non-negative real numbers, $\mathbb{Z}$ all integers, and $\mathbb{Z}^+$ all positive integers.

For any number x, we denote

$$|x|^+ = \max\{0, x\}.$$

We use $[n]$ to denote all positive integers no larger than n, i.e., the set $\{1, 2, \ldots, n\}$.

We use $x_i^j, j > i$ to denote a sequence of values $x_i, x_{i+1}, \ldots, x_j$, and x^n as a shorthand for x_1^n. We use the bold font to represent a vector, e.g., $\mathbf{x} \triangleq [x_1, x_2, \ldots, x_n]$.

We denote the indicator function by $\mathbb{1}\{A\}$, where

$$\mathbb{1}\{A\} = \begin{cases} 1 & \text{clause } A \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

We use the notation $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$, where $f(n) = O(g(n))$ if and only if $\limsup_{n\to\infty} |f(n)/g(n)| < \infty$, $f(n) = \Omega(g(n))$ if and only if $\liminf_{n\to\infty} |f(n)/g(n)| \geq 1$, and $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Probability: we use lower-case letters (e.g., x) to denote a particular value of the corresponding random variable denoted in upper-case letters (e.g., X). We denote the support of a probability density function (p.d.f.) f_X by $\mathrm{Supp}(f_X)$, namely, $\mathrm{Supp}(f_X) \triangleq \{x \in \mathbb{R} : f_X(x) > 0\}$.

We use "w.p." as a shorthand for "with probability", i.i.d. for "independent and identically distributed", p.m.f. for "probability mass function", p.d.f. for "probability density function", and c.d.f. for "cumulative density function".

We use $\mathsf{N}\left(\mu, \sigma^2\right)$ to denote a Gaussian distribution with mean μ and variance σ^2, $\mathsf{Unif}\left([a, b]\right)$ a uniform distribution over an interval $[a, b]$, $\mathsf{Bern}\left(p\right)$ a Bernoulli distribution with parameter p, and $\mathsf{Exp}\left(\lambda\right)$ an Exponential distribution with parameter λ.

Order statistics: for random variables $X_1, X_2, \cdots, X_n$, we define $X_{j:n}$ be the j-th order statistics of $X_i, 1 \leq i \leq n$, i.e., the j-th smallest sample in $\{X_i, 1 \leq i \leq n\}$. For simplicity, we also use $X_{(j)}$ to denote $X_{j:n}$ when the number of random variables n is clear from the context.

Part I

Reliable circuit design with fabrication flaws

Chapter 2

On the challenge of yield in semiconductor fabrication

Any physical quantity that is growing exponentially predicts a disaster. You simply can't go beyond certain major limits.

Gordon Moore, at the 2007 Intel Developer Forum

The semiconductor industry has enjoyed tremendous success in the past fifty years. One highlight of this success is the continued realization of Moore's law [2], which predicts the number of transistors on integrated circuits doubles approximately every two years. However, as transistor sizes get increasingly close to the physical limits, the growth specified by Moore's law is unlikely to be sustainable, as pointed out in [3]. Moreover, approaching the physical limits also makes semiconductor fabrication much more difficult, making a severe impact on the yield of fabrication. It is observed that without tackling the issue of yield, simply scaling up the number of transistors per chip, even if technologically feasible, may not be economically viable [4].

This emerging issue motivates us to consider the problem of reliable circuit design subject to fabrication flaws, which can be broadly categorized into *process variations* and *fabrication defects*. Process variations refer to the variations in the attributes of transistors (size, operating characteristics, etc.) introduced during fabrication [5], while fabrication defects usually refer to transistors with severe performance degradation that can be considered as failed ones. The challenge of process variations is especially salient for analog circuits that rely on reference voltages, such as comparators and inverters, and we investigate the problem of Flash Analog-to-Digital Converter (ADC) design with imprecise comparators in Chapter 3. For the issue of *fabrication defects*, we provide an analysis in Chapter 4 in the context of digital circuit design, where fabricated components fail with a certain probability.

While Moore's law may no longer be sustainable, our results suggest that it is likely that we can continue to improve integrated circuit performance through better circuit designs. The key insight is that we need redundancy to tackle unreliability, and we need to take the statistical properties of fabrication flaws into account to make efficient use of the introduced redundancy. This change may lead to new designs that is different from the traditional designs that do not take unreliability into account.

Chapter 3

Designing Flash ADC with imprecise comparators

3.1 Introduction

In this chapter we investigate *process variation* issue in semiconductor fabrication via an example, the design of Flash ADC design with imprecise comparators, where the reference voltages of comparators are subject to offsets due to process variations.

There exist investigations on the non-idealities of ADCs from both theoretical and practical perspectives. Existing theoretical investigations (cf. [6] and the references therein) aim to improve the quantization (estimation) performance based on a given ADC design via post-processing. However, this approach is not feasible when the process variations are large, because the traditional designs do not meet the performance specifications anymore. In this case, new ADC designs are called for. In the context of Flash ADC design, it is shown that for large process variations, redundancy and reconfigurability achieves a better trade-off between area and linearity [7] than increasing device size, and a series of work in circuit systems [7–10] have explore the topic of Flash ADC design with redundancy and/or reconfigurability. In these circuit design researches, simulations or empirical measurements are used to evaluate performance, which are usually computation or labor intensive. Furthermore, with little theory on the fundamental performance limits, it is unclear that how one can compare different redundancy/reconfiguration implementations. We address these issues by analyzing the performance limits of ADCs under process variations, and propose designs that make better use of redundancy by taking the statistical property of process variations into account.

The problem of analog-to-digital conversion can be formulated as a quantization problem, a subject that has been thoroughly investigated in classical quantization theory [11]. A useful technique in classical quantization theory is high resolution analysis, which approximates a sequence of values by its point density function. This approximation simplifies analysis and provides insights on the optimal quantizer design. In this investigation, we first observe that reference voltages offsets of imprecise comparators correspond to offsets to partition points in a quantization problem, and formulate the problem of ADC design with imprecise comparator as a problem of scalar quantization with noisy partition points, where partition points are perturbed from the designated values during the placement process. Then we extend high resolution analysis to this new formulation, and investigate how the choice of partition point density impacts the quantization performance, in terms of mean square error (MSE), maximum quantization cell size and maximum quantization error. With these results, we provide implications on both ADC design and technology scaling.

We restrict our attention to the static performance of an ADC, i.e., when the input is a DC or slowly varying signal, rather than the dynamic performance. This means we ignore performance measures such as timing errors, signal bandwidth, sample-and-hold errors,

etc.. For the high-speed ADC such as the Flash ADC, this corresponds to the best-case baseline performance without taking dynamic effects into account.

The rest of the chapter is organized as follows. In Section 3.2, we introduce ADC and the Flash ADC architecture, and describe the problem of Flash ADC design with imprecise comparators. Then in Section 3.3, we first review the classical scalar quantization problem setup and then formulate the problem of scalar quantization with noisy partition points in Section 3.3.2. Then we extend the traditional high resolution analysis technique to the case of noisy partition points in Section 3.4, and analyze the high resolution performance in terms of MSE, maximum quantization cell size and maximum quantization error. In Section 3.5, we show these results are accurate in the practical regimes of interest for Flash ADC design, and derive implications on both ADC design and technology scaling. Finally, we provide some concluding remarks in Section 3.6. We defer most proofs and derivations in this chapter to Appendix A.

Notation

In addition to the notation introduced in Section 1.1, we introduce the following notation in this chapter.

We let $\langle f, g\rangle$ denote the inner product of two functions on $\mathbb{R}$, i.e., for $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, $\langle f, g\rangle \triangleq \int_{x\in\mathbb{R}} f(x)g(x)d\,x$.

Given a sequence c^n, we denote the number of points in c^n that falls in an interval $[a, b]$ by $N(a, b; c^n)$, i.e.,

$$N(a, b; c^n) \triangleq \sum_{i=1}^{n} \mathbb{1}\{a \le c_i \le b\}. \tag{3.1}$$

We say $a_n \simeq b_n$ if $a_n = b_n(1 + \varepsilon_n)$ for some $\varepsilon_n \to 0$ as $n \to \infty$.

3.2 Background

In this section we introduce ADC and the Flash ADC architecture, and describe the problem of Flash ADC design with imprecise comparators, which motivates the problem formulation in Section 3.3.2.

Most ADC designs are concerned with uniformly distributed input X, in which case the optimal reference voltages should form an evenly spaced grid in the input range. In this section we we restrict our attention to these uniform ADCs and introduce the corresponding definitions and performance metrics. We remove this constraint later in Section 3.3, where we define the problem of quantization with respect to general input distributions.

3.2.1 ADC and performance metrics

An *Analog-to-Digital Converter* ADC is a device that converts analog (continuous-valued) signals, usually voltages, to digital (discrete-valued) signals. This conversion process is also known as *quantization*. Being the interface from the analog world to the digital world, ADCs are essential parts in devices for communication, imaging, media and measurement.

Given an input signal v_{in} in the ADC input range $[v_{\text{lo}}, v_{\text{hi}}]$, a b-bit ADC produces a corresponding b-bit output code, based on the comparison of v_{in} with its reference voltages,

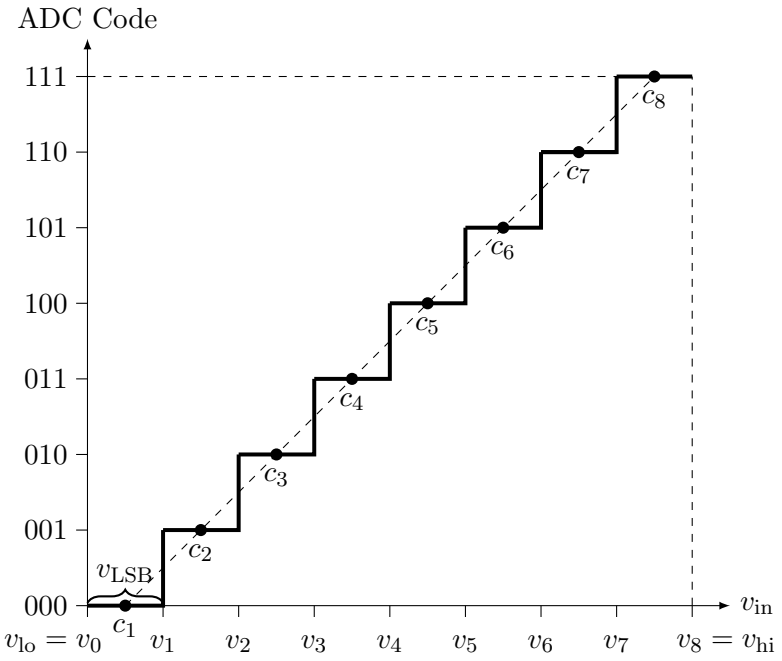


Figure 3-1: The relationship between input v_{in} and output ADC code of an ideal 3-bit uniform ADC. The black dots indicate reproduction point of each output code.

$v_1, v_2, \ldots, v_n, n = 2^b - 1$. Conceptually, each output code corresponds to a *reproduction value* $c_i, 1 \leq i \leq 8$, which represents an estimate of the input value given the output code. We say the *full-scale range* of an ADC is $v_{\text{FSR}} \triangleq v_{\text{hi}} - v_{\text{lo}}$, and call the minimum voltage change corresponding to an output code change the *least significant bit (LSB) voltage* of an ADC, which is $v_{\text{FSR}}/2^b$. In addition, for convenience, we often denote $v_0 \triangleq v_{\text{lo}}$ and $v_{n+1} \triangleq v_{\text{hi}}$.

The input-output relationship of an ideal 3-bit uniform ADC is shown in Fig. 3-1, which corresponds to $b = 3$, $n = 7$, and $c_i = v_{\text{lo}} + (i - 1/2)v_{\text{LSB}}, 1 \leq i \leq n + 1$.

As mentioned in Section 3.1, in this chapter we focus on the static performance of ADCs. The commonly-used static performance metrics for ADC include *mean-square error* (MSE), *differential nonlinearity* (DNL), and *integral nonlinearity* (INL).

Given an input X, MSE is a measure of the average square distortion between input value X and its reproduced value $\hat{X}$:

$$\text{MSE} = \mathbb{E}_X \left[(X - \hat{X})^2 \right]. \tag{3.2}$$

When X is uniformly distributed, the MSE for the ideal b-bit uniform ADC is simply $v_{\text{FSR}}^2/(12(n+1)^2)$.

In addition to the average-case performance measure MSE, DNL and INL are proposed to measure the worst-case performance of an ADC. DNL [12] is the difference between the

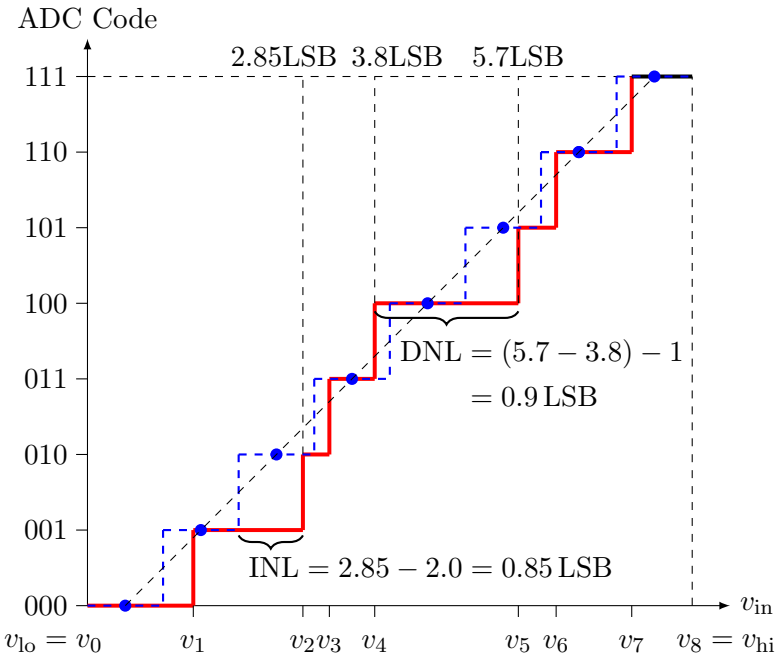


Figure 3-2: DNL and INL of a non-ideal 3-bit uniform ADC. The solid curve represents the actual IO relationship of the ADC, and the dashed curve represents the ideal IO relationship of the ADC.

largest actual quantization "step width" and the ideal "step width" v_{LSB}[1], while INL [12] is the largest deviation of code transition from its ideal location:

$$\text{DNL} \triangleq \max_{0 \le i \le n} \left(v_{i+1} - v_i - v_{\text{LSB}} \right), \tag{3.3}$$

$$\text{INL} \triangleq \max_{1 \le i \le n} \left| v_i - v_{i,\text{ideal}} \right|, \tag{3.4}$$

where $v_{i,\text{ideal}} = v_{\text{lo}} + i \cdot v_{\text{LSB}}$ in the case of uniform ADC[2].

We often normalize DNL and INL by v_{LSB} and quote them in terms of LSB. In the ideal case shown in Fig. 3-1, DNL and INL are simply 0. An example of the DNL and INL of a non-ideal uniform ADC is shown in Fig. 3-2.

Note that both DNL and INL are independent of the input distribution. Furthermore, Lemma A.1 indicates that the maximum quantization error (difference between v_{in} and

[1] A more refined definition of DNL represent it as a vector, where each element in the vector represent the deviations correspond to each "step width" (code transition location). By contrast, the definition in (3.3) is worst-case in nature, and we adopt this because they are more commonly used in practice due to its simplicity.

[2] The definition of INL in (3.4) corresponds to the so-called "end-point" INL, and there is another INL definition called the "best-fit" INL, where the ideal location of code transitions are decided by fitting a line (in the least-square sense) to the measured code transitions. The latter method leads to a smaller INL value and hence "end-point" INL can be seen as a lower bound to the "best-fit" INL.

its reproduction value)

$$e_{\max} = \max_{1 \le i \le n+1} \max\left\{v_i - c_i, c_i - v_{i-1}\right\}$$

is equal to $\mathrm{INL} + v_{\mathrm{LSB}}/2$ for most values of v^n. Therefore, INL can be seen as a proxy for the maximum quantization error.

In addition to the quantization metrics mentioned above, resource usages such as power and area are also important considerations in ADC design.

■ 3.2.2 The Flash ADC architecture

The problem of ADC design has been well investigated [13, 14] and there exists a variety of ADC architectures that can be broadly divided into three categories, serial ADCs, parallel ADCs (commonly called *Flash ADCs*), and subranging ADCs. Within serial ADC architectures, there are a variety of choices such as Ramp/Integrating, Delta-Sigma, Successive-Approximation (SAR), bit-serial pipelined and algorithmic ADCs.

In this paper we focus on the Flash ADC architecture. It is a high-speed (usually defined as more than 20 MSPS[3]) ADC architecture with a straightforward implementation, as shown in Fig. 3-3b: the input voltage is quantized by $n = 2^b - 1$ comparators with monotonically increasing reference voltages, and the output of these comparators form a thermometer code that is encoded as a b-bit output. With its high speed, Flash ADC is commonly used in high-speed communication, signal processing systems, as well as NAND flash memory. In current Flash ADC designs usually $b \le 8$, due to the size constraint of the circuit.

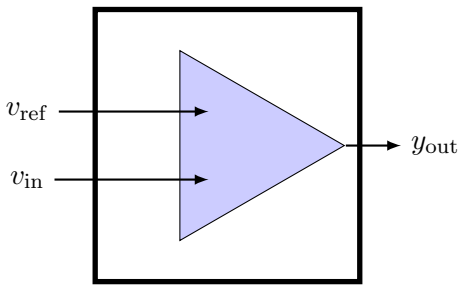
Remark 3.1 (Thermometer code). *A thermometer code represents a natural number k with k ones followed by zeros. It is also known as the* unary code*. For example, given a 3-bit ADC, the thermometer code for the ADC Code 000 (binary representation of 0) is 0000000 as all comparators output 0, while the thermometer code for the ADC Code 101 (binary representation of 5) is 1111100 as all but two comparators outputs 1.*

The decoding of a thermometer code is straightforward. One can simply search from left to right for location of the first 0 in the thermometer code. Alternatively, one can sum up all bits in the thermometer code, which is more complicated but more robust [15].

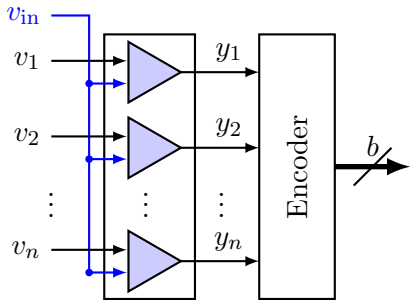
Remark 3.2. *Usually the INL of a Flash ADC is required to be less than 1LSB, with typical value being 0.6LSB [16] or 0.75LSB [17]. The discussion in Section 3.2.1 implies that 0.75LSB in INL essentially corresponds to 1.25LSB maximum quantization error.*

The key building block of a Flash ADC is a bank of comparators, whose outputs indicate the range that the input signal belongs to. As mentioned, in practice the comparators are imprecise due to process variations. As Fig. 3-4a shows, the input-output relationship of an imprecise comparator satisfies $Y_{\mathrm{out}} = \mathbb{1}\left\{V_{\mathrm{in}} + Z_{\mathrm{in}} \ge V_{\mathrm{ref}} + Z_{\mathrm{ref}}\right\}$. Let $Z = Z_{\mathrm{in}} - Z_{\mathrm{ref}}$, then the output satisfies $Y_{\mathrm{out}} = \mathbb{1}\left\{V_{\mathrm{in}} + Z \ge V_{\mathrm{ref}}\right\}$, where $Z \sim \mathsf{N}\left(0, \sigma^2\right)$ is the effective fabrication offset and $\sigma^2 = \sigma_1^2 + \sigma_2^2$ is the effective fabrication variance, as both Z_{in} and Z_{ref} can be modeled as independent zero-mean Gaussian random variables [18, 19].

[3] MSPS means "Million Samples Per Second".

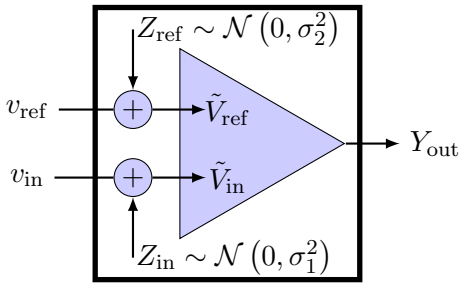


(a) Block diagram of an ideal comparator with $y_{\text{out}} = \mathbb{1}\{v_{\text{in}} > v_{\text{ref}}\}$.

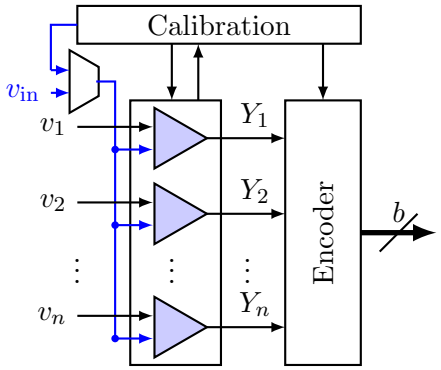


(b) Block diagram for a b-bit Flash ADC, consisting of $n = 2^b - 1$ comparators.

Figure 3-3: Flash ADC with ideal comparators.



(a) Block diagram of an imprecise comparator with $Y_{\text{out}} = \mathbb{1}\left\{\tilde{V}_{\text{in}} > \tilde{V}_{\text{ref}}\right\}$.



(b) Block diagram of a b-bit Flash ADC with calibration, consisting of $n \geq 2^b - 1$ comparators.

Figure 3-4: Flash ADC with imprecise comparators and calibration.

Remark 3.3. *We emphasize that the reference voltages offsets are* static *in the sense that they are determined at the time of fabrication and does not change afterwards.*

To tackle process variations, recently Flash ADC designs with redundancy and possibly reconfiguration are introduced to achieve good conversion resolution in the presence of imprecise comparators [7–10]. By using more than $2^b - 1$ comparators and suitable encoding, these designs improve the resolution and hence the effective number of bits (ENOB) of the ADC. A diagram of flash ADC design with redundancy and calibration proposed in [8] is shown in Fig. 3-4b, where the Calibration block measures the actual noisy fabrication reference voltages. Based on the calibration results, we may choose to disable certain comparators, and use a corresponding encoding scheme, such as summing the output of all comparators.

Remark 3.4 (Finite precision of calibration). *In practice the calibration process is only up to certain level of accuracy as the calibration process itself is an analog-to-digital*

process, and the calibration results, if stored on-chip, can only be stored with finite number of bits. For example, on-chip storage of calibration results with 9-bit accuracy is implemented in [8]. However, since the number of bits for storage is large enough, and the calibration can be done via external high-precision ADCs, we assume calibration is fully accurate as the calibration error can be made negligible in these cases.

■ 3.3 Problem formulation

In this section we abstract the problem of designing flash ADC with imprecise comparators, as described in Section 3.2, as the problem of scalar quantization with noisy partition points. Before proposing the mathematical model and corresponding performance metrics in Section 3.3.2, we first review the classical scalar quantization problem in Section 3.3.1 as our development can be seen as a generalization.

■ 3.3.1 Classical scalar quantization problem

In the classical scalar quantization problem, an m-point *scalar quantizer* Q_m is a mapping $Q_m : \mathbb{R} \to \mathcal{C}$, where $\mathcal{C} = \{c_1, \ldots, c_{m-1}, c_m\} \subset \mathbb{R}$ is the set of *reproduction values*. A quantizer $Q_m(x; v^n, \mathcal{C})$ is uniquely determined by its reproduction values $\mathcal{C}$ and its *partition points* v^n, where an input x is mapped to a value in $\mathcal{C}$ based on the *quantization cell* $(v_{i-1}, v_i]$ that x falls into. Given a scalar quantizer, we define the *quantization error* of an input x as

$$e(x; v^n, \mathcal{C}) \triangleq |Q_m(x; v^n, \mathcal{C}) - x| . \tag{3.5}$$

Given an input X with p.d.f. f_X, the MSE defined in (3.2) becomes

$$D(v^n, \mathcal{C}) \triangleq \mathbb{E}_X [d(X; v^n, \mathcal{C})] ,$$

where $d(\cdot; \cdot, \cdot)$ is the square error function

$$d(x; v^n, \mathcal{C}) \triangleq (x - Q_m(x; v^n, \mathcal{C}))^2 .$$

When the input is not uniformly distributed, the definition of LSB is less clear, making the definitions of DNL and INL not as meaningful. In this section we define the following two error metrics that do not rely on the definition of LSB but still capture the essence of DNL and INL in (3.3) and (3.4) when the input is uniformly distributed:

$$\text{maximum quantization cell size:} \qquad s_{\mathrm{D}} \triangleq \max_{0 \le i \le n} |v_{i+1} - v_i| , \tag{3.6}$$

$$\text{maximum quantization error:} \qquad s_{\mathrm{I}} \triangleq \max_{x} e(x; v^n, \mathcal{C}) . \tag{3.7}$$

Comparing these two metrics with DNL and INL defined in (3.3) and (3.4), we can see that s_{D} corresponds to DNL and s_{I} corresponds to INL. We note that s_{D} is independent of $\mathcal{C}$.

Unconstrained reproduction values

In the classical quantization problem, the set of reproduction values $\mathcal{C}$ are often unconstrained, i.e., we can assign values in $\mathcal{C}$ to any real number. In this case, we choose

$m = n + 1$, $c_i \in (v_{i-1}, v_i]$, and

$$Q_m\left(x; v^n, \mathcal{C}\right) = c_i \text{ if } x \in (v_{i-1}, v_i], \quad 1 \le i \le n+1, \tag{3.8}$$

where $v_1 < v_2 \ldots < v_n$, $v_0 \triangleq v_{\text{lo}}$, and $v_{n+1} \triangleq v_{\text{hi}}$.

In particular, scalar quantization theory indicates that the optimal $\mathcal{C}$ in the MSE sense satisfies the *centroid condition* [11], i.e.,

$$c_i = \mathbb{E}\left[X \middle| X \in (v_{i-1}, v_i]\right], 1 \le i \le m.$$

When the input is uniformly distributed, the centroid condition becomes the *midpoint condition*, i.e.,

$$c_i = (v_{i-1} + v_i)/2, 1 \le i \le m.$$

In this case, s_{D} relates to the maximum quantization error via $s_{\text{D}} = 2s_{\text{I}}$ and hence there is no need to discuss s_{I} performance given performance in terms of s_{D}.

Constrained reproduction values

In ADC design, it is often assumed the input is uniformly distributed and the set of reproduction values $\mathcal{C}$ are predetermined, where

$$c_i \triangleq (i - 1/2) \cdot v_{\text{LSB}} + v_{\text{lo}}.$$

In this case, it is more meaningful to use s_{I} as it takes constrained reproduction values into account.

Remark 3.5. *When m is large, say $m = 2^{16}$, the set $\mathcal{C}_m$ is so dense that we can essentially consider it as the unconstrained case.*

Based on the above discussion, we choose to use different performance metrics for the cases of unconstrained and constrained reproduction values.

- When $\mathcal{C}$ is unconstrained, we use MSE and maximum quantization cell size s_{D} to evaluate quantization performance. In particular, when discussing the MSE performance, we restrict our attention to the centroid reconstruction. In this case, a scalar quantizer is uniquely determined by its partition points v^n and we denote the corresponding MSE by $D\left(v^n\right)$.
- When $\mathcal{C}$ is constrained, we focus on the maximum quantization error s_{I} as the performance metric.

■ 3.3.2 Scalar quantization with noisy partition points

In this section we introduce the problem of scalar quantization when the partition points are subject to random variations. More specifically, an m-point scalar quantization is randomly generated by drawing each partition point $\tilde{v}_i$ in independently from a distribution $F_{\tilde{V}_i}$, and we denote the quantizer as $Q_m\left(\cdot; \tilde{V}^n, \mathcal{C}\right)$.

In this setting, all performance metrics become random variables. We let the random variable corresponding to maximum quantization cell size be $S_{\mathrm{D}}\left(\tilde{V}^n\right)$ and maximum quantization error be $S_{\mathrm{I}}\left(\tilde{V}^n, \mathcal{C}\right)$. Then noting the size of the i-th quantization cell is $\tilde{V}_{(i+1)} - \tilde{V}_{(i)}$ (cf. Section 1.1 for the definition of $\tilde{V}_{(i)}$),

$$S_{\mathrm{D}}\left(\tilde{V}^n\right) = \max_{0 \le i \le n} \left(\tilde{V}_{(i+1)} - \tilde{V}_{(i)}\right), \tag{3.9}$$

$$S_{\mathrm{I}}\left(\tilde{V}^n, \mathcal{C}\right) = \max_{x} e\left(x; \tilde{V}^n, \mathcal{C}\right). \tag{3.10}$$

In addition, we take expectation over the random partition points $\tilde{V}^n$ when calculating the MSE, which leads to

$$\mathrm{MSE} \triangleq \mathbb{E}_{\tilde{V}^n}\left[D\left(\tilde{V}^n\right)\right]. \tag{3.11}$$

Remark 3.6. *In (3.11) the expectation w.r.t. $\tilde{V}^n$ indicates we are averaging over different realizations of the partition points. Therefore, MSE here is the average of the mean-square error of an ensemble of quantizers, and an achievable MSE does not guarantee that all quantizers in this ensemble can achieve this MSE.*

For the problem of scalar quantization with noisy partition points, we investigate how the set of distributions $\left\{F_{\tilde{V}_i}\right\}$ impacts the system performance metrics in (3.9) to (3.11). To investigate this, we derive results by extending the high resolution analysis in classical scalar quantization theory to the setting with noisy partition points in Section 3.4.

More specifically, for the ADC with imprecise comparators model in Section 3.2, if we design the reference voltages to be v^n, then $\tilde{V}_i \overset{\text{indep.}}{\sim} \mathsf{N}\left(v_i, \sigma^2\right), 1 \le i \le n$. Given σ^2, we utilize the theory in Section 3.4 to investigate how the choice of v^n impacts performance metrics, and present the detailed investigation in Section 3.5.

Remark 3.7. *The problem of quantization with uniformly distributed partition points for the uniform input distribution has been investigated in [20], under a different motivation, and it turns out to be a useful building block in our analysis.*

■ 3.4 High resolution analysis for noisy partition points

In this section we extend the high resolution analysis technique to the case of scalar quantization with noisy partition points. High resolution analysis analyzes the performance of a quantizer as the number of partition points approaches infinity, and approximate this sequence of partition points by a point density function. For the classical quantization problem in Section 3.3.1, high resolution analysis of MSE [21–23] leads to mathematical tractable performance results and yields useful approximate results for MSE-optimal quantizer design. We apply the high resolution analysis techniques to the problem in Section 3.3.2. We analyze not only MSE but also maximum quantization cell size S_{D} and maximum quantization error S_{I}, in Section 3.4.1, Section 3.4.2 and Section 3.4.3 respectively. While the analysis for maximum quantization cell size and maximum quantization error is straightforward in the classical case, it is non-trivial in the case of quantization

with noisy partition points, because the variation in partition point location induces a random ordering of the partition points, as the order statistics in (3.9) indicates. Combining tools from *order statistics* and high resolution analysis, we derive analytical expressions for MSE and the distribution of S_{D} and S_{I}. Then in Section 3.5, we apply these results to gain more insights on ADC performance and obtain better Flash ADC designs.

We defer most proofs and derivations in this section to Appendix A.2.

We first introduce one of the key ideas in high resolution analysis, *point density function*, via which we can approximate a sequence of values as a density function.

Definition 3.1 (Point density function). *A sequence of values v^n is said to have* point density function $\lambda(x)$ *if*

$$\lambda(x) = \lim_{\delta\to 0}\lim_{n\to\infty}\frac{1}{n\delta}N\left(x, x+\delta; v^n\right), \quad x \in \mathbb{R}. \tag{3.12}$$

Given a point density function λ and n, we could "sample" n partition points v'^n by letting

$$v'_i = F_\lambda^{-1}\left(i/(n+1)\right), 1 \le i \le n, \tag{3.13}$$

where $F_\lambda(\cdot)$ is the c.d.f. corresponds to λ. This establishes an one-one correspondence between a point density function and a sequence of partition points.

Example 3.1. *For $\lambda \sim \mathit{Unif}([-1,1])$, then the v'^n corresponds to $\lambda(\cdot)$ is an n-point evenly spaced grid in $[-1,1]$.*

We can generalize the concept of point density function to the case of noisy partition points, where we represent each partition point by a random variable. Let W^n be the n random partition points, where $W_i \overset{\text{indep.}}{\sim} f_{W_i}(\cdot)$, then we say $f_{\bar{W}}(\cdot)$ is a point density function for W^n if

$$f_{\bar{W}}(x) = \lim_{\delta\to 0}\lim_{n\to\infty}\frac{1}{n\delta}\mathbb{E}_{W^n}\left[N(x, x+\delta; W^n)\right], \quad x \in \mathbb{R}. \tag{3.14}$$

In particular, when $W_i \overset{i.i.d.}{\sim} f_W$, $f_{\bar{W}}$ in (3.14) becomes f_W.

While we do not have full control on the location of the partition points as they are random variables, in some applications we may be able to influence their locations by controlling certain parameter of the random variable. In the context of Flash ADC design with reference voltage offsets, we have each W_i being a Gaussian random variable with certain mean v_i and variance σ^2. The variance σ^2 is determined by the fabrication process, but we can design the means v^n to impact the point density function $f_{\bar{W}}(\cdot)$. Therefore, by understanding how $f_{\bar{W}}(\cdot)$ impacts the quantization performance, we design v^n accordingly, as we shall see in Section 3.5.

In the following few sections, we analyze the performance of quantization with noisy partition points via high resolution analysis. In particular, we derive the functional relationship between $f_{\bar{W}}(\cdot)$ and performance metrics (MSE, maximum quantization cell size and maximum quantization error).

■ 3.4.1 High resolution analysis of MSE

In this section we develop an analogous result to the high-resolution approximation of MSE for non-uniform quantization, as Bennett [21] first did for the classical quantization problem.

High resolution approximation of MSE: given input X with p.d.f. f_X and n random partition points W^n, each with p.d.f. f_{W_i}, if the set of densities $\{f_{W_i}, 1 \leq i \leq n\}$ are smooth over $\text{Supp}(f_X)$ and corresponds to the point density function $f_{\bar{W}}(\cdot)$ defined in (3.14), then in the high resolution regime,

$$\mathbb{E}_{X,W^n}\left[d\left(X, W^n\right)\right] \simeq \frac{1}{2n^2} \int f_X(x) f_{\bar{W}}^{-2}(x)\, dx, \tag{3.15}$$

provided that the integral in (3.15) is finite (in particular, $\text{Supp}(f_{\bar{W}}) \supset \text{Supp}(f_X)$).

The related derivations are presented in Appendix A.2.1.

Remark 3.8. *In the classical scalar quantization problem, the partition points are deterministic. Therefore, given $f_{\bar{W}}(\cdot)$, if a sequence of (non-random) partition points v^n correspond to point density function $f_{\bar{W}}(\cdot)$, i.e.,*

$$f_{\bar{W}}(x)\, \delta \simeq \frac{1}{n} N(x, x+\delta; v^n),$$

where v^n can be obtained via (3.13). Bennett [21] shows for a quantizer with partition points v^n, the high-resolution approximation of MSE satisfies

$$\text{MSE} \simeq \frac{1}{12n^2} \int f_X(x) f_{\bar{W}}^{-2}(x)\, dx, \tag{3.16}$$

which is exactly $1/6$ of (3.15). The increase of MSE in (3.15) with respect to (3.16) comes from the random sizes of quantization cells, because the increase in square error due to larger cells outweighs the decrease in square error due to smaller cells.

Remark 3.9. *Since high resolution approximation requires sufficiently many points in each small interval within the support of f_X, the smaller the minimum of $f_{\bar{W}}(\cdot)$ over f_X, the larger n we need to achieve the high resolution approximation. The Monte-Carlo simulations in Fig. 3-5 demonstrate this effect for different f_X with $f_{W_i} \overset{i.i.d.}{\sim} f_W \sim \mathcal{N}\left(0, \sigma^2\right)$, where a smaller σ leads to smaller minimum of $f_{\bar{W}}(\cdot)$ over f_X, and hence a larger n is needed for the high resolution approximation to hold.*

■ 3.4.2 High resolution analysis of maximum quantization cell size

With noisy partition points, the maximum quantization cell size S_{D} is a random quantity and in this section we derive its c.d.f. in the high resolution regime. We restrict our attention to an input distribution with finite support $[a, b]$ as otherwise S_{D} is infinite. As discussed in Section 3.3.1, this performance measure is most meaningful when the set of output codes $\mathcal{C}$ is unconstrained.

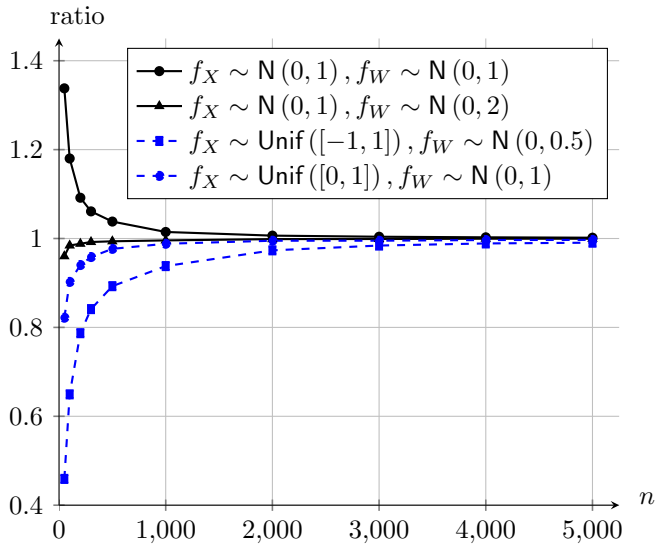


Figure 3-5: The ratio of $\mathbb{E}_{X,W^n}\left[d\left(X, W^n\right)\right]$ obtained from Monte-Carlo simulations and numerical calculations of the integral in (3.15) for a variety of $f_X(\cdot)$ and $f_W(\cdot)$.

High resolution analysis of the maximum quantization cell size: given input X with p.d.f. f_X, where $\mathrm{Supp}(f_X) = [a,b]$, and n independent random partition points W^n, each with p.d.f. f_{W_i}, if the set of densities $\{f_{W_i}, 1 \le i \le n\}$ are smooth over $[a,b]$ and corresponds to the point density function $f_{\bar{W}}(\cdot)$ defined in (3.14), then in the high resolution regime,

$$\mathbb{P}\left[S_{\mathrm{D}}\left(W^n\right) \le s\right] \simeq \exp\left\{-n\int_a^b f_{\bar{W}}(w)\, e^{-nsf_{\bar{W}}(w)}\, dw\right\}, \tag{3.17}$$

provided $f_{\bar{W}}(x) > 0$ for any $x \in [a,b]$.

Remark 3.10. *The requirement $f_W(x) > 0$ for any $x \in [a,b]$ is crucial in the asymptotic approximation (3.17), as the approximation use the fact that for any interval with size s within $[a,b]$, there are sufficiently many number of partition points in the interval.*

In particular, the right hand side in (3.17) is a poor approximation for $\mathbb{P}\left[S_{\mathrm{D}}\left(W^n\right) \le s\right]$ when the assumption $f_W > 0$ is violated. For example, when $f_W(x) = 0$ for any $x \in [a,b]$,

$$\exp\left\{-n\int_a^b f_{\bar{W}}(w)\, e^{-nsf_{\bar{W}}(w)}\, dw\right\} = 1.$$

As a result, we cannot obtain a design that achieves small $S_{\mathrm{D}}\left(W^n\right)$ by finding the $f_{\bar{W}}(\cdot)$ that minimizes the right hand side of (3.17).

With (3.17), we can derive the range of S_{D} in the high resolution regime.

Range of maximum quantization cell size: in the high resolution regime,

$$\mathbb{P}\left[\frac{1}{f_{\max}^{[a,b]}}\frac{\log n}{n} \le S_{\mathrm{D}}\left(W^n\right) \le \frac{1}{f_{\min}^{[a,b]}}\frac{\log n}{n}\right] \to 1 \tag{3.18}$$

as $n \to \infty$, where

$$f_{\max}^{[a,b]} \triangleq \max_{x\in[a,b]} f_{\bar{W}}\left(x\right), \tag{3.19}$$

$$f_{\min}^{[a,b]} \triangleq \min_{x\in[a,b]} f_{\bar{W}}\left(x\right). \tag{3.20}$$

The related derivations are presented in Appendix A.2.2.

Remark 3.11. *When n partition points form an evenly spaced grid in $[a,b]$, the maximum quantization cell size is $(b-a)/(n+1)$. Therefore, (3.18) indicates in the high resolution regime, the randomness in partition locations leads to an order of $\log n$ increase in the maximum quantization cell size.*

■ 3.4.3 High resolution analysis of maximum quantization error

In this section we derive the c.d.f. of S_{I} in the high resolution regime. Again, we restrict our attention to an input distribution with finite support $[a,b]$ as otherwise S_{I} is infinite. As discussed in Section 3.3.1, when the set of output codes $\mathcal{C}$ is unconstrained, the maximum quantization error is exactly half of the maximum quantization cell size. Hence in this section we focus on the case that $\mathcal{C}$ is constrained, where this performance measure is most meaningful. In particular, we choose $\mathcal{C} = \mathcal{C}_m$ that corresponds to the reproduction values of a uniform ADC, i.e., the midpoints of an evenly spaced grid in $[a,b]$,

$$c_i = a + (b-a)\cdot(i-0.5)/m, \quad 1 \le i \le m. \tag{3.21}$$

These results regarding $\mathcal{C}_m$ are applied in Section 3.5, and it is not hard to see that our analysis can be extended to other choices of $\mathcal{C}$ as well.

High resolution analysis of maximum quantization error: given input X with p.d.f. f_X, where $\mathrm{Supp}(f_X) = [a,b]$, and n independent random partition points W^n, each with p.d.f. f_{W_i}, if the densities $f_{W_i}, 1 \le i \le n$ are smooth over $[a,b]$ and correspond to the point density function $f_{\bar{W}}(\cdot)$ defined in (3.14), then in the high resolution regime,

$$\mathbb{P}\left[S_{\mathrm{I}}\left(W^n, \mathcal{C}_m\right) \le \frac{b-a}{2m} + s\right] \simeq 1 - \sum_{i=1}^{m} e^{-np_i(s)}, \tag{3.22}$$

where $p_i(s) \triangleq F_{\bar{W}}\left(c_i+s\right) - F_{\bar{W}}\left(c_i-s\right) \approx 2f_{\bar{W}}\left(c_i\right)s$ and provided that $f_{\bar{W}}\left(x\right) > 0$ for any $x \in [a,b]$.

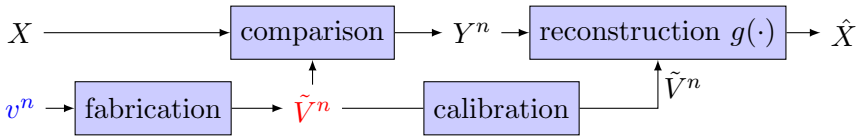


Figure 3-6: Block diagram of a Flash ADC with imprecise comparators. X is the input signal, v^n are the designed reference voltages and the $\tilde{V}^n$ are the fabricated reference voltages, which is a noisy version of v^n. A comparison of X and $\tilde{V}^n$ leads to the comparator outputs Y^n. The reconstructor $g(\cdot,\cdot)$ takes both Y^n and $\tilde{V}^n$ to produce $\hat{X} \in \mathcal{C}$.

Based on (3.22) we derive a result on the range of the maximum quantization error.

The range of maximum quantization error: in the high resolution regime, for $\mathcal{C}_m$ with values specified in (3.21), and any $t > 0$,

$$\mathbb{P}\left[\frac{1}{2f_{\max}^{[a,b]}}\frac{\log m}{n} \le S_{\mathrm{I}}\left(W^n, \mathcal{C}_m\right) - \frac{b-a}{2m} \le \frac{1}{2f_{\min}^{[a,b]}}\frac{\log m + t}{n}\right] \ge 1 - e^{-t} \tag{3.23}$$

as $n \to \infty$, where $f_{\max}^{[a,b]}$ and $f_{\min}^{[a,b]}$ are defined in (3.19) and (3.20).

The related derivations are presented in Appendix A.2.3.

Remark 3.12. *When $m-1$ partition points form an evenly spaced grid in $[a,b]$, the maximum quantization error is $(b-a)/(2m)$. Therefore, (3.23) indicates in the high resolution regime, the randomness in partition locations leads to an increase on the order of $\log m/n$ in maximum quantization error.*

■ 3.5 Applications to Flash ADC design

In this section we apply results developed in Section 3.4 to the problem of Flash ADC design with imprecise comparators described in Section 3.2.2. This problem can be described by the framework of scalar quantization with noisy partition points, as the diagram in Fig. 3-6 indicates. More specifically, for a given ADC, we design the comparators to have reference voltages v^n, and the resulting fabricated reference voltages $\tilde{V}^n$ are independent random variables that satisfy $\tilde{V}_i \overset{i.i.d.}{\sim} \mathsf{N}\left(v_i, \sigma^2\right)$. The output Y_i of comparator i satisfies

$$Y_i = \mathbb{1}\left\{X > \tilde{V}_i\right\},$$

where X is the input signal. The fabricated reference voltages are provided to the decoder via the calibration process, and the reconstructor $g(\cdot,\cdot)$ takes both Y^n and $\tilde{V}^n$ to produce $\hat{X} \in \mathcal{C}$, an estimate of X.

For this formulation, we ask the following question:

> How should we design v^n such that a Flash ADC with imprecise comparators still achieves good performance in terms of MSE, maximum quantization cell size, and maximum quantization error?

To answer this question, we use the idea of high resolution analysis again, and represent the v^n by its point density function $\tau(x)$, and let $\phi(\cdot)$ be the probability density function for Gaussian distribution $\mathsf{N}\left(0, \sigma^2\right)$, where $\sigma > 0$. We first analyze the expected number of fabricated reference voltages in each small interval $[x, x + dx]$ and obtain the following result:

Lemma 3.1.

$$\frac{1}{n}\mathbb{E}\left[N\left(x, x + dx; \tilde{V}^n\right)\right] \simeq (\tau * \phi)(x)dx.$$

We defer all proofs and derivations in this section to Appendix A.3.

Then by (3.14), the resulting point density function of the random partition points $\lambda(\cdot)$ (which corresponds to $f_{\bar{W}}(\cdot)$ in (3.14)) is the convolution of two densities τ and ϕ

$$\lambda(x) = (\tau * \phi)(x), \tag{3.24}$$

where $*$ indicates convolution, i.e.,

$$(f * g) \triangleq \int f(t) \cdot g(x - t)\, dt.$$

This leads to the relationship summarized in Fig. 3-7, where we can design v^n, and the system performance is determined by $\tilde{V}^n$. Noting that results in Section 3.4 imply how the ADC performance metrics relate to λ, and (3.24) shows how λ relates to τ, we derive how τ impacts ADC performance in terms of MSE in Section 3.5.1, which provides us with insights on designing v^n in Section 3.5.2. In addition, we analyze the technology scaling in Section 3.5.3.

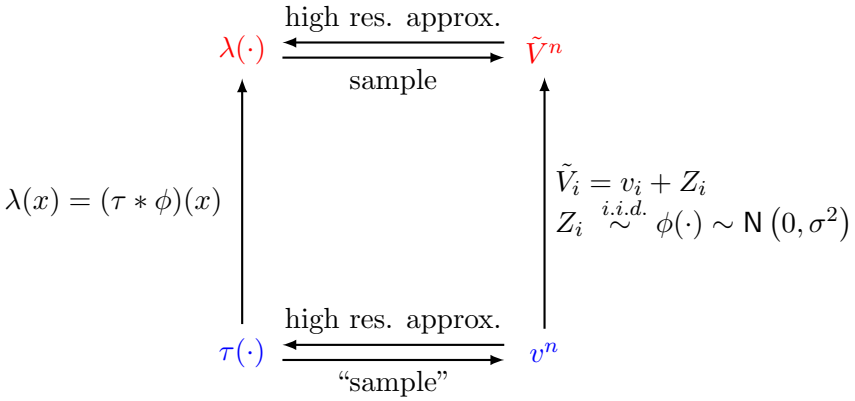


Figure 3-7: Relationship between the design and fabricated reference voltages and their point density functions.

■ 3.5.1 MSE-optimal partition point density analysis

In this section we investigate the point density function τ of designed reference voltages that minimizes MSE for a general input distribution, and then specialize our results to both Gaussian and uniform input distributions to obtain the corresponding MSE-optimal reference voltage designs.

Following (3.15) and (3.24), the MSE of a Flash ADC with imprecise comparators satisfies

$$\mathbb{E}_{X,\tilde{V}^n}\left[d\left(X,\tilde{V}^n\right)\right] \simeq \frac{1}{2n^2}\int f_X(x)\lambda^{-2}(x)\,dx. \tag{3.25}$$

(3.25) indicates the integral

$$R(\tau) = \int f_X(x)(\tau * \phi)^{-2}(x)\,dx \tag{3.26}$$

is the key quantity in MSE calculation, and in this section we characterize the τ^* that minimizes $R(\tau)$ in a variety of scenarios of interest.

Remark 3.13. *It is not hard to see that for a fixed n, taking $\sigma \to 0$ leads to high resolution approximation in (3.16) rather than (3.15), as $\sigma = 0$ corresponds to the classical ADC design with no reference voltage offsets. When $\sigma > 0$, (3.15) holds as $n \to \infty$. However, if an ADC design corresponds to an n that is not too large, then using (3.16) to calculate MSE may be more appropriate, because the convergence of MSE to (3.15) depends on n, as indicated in Fig. 3-5. However, as both MSE expressions share the integral (3.26), the search of τ that minimizes (3.26) is beneficial regardless of the regime we operate in.*

Theorem 3.2. *τ^* minimizes $R(\tau)$ if and only if*

$$\sup_{x\in\mathcal{A}}\left[\frac{f_X}{(\tau^**\phi)^3} * \phi\right](x) \le \left\langle f_X, \frac{1}{(\tau^**\phi)^2}\right\rangle. \tag{3.27}$$

In particular, if there exists a τ^* such that

$$\tau^* * \phi \propto f_X^{1/3}, \tag{3.28}$$

then τ^* minimizes $R(\tau)$ and

$$R(\tau^*) = \left(\int f_X^{1/3}(x)dx\right)^3. \tag{3.29}$$

Note (3.28) and (3.29) corresponds to the classical Panter-Dite formula [22].

Based on Theorem 3.2, we can derive the optimal τ when the input distribution is Gaussian or uniformly distributed.

Theorem 3.3 (Gaussian input distribution). *When $X \sim \mathsf{N}\left(0,\sigma_X^2\right)$,*

$$\tau^* \sim \begin{cases} \mathsf{N}\left(0, 3\sigma_X^2 - \sigma^2\right) & \text{when } 3\sigma_X^2 > \sigma^2 \\ \delta(x) & \text{when } 3\sigma_X^2 \le \sigma^2 \end{cases},$$

and

$$R(\tau^*) = \begin{cases} 6\sqrt{3}\pi\sigma_X^2 & \text{when } 3\sigma_X^2 > \sigma^2 \\ 2\pi\sigma^3/\sqrt{\sigma^2 - 2\sigma_X^2} & \text{when } 3\sigma_X^2 \le \sigma^2 \end{cases}.$$

Theorem 3.4 (Uniform input distribution). *When $X \sim \mathsf{Unif}([-1,1])$ and $\sigma \ge \sigma_0$, where σ_0 is a constant and $\sigma_0 \approx 0.7228$, $\tau^*(x) = \delta(x)$ and*

$$R(\tau^*) = 2\pi\sigma^2 \int_0^1 \exp\left(-\frac{x^2}{2\sigma^2}\right) dx.$$

Remark 3.14. *While in current circuit design the value of σ is often much less than the ADC input range (with the exception of [9]), our results may turn out to be important with aggressive technology scaling that eventually lead to σ values that are on the same order of the ADC input range.*

Remark 3.15. *For both Gaussian and uniform input distributions, when σ large enough, $\tau^*(x) = \delta(x)$. In this case, simply aiming to place all partition points at $x = 0$ and letting the noisy placement process spread them out naturally is optimal in terms of MSE.*

When the input is uniform and $\sigma < \sigma_0$, we do not have a closed form expression for τ^* and instead we search for it via numerical optimization. Noting that the classical Lloyd-Max iterative algorithm no longer applies as the placement of partition points is subject to random variations, we propose an alternative algorithm guarantees convergence to local optimum. In particular, we approximate τ^* by a discrete distribution $\hat{\tau}$, where

$$\hat{\tau}(x; \mathbf{p}, \mathbf{a}) = \sum_{i=0}^{k} p_i(\delta(x - a_i) + \delta(x + a_i)),$$

where $a_i \ge 0$ and the symmetry of $\hat{\tau}^*$ follows from the symmetry of f_X. Without loss of generality, we assume $a_0 = 0$. Then we find the best $\mathbf{p}$ and $\mathbf{a}$ that corresponds to $\hat{\tau}^*$ that minimizes $R(\tau)$ via Algorithm 1.

Remark 3.16. *Since the optimization problem is non-convex, Algorithm 1 only guarantees that it converges to a local optimum rather than a global optimum. In practice we run the algorithm with multiple randomly perturbed initial solutions to increase the probability of reaching global optimum.*

Some examples of $\hat{\tau}^*$ for different values of σ are shown in Fig. 3-8 on page 43.

In our numerical optimization procedure, we observe the phenomenon that the probability mass tends to concentrate to a few locations even when the initial solution has non-zero probability mass at more locations, leading to the following conjecture.

Conjecture 3.5. *For any $\sigma > 0$, the optimal density τ^* is singular, i.e., has the form*

$$\hat{\tau}(x; \mathbf{p}, \mathbf{a}) = \sum_{i=0}^{k} p_i(\delta(x - a_i) + \delta(x + a_i))$$

for some $\mathbf{p}$ and $\mathbf{a}$.

Algorithm 1 Iterative optimization for $\hat{\tau}$.

$p_i^{(1)} = 1/(2k+1)$ for $0 \le i \le k$
$a_i^{(1)} = i/(k-1)$ for $1 \le i \le k$
$E_0 = 0$
$E_1 = R\left(\hat{\tau}\left(\cdot; \mathbf{p}^{(1)}, \mathbf{a}^{(1)}\right)\right)$
$t = 1$
while $|E_t - E_{t-1}| \ge \varepsilon$ **do**
  $\mathbf{p}^{(t+1)} = \arg\min_{\mathbf{p}} \hat{\tau}\left(x; \mathbf{p}, \mathbf{a}^{(t)}\right)$
  $\mathbf{a}^{(t+1)} = \arg\min_{\mathbf{a}} \hat{\tau}\left(x; \mathbf{p}^{(t+1)}, \mathbf{a}\right)$
  $E_{t+1} = R\left(\hat{\tau}\left(\cdot; \mathbf{p}^{(t+1)}, \mathbf{a}^{(t+1)}\right)\right)$
  $t = t + 1$
end while

We note that a similar phenomenon has been observed in [24] and the proof technique therein could be useful.

■ 3.5.2 Flash ADC design with imprecise comparators

In this section we investigate the problem of designing a b-bit Flash ADC with imprecise comparators. Without loss of generality, we assume the input range of the ADC is $[-1, 1]$, and hence the LSB of a b-bit ADC is $2/2^b$.

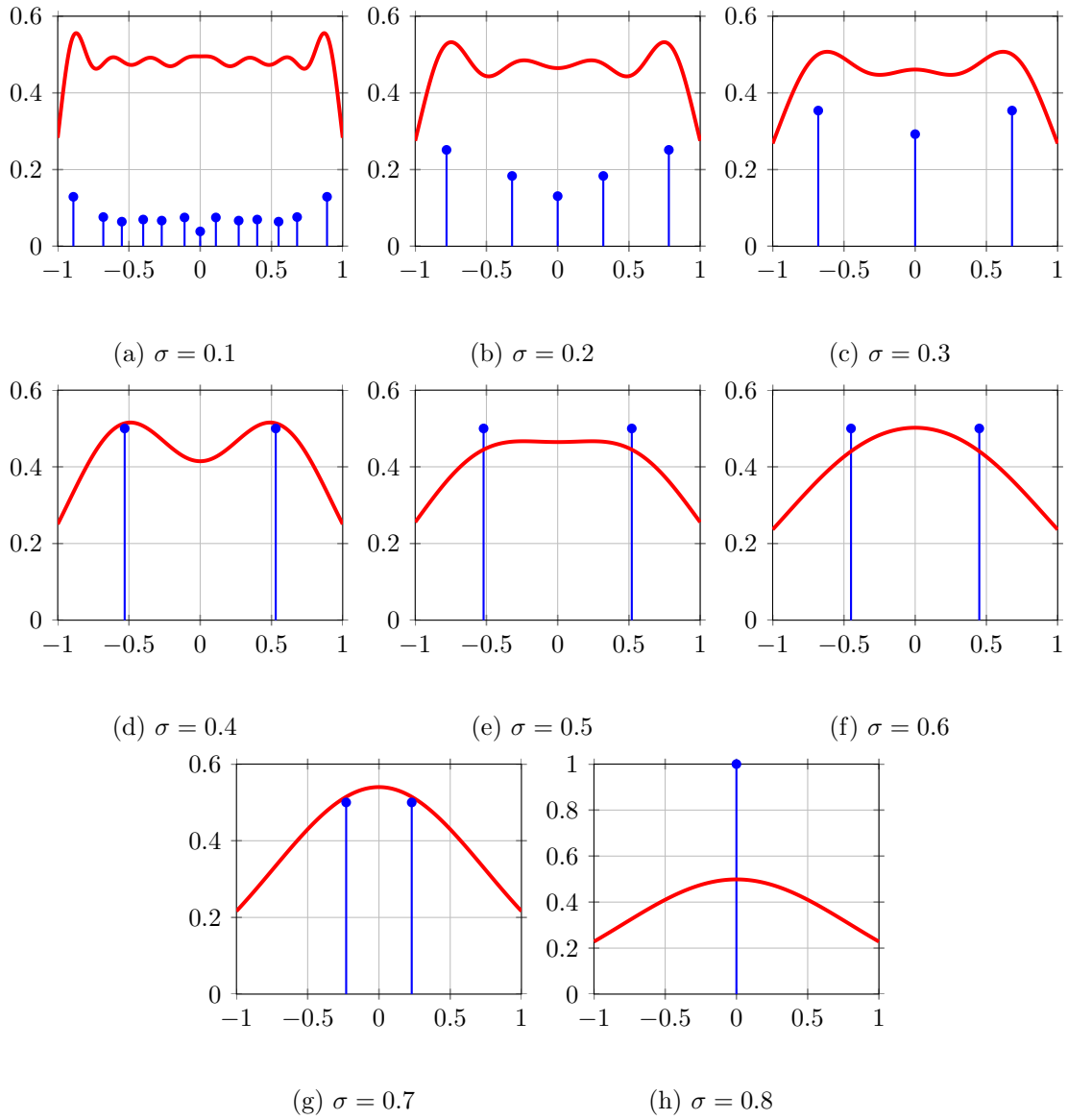
In this section we consider the Flash ADC to conform to the standard output interface, i.e., a b-bit ADC has $m = 2^b$ output codes $\mathcal{C}_m = \{c_i, 1 \le i \le m\}$, where

$$c_i = -1 + (i - 0.5) \cdot 2/m.$$

As discussed in Section 3.3, we use maximum quantization error as the main performance metric.

As mentioned in Remark 3.2, in practice the requirement for maximum quantization error is often in the range of 1LSB to 1.25LSB. As an example, in the following sections we analyze designs that achieve maximum quantization errors of less than 1LSB. Due to the fabrication variation, it is unlikely that $n = m - 1$ comparators, which corresponds to n reference voltages, can achieve the required maximum quantization error. Therefore, more comparators are needed and we assume $n = r(m-1)$, where we say r is the *redundancy factor*.

Below we first show that while it is possible to find S_{I}-optimal designs by minimizing (3.22), empirically these designs achieve very similar performance with the MSE-optimal designs obtained in Section 3.5.1, in terms of the c.d.f. of $S_{\mathrm{I}}\left(\tilde{V}^n\right)$. Then we show that using MSE-optimal designs indeed leads to better S_{I} performance comparing to using traditional designs, where v^n form an evenly spaced grid. Furthermore, we show that given the same number of comparators n, using more than 2^b reproduction points (e.g., $m = 2^{b+1}$) improves S_{I} performance significantly as we can make better use of all the fabricated comparators. Finally, we compare our design to *stochastic ADC* [9], another



(a) $\sigma = 0.1$ (b) $\sigma = 0.2$ (c) $\sigma = 0.3$

(d) $\sigma = 0.4$ (e) $\sigma = 0.5$ (f) $\sigma = 0.6$

(g) $\sigma = 0.7$ (h) $\sigma = 0.8$

Figure 3-8: $\hat{\tau}^*(x)$ and $\hat{\tau}^* * \phi(x)$ obtained from Algorithm 1 for uniform input distribution over $[-1, 1]$, with $k = 7$ for all σ values. The stems indicate $\hat{\tau}(x)$ and the solid curves indicate $(\hat{\tau} * \phi)(x)$.

design that aims to take process variations into account, and demonstrate this intuitive design, unfortunately, does not achieve good performance.

MSE-optimal and S_{I}-optimal design achieves similar maximum quantization error

We compare the S_{I} performance for 6-bit ADC designs obtained from optimizing (3.22) and (3.15) respectively, at redundancy factor $r = 6$ and various values of σ. Fig. 3-9 show

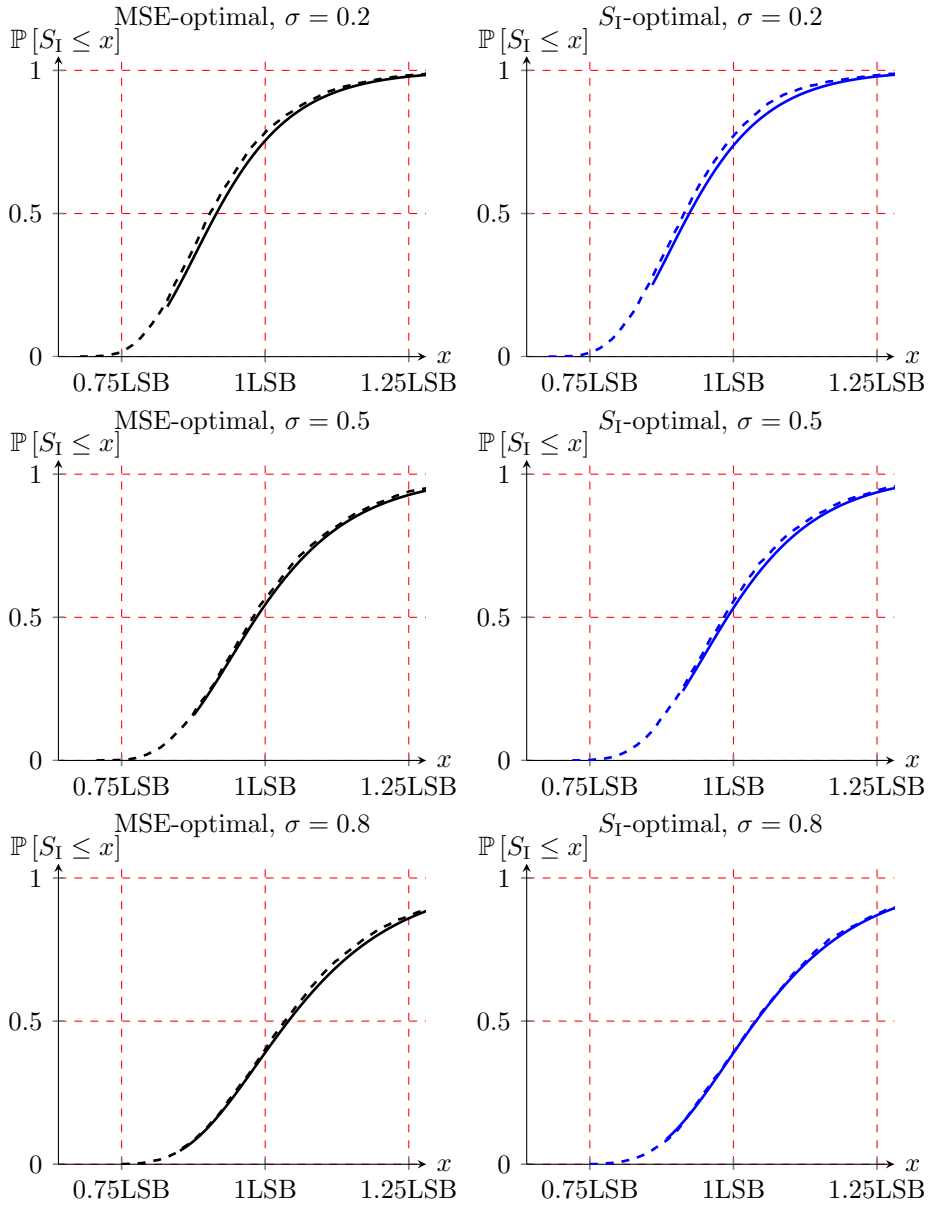


Figure 3-9: Comparisons of c.d.f.s correspond to MSE-optimal and S_{I}-optimal designs for $b = 6, r = 6$ and $\sigma = 0.2, 0.5, 0.8$. The solid lines are computed based on (3.22) and the dashed lines are obtained from Monte-Carlo simulations. The solid lines are only plotted between asymptotic lower and upper bounds obtained from (3.23) with $t = 4$.

that for all cases evaluated, the c.d.f.s of the two designs are essentially the same, and hence we choose the MSE-optimal designs due to its simplicity as the locations of v^n are more concentrated.

MSE-optimal design is better than uniform design

We compare the S_{I} performance of the MSE-optimal design and the uniform design. In the uniform design, the $n = r(2^b - 1)$ designed reference voltages v^n form a grid as follows:

$$v_{ir+j} = u_i, 1 \leq j \leq r, 0 \leq i \leq 2^b - 2. \tag{3.30}$$

This design corresponds to the traditional uniform ADC design when $r = 1$, and most existing research works on Flash ADC with redundancy apply this design with $r = 2$ or $r = 4$ [7, 8]. Fig. 3-10 shows that for different values of σ and appropriate redundancy factors r, for a range of x within $[0.75\mathrm{LSB}, 1.25\mathrm{LSB}]$, MSE-optimal designs increases $\mathbb{P}[S_{\mathrm{I}} \leq x]$ by 5%—10%! And this is achieved by simply modifying the design of reference voltages. Alternatively, defining the yield as the maximum quantization error being less than 1LSB, Fig. 3-11 shows how yield changes as σ increases, given a certain redundancy factor r.

Using more output codes reduces maximum quantization error

We compare the S_{I} performance for ADCs with the same number of comparators but different number of output codes m. In particular, we consider the case of $n \approx 2^9$ comparators and $m = 2^6, 2^7$ and 2^8 respectively, which corresponds to a 6-bit ADC with 8 times redundancy, a 7-bit ADC with 4 times redundancy, and an 8-bit ADC with 2 times redundancy. Noting that 1LSB in 6-bit ADC is 2LSB in 7-bit ADC and 4LSB in 8-bit ADC, and letting the 1LSB for 6-bit ADC be Δ, Fig. 3-13 indicate that 6-bit ADC with $r = 8$ ($n = r(2^b - 1) = 504$ comparators) achieves $\mathbb{P}[S_{\mathrm{I}} \leq \Delta] \approx 0.88$, a 7-bit ADC with $r = 4$ ($n = 508$) achieves $\mathbb{P}[S_{\mathrm{I}} \leq \Delta] \approx 0.97$, a 8-bit ADC with $r = 2$ ($n = 510$) achieves $\mathbb{P}[S_{\mathrm{I}} \leq \Delta] \approx 1.0$. A similar phenomenon can be observed for $\sigma = 0.8$ in Fig. 3-13 and the increase is even more significant. Therefore, an 1-bit increase in output code could lead to significant improvement in S_{I} performance, as with more reproduction values, we can convey more information about the input, which is available from the fabricated comparators. This phenomenon is demonstrated in Fig. 3-12, where we show that a 2-bit ADC with reproduction values $\mathcal{C}_8$ can achieve smaller quantization error than the same ADC with reproduction values $\mathcal{C}_4$.

Remark 3.17. *In practice, designers often use a higher resolution ADC (say, 7-bit ADC) when a quantization accuracy of a 6-bit ADC is required, which naturally leads to performance improvement in INL. However, as the number of comparators in traditional ADCs satisfy $n = 2^b - 1$, a 7-bit ADC has about two times the number of comparators, while in our discussion above, all designs have the same number of comparators and hence using more output bits does not lead to increase in the number of comparators.*

Comparison with stochastic ADC

In circuit system research, [9] presents a design that explores the idea of high resolution quantization. Assuming uniform input over $[-\sigma, \sigma]$, their design corresponds to n in the range of 1000 to 2000, and $\tau(x) = \delta(x - 1.078\sigma)/2 + \delta(x + 1.078\sigma)/2$, with the rationale of making the resulting density $\lambda = \tau * \phi$ as uniform as possible in the input range $[-\sigma, \sigma]$. However, as we showed in Theorem 3.4, the MSE-optimal solution is $\tau^*(x) = \delta(x)$. As Fig. 3-14 shows, assuming $\sigma = 1$, while $\lambda_{\mathrm{stochastic}}$ is approximately flat in the input range

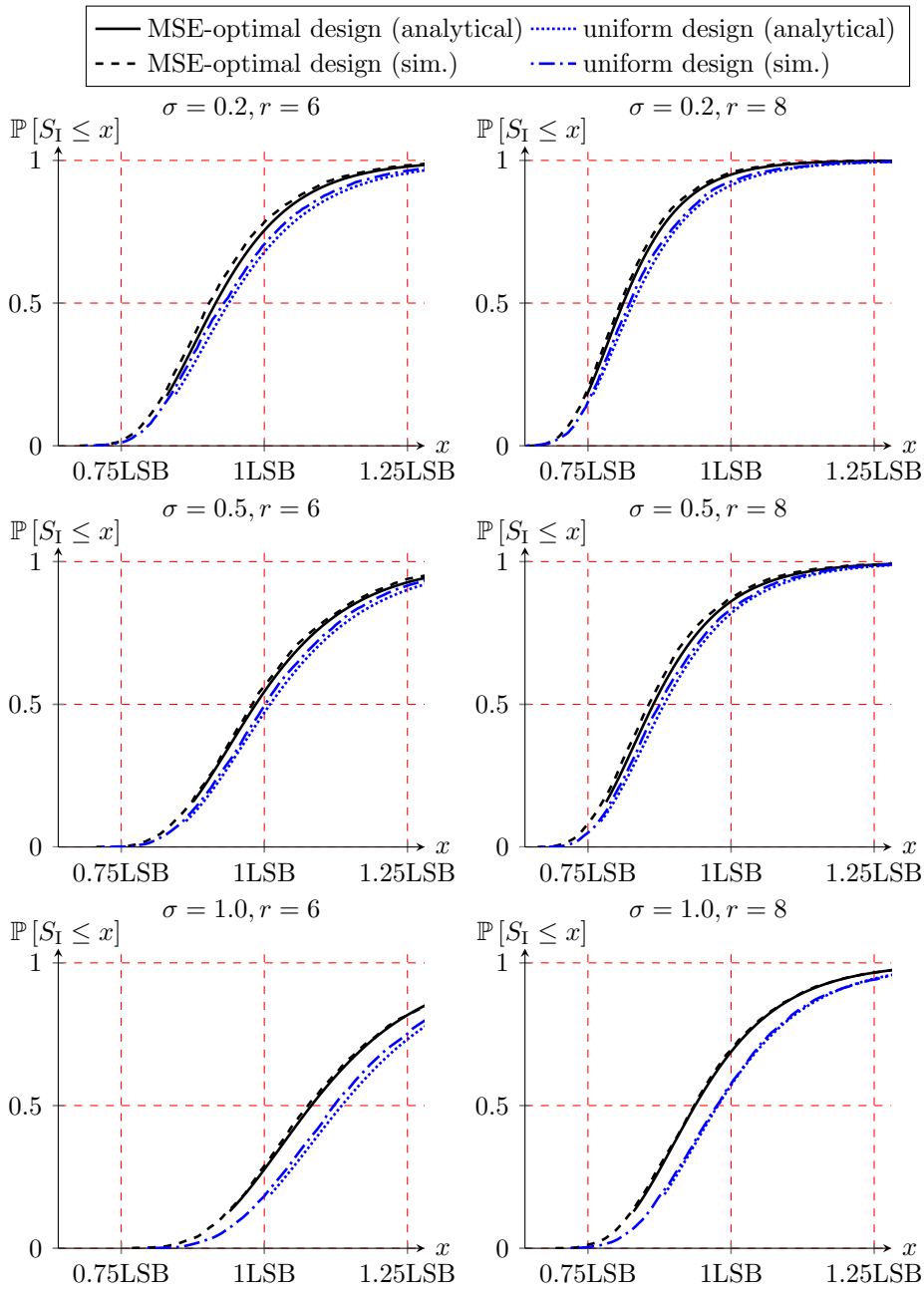


Figure 3-10: Comparisons of c.d.f.s correspond to MSE-optimal and uniform designs for 6-bit Flash ADCs. The "analytical" lines are computed based on (3.22) and the "simulated" lines are obtained from Monte-Carlo simulation.

$[-1, 1]$, many partition points are wasted as they are out of the input range. Calculation shows $\mathrm{MSE}_{\mathrm{stochastic}}/\mathrm{MSE}^* \approx 2.15$, which corresponds to slightly more than 1 effective

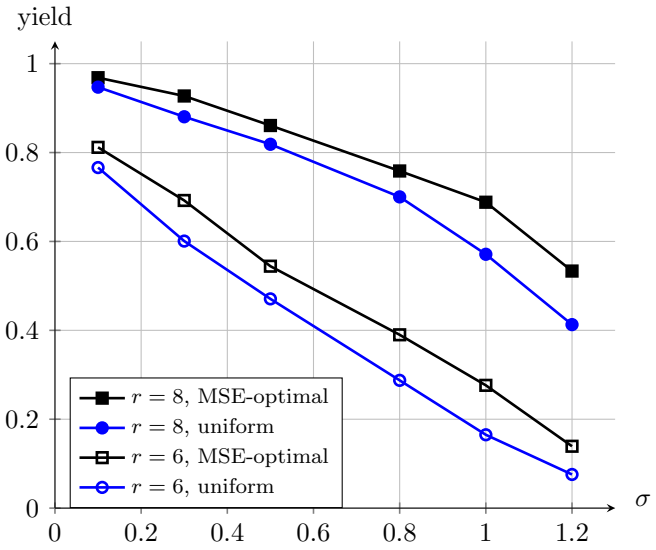


Figure 3-11: The probability of maximum quantization error less than 1LSB (yield) for a 6-bit ADC with different designs and σ values. The MSE-optimal designs are the ones shown in Fig. 3-8, and the "uniform" designs are specified in (3.30).

number of bit (ENOB) difference. This is significant for the design in [9] with ENOB in the range of 5 to 6 bits.

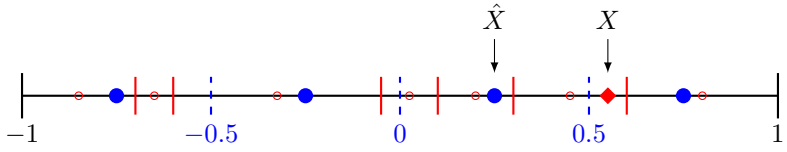
Remark 3.18. *The above analysis shows that for the case of full calibration, the stochastic ADC design is suboptimal. However, in [9] no calibration is used because the fabricated reference voltages are essentially estimated. We expect the stochastic ADC to be suboptimal comparing to the MSE-optimal design in the no-calibration case as well because its design leads to too many wasted fabricated reference voltages.*

■ 3.5.3 Technology scaling

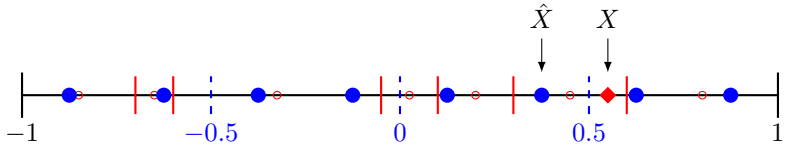
In this section we discuss how MSE, maximum quantization cell size and maximum quantization error scales as we fabricate more and more comparators in a fixed area. Section 3.3.1 shows that in classical quantization, when $m = n + 1$, for an ideal uniform ADC, MSE scales with the number of quantization points n as $1/n^2$, and S_{I} and S_{D} scales with the number of quantization points as $1/n$. Therefore, more ideal comparators improves quantization accuracy. However, with process variations, the smaller the comparator, the larger the reference voltage offsets. Therefore, it is unclear whether more but noisier comparators improve quantization accuracy, and in this section we show that with calibration, more but noisier comparators again improves quantization accuracy, although at a slower rate.

Remark 3.19. *Given a b-bit ADC, increasing n means we increase the redundancy factor r, which is defined in Section 3.5.2.*

As large n leads to large σ, we focus our attention to uniform input distribution with $\sigma > \sigma_0$ (cf. Theorem 3.4 for the definition of σ_0), where the MSE-optimal design



(a) A 2-bit ADC with $m = 4$.



(b) A 2-bit ADC with $m = 8$.

Figure 3-12: The quantization regions and reconstruction points of two 2-bit ADCs with different reproduction values. Both ADCs have exactly the same set of reference voltages, which are indicated by the solid vertical lines, and the reproduction values are indicated by the solid dots. The dashed vertical lines indicate an evenly spaced grid of [-1, 1], and the small circles indicate the midpoint of each quantization cell formed by the reference voltages.

corresponds to $\tau(x) = \delta(x)$. As mentioned in Remark 3.15, $\tau(x) = \delta(x)$ corresponds to fabricating identical comparators with the same reference voltage, possibly leading to simpler circuit implementations.

When $\tau(x) = \delta(x)$, $\lambda(x)$ is simply the Gaussian p.d.f., i.e.,

$$\lambda(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}.$$

Then over range [-1, 1],

$$f_{\max}^{[-1,1]} = \lambda(0) = \frac{1}{\sqrt{2\pi}\sigma},$$
$$f_{\min}^{[-1,1]} = \lambda(\pm 1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}},$$

where $e^{-\frac{1}{2\sigma^2}}$ approaches 1 as σ increases. Therefore, when σ large enough,

$$\text{MSE} \approx 2\pi\sigma^2/n^2, \tag{3.31}$$
$$S_{\text{D}} \approx \sqrt{2\pi}\sigma \frac{\log n}{n}, \tag{3.32}$$
$$S_{\text{I}} \approx \sqrt{\pi/2}\sigma \frac{\log m}{n}. \tag{3.33}$$

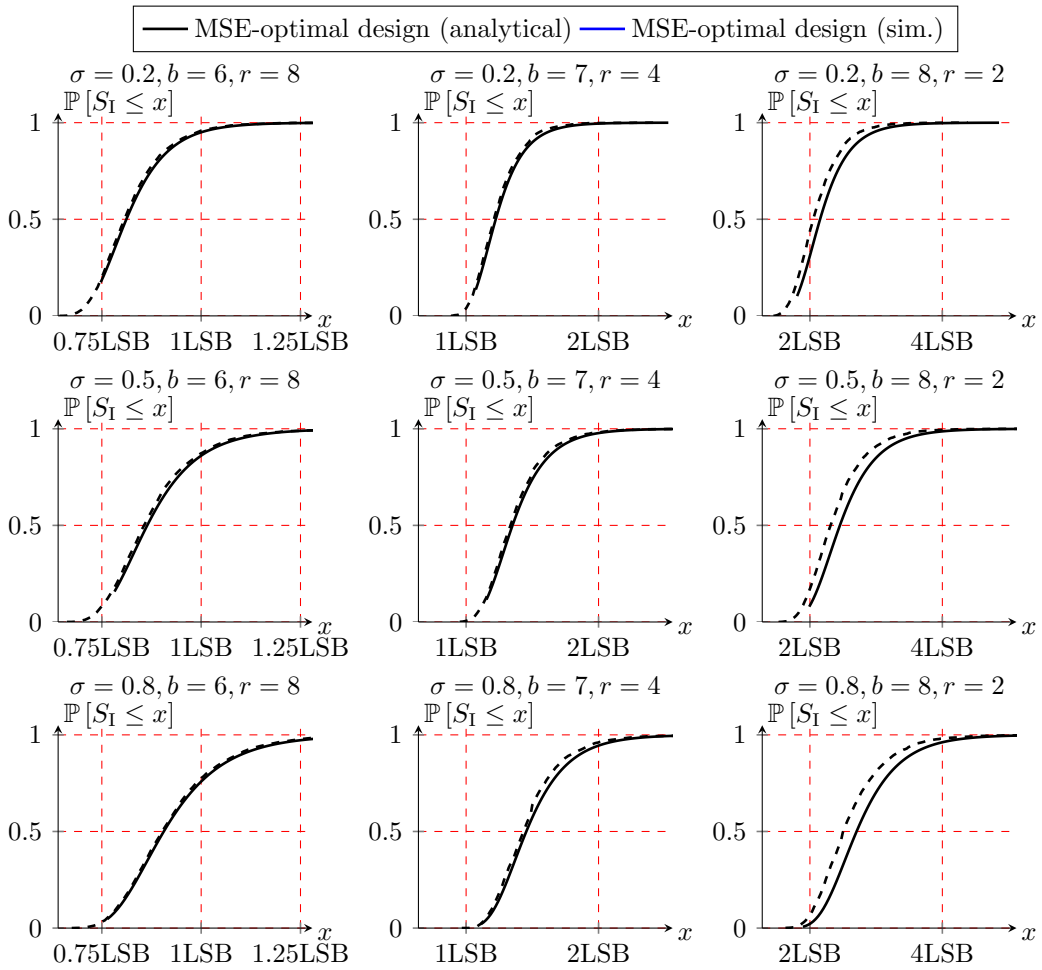


Figure 3-13: c.d.f.s of MSE-optimal designs for 6-bit Flash ADC with (approximately) the same number of comparators but different number of output codes at different values of σ. The dashed lines are c.d.f. computed based on (3.22) and the solid lines are the cumulative histograms of the S_{I} obtained from Monte-Carlo simulation.

Note that σ increases as the component size shrinks, which can be specified via the relationship [18,19] $\sigma^2 \propto 1/(\text{Component area})$. Ignoring the wiring overhead, then the number of components n is inversely proportional to the component area, i.e.,

$$n \propto 1/(\text{Component area}) \propto \sigma^2. \tag{3.34}$$

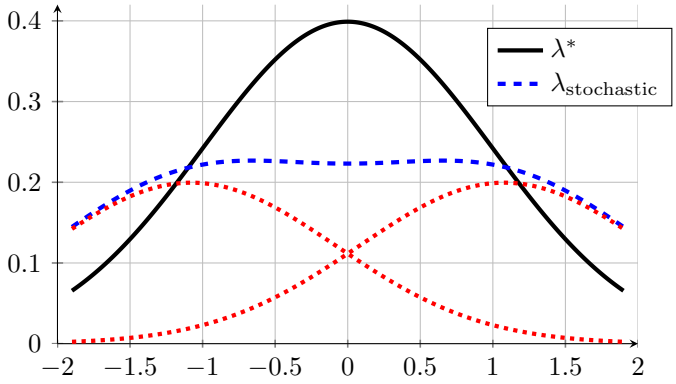


Figure 3-14: Comparison of the optimal λ^* with the stochastic ADC density $\lambda_{\text{stochastic}}$. The two dotted lines show the noisy partition point densities corresponding to $\delta\left(x-1.078\right)/2$ and $\delta\left(x+1.078\right)/2$, which are $\{\phi(x\pm 1.078)/2\}$ and sum to $\lambda_{\text{stochastic}}$.

	MSE	S_{D}	S_{I}
Classical	$\Theta\left(1/n^2\right)$	$\Theta\left(1/n\right)$	$\Theta\left(1/n\right)$
Process variation	$\Theta\left(1/n\right)$	$\Theta\left(\log n/\sqrt{n}\right)$	$\Theta\left(\log m/\sqrt{n}\right)$

Table 3.1: The scaling of quantization accuracy for ADC in classical and process variation settings.

Therefore, setting $\sigma=\Theta\left(\sqrt{n}\right)$, (3.31) to (3.33) indicate

$$\text{MSE}=\Theta\left(1/n\right), \tag{3.35}$$

$$S_{\text{D}}=\Theta\left(\frac{\log n}{\sqrt{n}}\right), \tag{3.36}$$

$$S_{\text{I}}=\Theta\left(\frac{\log m}{\sqrt{n}}\right). \tag{3.37}$$

We summarize the scaling of MSE, S_{D} and S_{I} in both the classical and process variation settings in Table 3.1.

Remark 3.20. *Taking wiring and other components in the circuit implementation into account, our results indicate that as long as σ increases at a speed slower than $n/\log n$, building more comparators reduces MSE, INL and DNL. Therefore, given a total silicon area, we only need to allocate more than $\Theta\left(\log^2 n/n\right)$ fraction of the area to comparators, making each comparator having an area of more than $\Theta\left(\log^2 n/n^2\right)$ and hence correspond to σ less than $\Theta\left(logn/n\right)$. The rest of silicon area can be used for other circuit components such as wiring and calibration logic.*

Remark 3.21. *It is conceivable that as we continue scaling down the size of comparators, the process variation may increase faster than the relationship indicated in (3.34), and may no longer be Gaussian distributed. Then in this new regime (3.35) to (3.37) would*

not hold, and we need to analyze the problem by taking the new statistical property of the process variation into account.

■ 3.6 Concluding remarks

In this chapter we investigate the problem of building a reliable Flash ADC from imprecise comparators. We formulate this problem as scalar quantization with noisy partition points, and use high resolution analysis and order statistics to analyze fundamental limits in terms of MSE, maximum quantization cell size and maximum quantization error. Our designs, obtained based on minimizing the MSE, is effective in reducing maximum quantization error, as demonstrated by both analytical and simulation results. In addition, our scaling analysis shows that building more but noisier comparators is beneficial for quantization accuracy.

Our research has shown the promise of building Flash ADC from imprecise comparators, and to take it for practical implementation, we need to address the following issues:

- calibration: in some current implementations [8] is implemented on-chip, which occupies large amount of silicon area, and does not scale as we increase the number of comparators. Therefore, it is desirable to move calibration off-chip, as it does not need to be conducted often;
- dynamic performance: as mentioned in Section 3.1, we only analyze the static performance of ADCs while in practice, the dynamic performance such as bandwidth and timing errors are also important;
- power: with more comparators, the power consumption of an ADC is likely to go up, and it is important to understand how power consumption scales with the number of comparators.

The calibration issue motivates the problem of calibration-free or partial-calibration ADC design. In this setting, we can assume a subset of the comparators are calibrated, with the two extreme cases being no calibration and full calibration, and analyze how the ADC performance degrades as we reduce the amount of calibration.

Another research direction that could lead to better ADC designs is reconfiguration, which is used in [8] to disable comparators after fabrication. While this potentially reduces the quantization accuracy, it is beneficial for power-saving due to smaller number of active comparators. A good understanding on this trade-off between quantization accuracy and power consumption could lead to better low-power ADC designs.

Finally, while in this work we focus our attention on Flash ADC, our results can be view as a building block to other types of ADCs that have the Flash ADC as sub-components, such as the pipelined ADC. Extending the analysis to other relevant ADC architectures may lead to new trade-offs and help to improve more ADC designs.

Chapter 4

Designing digital circuit with faulty components

4.1 Introduction

The challenge of component variations mentioned in Section 3.1 appears in not only in analog circuits but also digital circuits. Due to the nature of digital circuits, component variations often exhibit as component failures or component errors. Again, redundancy and reconfiguration provides a way to combat component failures or errors, and in this chapter we offer a preliminary investigation on various aspects of this approach, with our focus on combinatorial circuits with possible component failures.

The idea of introducing on-chip redundancy, or redundancy in combinatorial circuit design was first investigated in [25] and later by [26–28], where the focus was to achieve reliable computation from unreliable components, such as noisy gates, and errors occurrences could be dynamic[1] for each component. These analysis shows a relative large amount of redundancy is needed to achieve reliable computation.

However, in practice, fabrication defects are often static, i.e., fixed after fabrication, and circuit designers use the approach of redundancy and reconfiguration after fabrication for yield improvement [29]. In this approach, redundancy is usually introduced at the design stage, and implemented by fabrication. Reconfiguration can happen right after fabrication, where circuits are altered via Engineering Change Order (ECO), or it can happen when the device is boot-up and after some built-in self-test (BIST) is conducted. The BIST approach is popular for memory design due to the cost of in-factory test and reconfiguration [30].

While there exists design and analysis for redundancy and reconfiguration in special cases, especially random-access memory (RAM) [31–34], existing research usually relies on specific assumptions about the underlying circuit implementation. In this chapter we adopt the static fabrication error model, and take an abstract approach to analyze the problem of circuit design with redundancy and reconfiguration. For a given reliability requirement, we analyze the amount of additional resource needed in terms of both redundancy and reconfigurability, and the fundamental trade-off between the two. These results not only provide performance limits but also propose designs that achieve reliability with relatively small amount of additional resource usage.

The rest of the chapter is organized as follows. In Section 4.2, we propose a mathematical model for the reconfigurable redundant circuit and identify design settings of interest. Then we present analysis regarding the deterministic and probabilistic error correction settings in Section 4.3 and Section 4.4 respectively, and finally conclude by discussing the various implications of our analysis in Section 4.5.

[1] *Dynamic* means for each use of an element, the error occurs independently with certain probability.

4.2 Problem formulation

In this section we first propose a redundant circuit model in Section 4.2.1, which is motivated by existing circuit designs with redundancy and reconfiguration. Then in Section 4.2.2 we introduce the measures of resource usage, in terms of redundancy and reconfigurability. With these resource usage measures, we categorize design settings into two classes in Section 4.2.3, which lead to different performance trade-offs in terms of resource usage.

4.2.1 The redundant circuit model

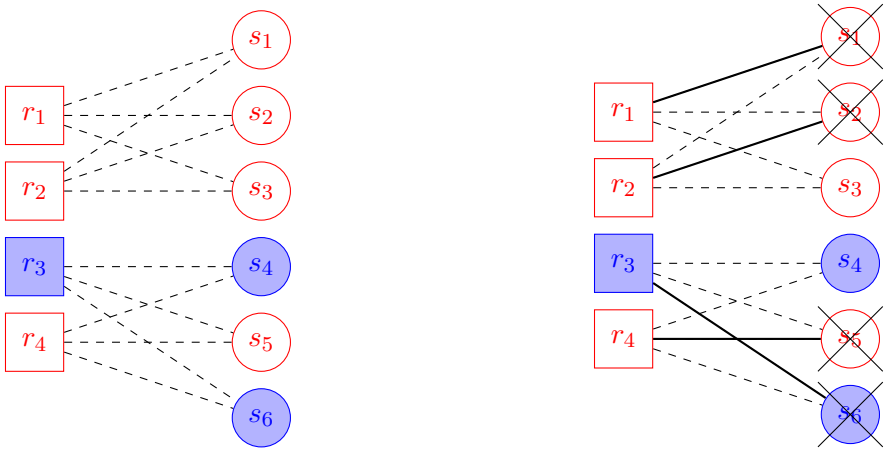
A circuit consists of a set of circuit elements and wires connecting these elements to achieve some functionality, and we call these elements *functional elements*. Depending on the application, the set of functional elements could be transistors, gates (AND gate, OR gate, ...), or circuit components (CPU, memory, ...). A circuit with redundancy and reconfiguration contains redundant elements in addition to functional elements, where redundant elements are introduced to replace failed functional elements. The replacement process can be achieved by enabling reconfigurable wires, which re-route inputs and outputs of the element being replaced to the redundant element. In practical implementations, due to routing constraints, a redundant element may not replace any functional element, but rather functional elements in a certain subset (e.g., the ones that are not far from the redundant element). Based on these observations, we propose the following redundant circuit model that allows us to capture the aspects of redundancy, reconfiguration and routing constraints. We consider all wiring to be reliable and restrict our attention to the case of circuit element failures, where each functional or redundant element fails with some probability.

Given a finite non-empty set of circuit elements $\mathcal{X}$, we say a circuit has k functional elements $s_1, s_2, \ldots, s_k$, where s^k is chosen from a set of possible configurations $\mathcal{S} \subset \mathcal{X}^k$ (e.g., the largest set of configurations is simply $\mathcal{S} = \mathcal{X}^k$). In a *redundant circuit*, we add m redundant elements $r_1, r_2, \ldots, r_m$ and connect these redundant elements to the k elements $s_1, s_2, \ldots, s_k$ via a configurable interconnect $\mathcal{E}$. By default all wires in the interconnect are disabled. After fabrication, we test all elements in the circuit to obtain the set of failed functional and redundant elements $\mathcal{V}_\mathrm{f}$ and $\mathcal{U}_\mathrm{f}$ respectively, and enable a subset of wires in the interconnect during reconfiguration. We say that the redundant element r_i *corrects* the functional element s_j if there exists an enabled wire in $\mathcal{E}$ between r_i and s_j.

Based on the above description, we can model a redundant circuit and its error correction via graph-theoretic concepts.

Definition 4.1 (Graph-theoretic model for the redundant circuit). *A redundant circuit model* $\mathcal{C} \triangleq \mathcal{C}(s^k, r^m, \mathcal{G})$ *consists of functional elements* s^k*, redundant elements* r^m *and a bipartite graph* $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$*, where* $\mathcal{U} = \{r_1, r_2, \ldots, r_m\}$*,* $\mathcal{V} = \{s_1, s_2, \ldots, s_k\}$ *and* $\mathcal{E} \subseteq \{(r, s), r \in \mathcal{U}, s \in \mathcal{V}\}$ *are the edges representing the configurable interconnect.*

Given the set of failed functional and redundant elements $\mathcal{V}_\mathrm{f}$ *and* $\mathcal{U}_\mathrm{f}$*, a redundant circuit correct all its errors if and only if there exists a* $\mathcal{V}_\mathrm{f}$*-saturated matching in the subgraph* $\mathcal{G}_\mathrm{c} = (\mathcal{U} \setminus \mathcal{U}_\mathrm{f}, \mathcal{V}_\mathrm{f}, \mathcal{E}_\mathrm{c})$*, where* $\mathcal{E}_\mathrm{c}$ *are all edges in* $\mathcal{G}$ *that have one end point in* $\mathcal{U} \setminus \mathcal{U}_\mathrm{f}$ *and the other in* $\mathcal{V}_\mathrm{f}$*.*



(a) A redundant circuit that has two types of circuit elements (shaded and non-shaded), with six functional elements and four redundant elements. The dashed lines represent the reconfigurable interconnect.

(b) Circuit after test and reconfiguration, where the failed functional elements are $\mathcal{V}_{\mathrm{f}} = \{s_1, s_2, s_5, s_6\}$, and are replaced by r_1, r_2, r_4 and r_3 accordingly, as indicated by the matching consists of solid lines.

Figure 4-1: Example of an redundant circuit model and its reconfiguration process.

Given this graph-theoretic model, the reconfiguration process can be seen as finding a $\mathcal{V}_{\mathrm{f}}$-saturated matching in $\mathcal{G}_{\mathrm{c}}$ and then enabled the edges in the matching.

Example of the redundant circuit model and its reconfiguration process are shown in Fig. 4-1a and Fig. 4-1b. In Fig. 4-1a, there is an edge between the shaded redundant element r_3 and non-shaded functional element s_5 as we may not know s^6 when designing the graph G or r^4. In Fig. 4-1b,

Remark 4.1. *We do not need to model wires connecting functional elements because we assume they are reliable and they do not change in the reconfiguration process.*

■ 4.2.2 Resource usage in the redundant circuit

In our redundant circuit model we introduce redundancy and reconfigurability to achieve reliability. However, both redundancy and reconfigurability require additional resource usage, in terms of the number of redundant elements and wires in the reconfigurable interconnect. These translate to additional silicon area usage[2]. When designing a redundant circuit, we would like to minimize the overhead due to redundancy and reconfiguration, and we first introduce the notions of *circuit redundancy* and *wiring complexity* to measure the overhead due to redundancy and reconfigurability respectively.

Definition 4.2 (Circuit redundancy). *For a given redundant circuit $\mathcal{C} = (s^k, r^m, \mathcal{G})$, its* circuit redundancy *is the ratio of the number of redundant elements to the number of functional elements, i.e.,*

$$\rho(\mathcal{C}) \triangleq \frac{m}{k}.$$

[2]They may lead to additional power consumption as well if the reconfigurable interconnect consumes more power than regular wires.

Definition 4.3 (Wiring complexity). *For a given redundant circuit* $\mathcal{C} = (s^k, r^m, \mathcal{G})$*, its* wiring complexity *is the number of wires between the redundant and functional elements* $E(\mathcal{C})$ *normalized by the number of functional elements, i.e.,*

$$E(\mathcal{C}) \triangleq |\mathcal{E}|/k. \tag{4.1}$$

As we shall see, given a certain reliability requirement, there may be a trade-off between these two types of resource usages, i.e., between circuit redundancy and wiring complexity, and we define the achievable region of the wiring complexity and circuit redundancy to capture this trade-off.

Definition 4.4 (Achievable region of (E, ρ)**).** *A pair* (E, ρ) *is achievable if there exists a sequence of circuits* $\{\mathcal{C}_k, k \in \mathbb{Z}^+\}$ *such that*

$$\begin{cases} E & = \lim_{k\to\infty} E(\mathcal{C}_k) \\ \rho & = \lim_{k\to\infty} \rho(\mathcal{C}_k) \end{cases}.$$

■ 4.2.3 The redundant circuit design settings

As mentioned, we are interested in the trade-off between circuit redundancy and wiring complexity, and this trade-off behaves differently, depending on the different amount of information available at the design stage. In this section we identity the following two different design settings that correspond to two different design scenarios.

General-purpose setting: design $\mathcal{G}$ **and** r^m **based on** $\mathcal{S}$**.** We design $\mathcal{G}$ and r^m with only knowledge about $\mathcal{S}$, i.e., we do not know the exact s^k to fabricate.

Application-specific setting: design $\mathcal{G}$ **based on** $\mathcal{S}$ **and** r^m **based on** s^k**.** We design $\mathcal{G}$ with only knowledge about $\mathcal{S}$, but design r^m after knowing the exact s^k to fabricate.

Depending whether $|\mathcal{S}| = 1$, these two design settings correspond to different level of customization in redundant circuit design, as shown in Table 4.1.

In the rest of this chapter, we evaluate the error correction capability of a redundant circuit in two error correction settings, deterministic and probabilistic, in Section 4.3 and Section 4.4 respectively. The main objective of our investigation is to characterize the maximal achievable region for redundant circuits under both deterministic and probabilistic error correction settings and both general-purpose and application-specific settings.

Setting	$\lvert\mathcal{S}\rvert = 1$	$\lvert\mathcal{S}\rvert > 1$
General-purpose	given s^k, design both interconnect and redundancy	design both redundancy and interconnect for all possible $s^k \in \mathcal{S}$
Application-specific		design interconnect for all possible $s^k \in \mathcal{S}$, and then given s^k, design redundancy

Table 4.1: The design settings correspond to factory designs for different parts of the redundancy circuit.

Name	Domain	Meaning
k	$\mathbb{Z}^+$	The number of functional elements.
m	$\mathbb{Z}^+$	The number of redundant elements.
t	$\mathbb{Z}^+$	The number of element failures to correct.
d	$\mathbb{Z}^+$	The maximum number of functional elements that a redundant element can connect to.

Table 4.2: System parameters for reliable circuit design in the deterministic error correction setting.

■ 4.3 Deterministic error correction setting

In this section we investigate the design of redundant circuit in the deterministic error correction setting. By deterministic error correction we mean the case of guaranteed correction of a certain number of functional element failures. More specifically, we say a redundant circuit $\mathcal{C}$ is *t-correcting for $\mathcal{S}$* if it corrects any t circuit element failures for any $s^k \in \mathcal{S}$. Furthermore, we simply say a redundant circuit is t-correcting when it is t-correcting for $\mathcal{S} = \mathcal{X}^k$.

As mentioned in Section 4.2, we are interested in the trade-off between wiring complexity and circuit redundancy, and we define the notion of capacity region to capture the optimal trade-off of these two quantities.

Definition 4.5 (Capacity region in deterministic error correction setting). *Given $\mathcal{S}$ and a design setting (general-purpose or application-specific) and a set $\mathcal{R}$ of (E, ρ) pairs, if (E, ρ) is achievable for some t-correcting redundant circuit for $\mathcal{S}$ if and only if $(E, \rho) \in \mathcal{R}$, then we say the t-correcting redundant circuit for $\mathcal{S}$ has* capacity region $\mathcal{R}$.

Noting that in practice a redundant element may not be able to connect to too many functional element, we impose a constraint on the graph $\mathcal{G}$ so that each node $r \in \mathcal{U}$ has degree at most d, with the unconstrained case simply being the special case $d = k$. With this observation, we summarize the system parameters in Table 4.2.

Given $\mathcal{S}$, k, t and d, we denote the capacity region in the general-purpose setting and the application-specific setting as $\mathcal{R}_{\mathrm{GP}}(\mathcal{S}, k, t, d)$ and $\mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d)$ respectively.

We first state the following simple fact, without proof, regarding the achievable regions for both the general-purpose setting and the application-specific setting.

Proposition 4.1. *Given $\mathcal{S}$ and system parameters in Table 4.2, if $(E, \rho) \in \mathcal{R}_{\mathrm{GP}}(\mathcal{S}, k, t, d)$, then $(E, \rho) \in \mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d)$.*

Given $\mathcal{S}$, define the set of possible elements at each index i as $\mathcal{A}_i(\mathcal{S})$ and its size as n_i, i.e.,

$$\mathcal{A}_i(\mathcal{S}) \triangleq \left\{ s_i : s^k \in \mathcal{S} \right\}, \tag{4.2}$$

$$n_i(\mathcal{S}) \triangleq \left| \mathcal{A}_i(\mathcal{S}) \right|. \tag{4.3}$$

Furthermore, we define the average of $\{n_i, 1 \le i \le k\}$ as $\bar{n}(\mathcal{S}) = \sum_{i=1}^{k} n_i / k$.

With these definitions, we characterize the capacity region for the general-purpose setting in the following theorem.

Theorem 4.2 (Capacity region for the general-purpose setting). *The capacity region for a t-correcting redundant circuit for $\mathcal{S}$ with k functional elements and degree constraint d under the general-purpose setting is*

$$\mathcal{R}_{\mathrm{GP}}(\mathcal{S}, k, t, d) = \{(E, \rho) : E \geq t\bar{n}(\mathcal{S}), \rho \geq t\bar{n}(\mathcal{S})/d\}.$$

Remark 4.2. *For the case of $\bar{n}(\mathcal{S}) = |\mathcal{X}|$, the capacity region $\mathcal{R}_{\mathrm{GP}}(\mathcal{S}, k, t, d)$ is the region marked by dashed lines in Fig. 4-2.*

We defer the detailed analysis to Section 4.3.1 and present some results on the achievable region for the application-specific setting, as the exact characterization for the capacity region of this setting is still open. For simplicity, we only consider the $\mathcal{S} = \mathcal{X}^n$.

Theorem 4.3 (Achievable region and converse for the application-specific setting). *When $\mathcal{S} = \mathcal{X}^k$, the achievable region for a t-correcting redundant circuit with k functional elements and degree constraint d under the application-specific setting satisfies*

When $d < |\mathcal{X}|$,

$$\mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d) \supseteq \{(E, \rho) : E \geq t, \rho \geq t\};$$

When $d \geq |\mathcal{X}|$,

$$\mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d) \supseteq \left\{(E, \rho) : E \geq t\,|\mathcal{X}|\,/d, \rho \geq t, \rho \geq \frac{d - |\mathcal{X}|}{d - d\,|\mathcal{X}|}(E - t) + t\right\}.$$

In addition, if $E < t$ or $\rho < t/d$, then $(E, \rho) \notin \mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d)$.

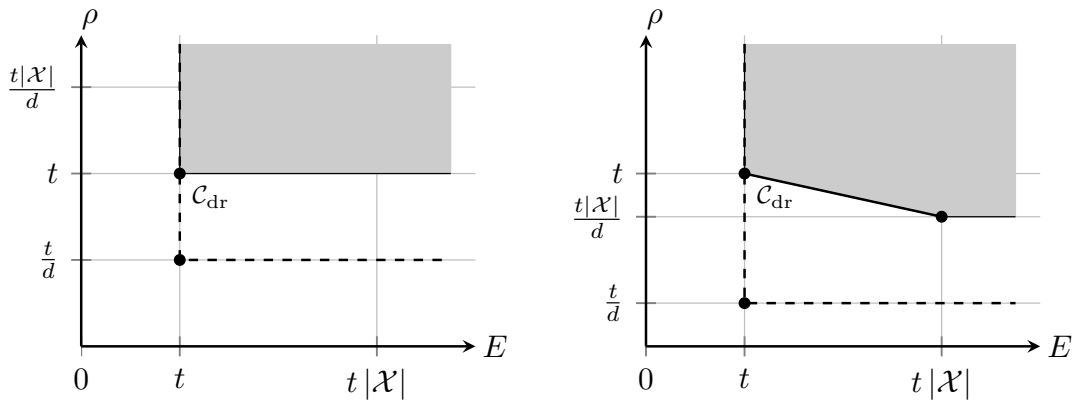
Theorem 4.3 is summarized in Fig. 4-2. and we defer the detailed discussion to Section 4.3.2.

During the process of achievable region investigation, we discover the following two design techniques, illustrated in Fig. 4-3, *spatial sharing* and *circuit merging*, that enable us to generate new circuit designs with different performance parameters from existing circuit designs.

Lemma 4.4 (Spatial sharing). *Given two t-correcting redundant circuit $\mathcal{C}_1$ and $\mathcal{C}_2$ that achieves (E_1, ρ_1) and (E_2, ρ_2) respectively, then for any $0 \leq \lambda \leq 1$, we can obtain a t-correcting redundant circuit that achieves $(E_\lambda, \rho_\lambda)$, where*

$$E_\lambda = \lambda E_1 + (1 - \lambda)E_2,$$
$$\rho_\lambda = \lambda \rho_1 + (1 - \lambda)\rho_2.$$

Lemma 4.5 (Circuit merging). *Given a t_1-correcting redundant circuit $\mathcal{C}_1$ and a t_2-correcting redundant circuit $\mathcal{C}_2$ that achieves (E_1, ρ_1) and (E_2, ρ_2) respectively, then we*



(a) $d < |\mathcal{X}|$ (b) $d \geq |\mathcal{X}|$

Figure 4-2: Characterization of the capacity region for the application-specific setting: the shaded area is the achievable region, and the dashed line is one (not-necessarily tight) outer bound of the capacity region.

can obtain a $t_1 + t_2$*-correcting redundant circuit that achieves* (E, ρ)*, where*

$$\begin{aligned} t &= t_1 + t_2, \\ E &= E_1 + E_2, \\ \rho &= \rho_1 + \rho_2. \end{aligned}$$

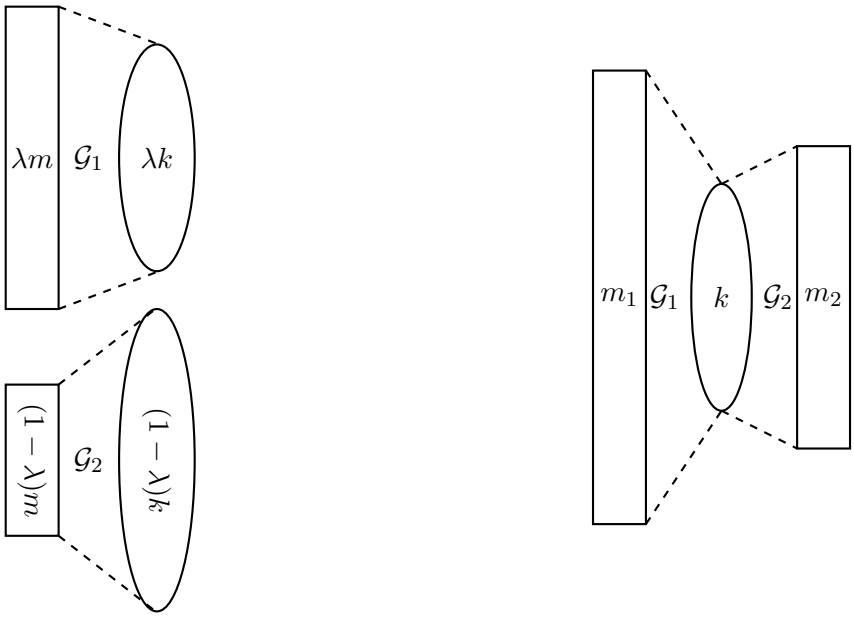
We omit the proofs for Lemma 4.5 and Lemma 4.4 as they are straightforward.

■ 4.3.1 Analysis for the general-purpose setting

In this section we characterize the capacity region of (E, ρ) for the general-purpose setting by first proposing a redundant circuit design and then showing that it is indeed optimal.

The following redundant circuit construction algorithm finds the functional element with the smallest *normalized degree* $\widetilde{\deg}(s_i) \triangleq \deg(s_i)/n_i(\mathcal{S})$, and then for each of its possible element type $x \in \mathcal{A}_i(\mathcal{S})$, we connect an available redundant element (with degree less than d) of the same type to it. We repeat this process until all functional elements have normalized degree t, which guarantees the redundant circuit is t-correcting. It is not difficult to see that for a given k, Algorithm 2 produces a redundant circuit $\mathcal{C}_k$ such that

$$\begin{cases} \rho(\mathcal{C}_k) & = \dfrac{t\bar{n} + |\mathcal{X}|/k}{d}, \\ E(\mathcal{C}_k) & = t\bar{n} + |\mathcal{X}|\, d/k. \end{cases}$$



(a) Spatial sharing. (b) Circuit merging.

Figure 4-3: Two circuit design techniques.

Therefore, given a sequence of redundant circuits $\{\mathcal{C}_k, k \in \mathbb{Z}^+\}$, we have

$$\lim_{k\to\infty} \rho(\mathcal{C}) = \frac{t\bar{n}}{d}$$
$$\lim_{k\to\infty} E(\mathcal{C}) = t\bar{n}$$

for any $0 \le t, d \le k$.

The following lemma shows the construction provided by Algorithm 2 is indeed optimal.

Lemma 4.6 (Converse for the general-purpose setting). *Given d and $\mathcal{S}$, any t-correcting redundant circuit for the general-purpose setting satisfies*

$$\rho \le \frac{t\bar{n}}{d} \text{ and } E \ge t\bar{n}. \tag{4.4}$$

We defer the detailed proof to Appendix A.4.1.

■ 4.3.2 Analysis for the application-specific setting

In this section we first note that the capacity region for t-correcting redundant circuits are convex, which following follows from Lemma 4.4 directly.

Corollary 4.7. *The capacity region for t-correcting redundant circuits is convex.*

Now we derive the inner bound of the capacity region for the application-specific setting based on a few specific designs.

Algorithm 2 Construction of the optimal $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ for the general-purpose setting

Initialization: $done \leftarrow False$, $\mathcal{V} \leftarrow \{s_1, s_2, \ldots, s_k\}$, $\mathcal{U} \leftarrow \emptyset$, $\mathcal{E} \leftarrow \emptyset$,
while NOT $done$ **do**
  $i^* \leftarrow \arg\min_i \widetilde{\deg}(s_i)$
  if $\widetilde{\deg}(s_{i^*}) < t$ **then**
    for $x \in \mathcal{A}_{i^*}(\mathcal{S})$ **do**
      if Exists $r \in U$ such that $\deg(r) < d$ and r is a type-x element **then**
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s_{i^*}, r)\}$
      else
        Let r an element r with type x
        $\mathcal{U} \leftarrow \mathcal{U} \cup \{r\}$
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s_{i^*}, r)\}$
      end if
    end for
  else
    $done \leftarrow True$
  end if
end while

Lemma 4.8 (Designs for application-specific setting).

$$(E = t, \rho t) \in \mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d) \tag{4.5}$$

$$(E, \rho) \in \mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d) \ \text{ if } E \geq t\,|\mathcal{X}|\,, \rho \geq t\,|\mathcal{X}|\,/d. \tag{4.6}$$

Proof. (4.5) is achieved by the redundant circuits design $\mathcal{C}_{\mathrm{dr}}$, where each functional element has its dedicated redundancy, as shown in Fig. 4-4. (4.6) follows from the fact that for any $\mathcal{S} \subset \mathcal{X}^k$,

$$\mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d) \supseteq \mathcal{R}_{\mathrm{GP}}\left(\mathcal{X}^k, k, t, d\right). \qquad \square$$

We also derive the outer bound for the capacity region of application-specific setting.

Lemma 4.9 (Outer bound for the capacity region of application-specific setting). *In application-specific setting,*

(a) any circuit that is t-correcting with $E = t$ must have $\rho \geq t$,

(b) $(E, \rho) \notin \mathcal{R}_{\mathrm{AS}}(\mathcal{S}, k, t, d)$ if $E < t, \rho < t/d$.

Proof. We defer the proof of (a) to Appendix A.4.2. (b) follows from the observation that for any $\mathcal{S} \subset \mathcal{X}^k$, $\mathcal{R}_{\mathrm{GP}}\left(s^k, k, t, d\right) \supseteq \mathcal{R}_{\mathrm{AS}}\left(\mathcal{X}^k, k, t, d\right)$, where s^k is any element in $\mathcal{S}$. $\square$

Finally, Lemma 4.4 and Lemma 4.8 leads to the achievable region in Theorem 4.3, while Lemma 4.9 leads to the converse in Theorem 4.3.

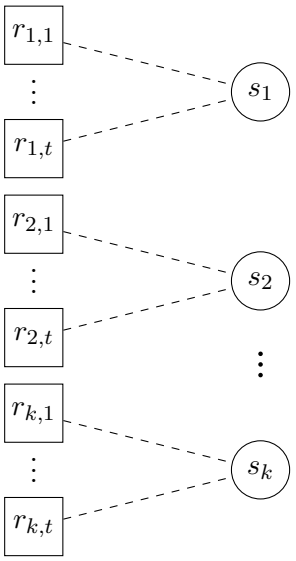


Figure 4-4: Redundant circuit $\mathcal{C}_{\text{dr}}$: each functional element has t dedicated redundant elements.

Comparing the general-purpose setting and the application-specific setting

In this section we show two examples to demonstrate the differences between the general-purpose setting and the application-specific setting.

The first example shows how $|\mathcal{X}|$ impacts the error correcting capability differently in these two settings.

Example 4.1. *Given $\mathcal{S} = \mathcal{X}^k$, for a redundant circuit $\mathcal{C}$ with adjacency matrix*

$$G(k = 2, m = 2, d = 2) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

it is 1-correcting for both the general-purpose setting and the application-specific setting when $|\mathcal{X}| = 2$, but 1-correcting only for the application-specific setting when $|\mathcal{X}| = 3$.

The second examples shows the error correcting capability in application-specific setting in general depends on $\mathcal{S}$ and not just $\bar{n}(\mathcal{S})$.

Example 4.2. *Given $k = 4, d = 2, t = 1$ and $\mathcal{X} = 1, 2$, for application-specific setting, a redundant circuit $\mathcal{C}$ with adjacency matrix*

$$G(k = 4, m = 3, d = 2) = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

is 1-correcting when $\mathcal{S} = \left\{ \text{all } s^k \text{ with type } (3/4, 1/4) \right\}$ but not when $\mathcal{S} = \mathcal{X}^k$.

4.4 Probabilistic error correction setting

In this section we adopt a probabilistic error correction setting for the reliable circuit design problem. We assume the elements in the redundant circuit fail independently with probability ε, and define the event of a circuit failure, i.e., the existence of functional elements that cannot be corrected as follows:

$$\mathcal{F}\left(s^k, r^m, \mathcal{G}\right) \triangleq \left\{\text{Given } r^m \text{ and } \mathcal{G},\ s^k \text{ cannot be fully corrected}\right\},$$

then the error probability in the general-purpose setting and the application-specific setting can be expressed as

$$P_{\mathrm{e}}^{\mathrm{GP}}(\mathcal{S}) \triangleq \min_{r^m, \mathcal{G}} \max_{s^k \in \mathcal{S}} \mathbb{P}\left[\mathcal{F}\left(s^k, r^m, \mathcal{G}\right)\right]$$
$$P_{\mathrm{e}}^{\mathrm{AS}}(\mathcal{S}) \triangleq \min_{r^m} \max_{s^k \in \mathcal{S}} \min_{\mathcal{G}} \mathbb{P}\left[\mathcal{F}\left(s^k, r^m, \mathcal{G}\right)\right].$$

By the min-max inequality [35],

$$P_{\mathrm{e}}^{\mathrm{GP}}(\mathcal{S}) \geq P_{\mathrm{e}}^{\mathrm{AS}}(\mathcal{S}).$$

For a given design setting, we say a redundant circuit is ε-reliable for $\mathcal{S}$ if it achieves $P_{\mathrm{e}}(\mathcal{S}) \to 0$ in that setting as $k \to \infty$.

In our analysis, we first investigate the trade-off between circuit redundancy and wiring complexity when the circuit is consists of a single-type of elements in Section 4.4.1, then extend the results to multiple element types in Section 4.4.2.

As we shall see, for the circuits of interest in the probabilistic error correction setting, the amount of interconnect $|\mathcal{E}|$ is on the order of $k \log k$. As a result, we normalize the wiring complexity by $k \log k$ and introduce a new definition of wiring complexity.

Definition 4.6 (Wiring complexity in probabilistic error correction setting). *The wiring complexity in probabilistic error correction setting is the same as Definition 4.3 except (4.1) is changed to*

$$E(\mathcal{C}) \triangleq |\mathcal{E}|/(k \log k).$$

Again, we introduce the notion of capacity region, which is the central object of interest in our analysis.

Definition 4.7 (Capacity region in probabilistic error correction setting). *Given $\mathcal{S}$, a design setting (general-purpose or application-specific) and a set $\mathcal{R}$ of (E, ρ) pairs, if (E, ρ) is achievable for some ε-reliable redundant circuit for $\mathcal{S}$ if and only if $(E, \rho) \in \mathcal{R}$, then we say the ε-reliable redundant circuit for $\mathcal{S}$ has* capacity region $\mathcal{R}$.

4.4.1 Analysis of redundant circuit with a single element type

In this section we investigate the problem of designing redundant circuit when there is only one type of element, i.e., $|\mathcal{X}| = 1$. In this case the general-purpose setting and the application-specific setting are equivalent. As we shall see later, results in this section serve as useful building blocks for analysis of more general cases.

Our main results are summarized in the following theorem.

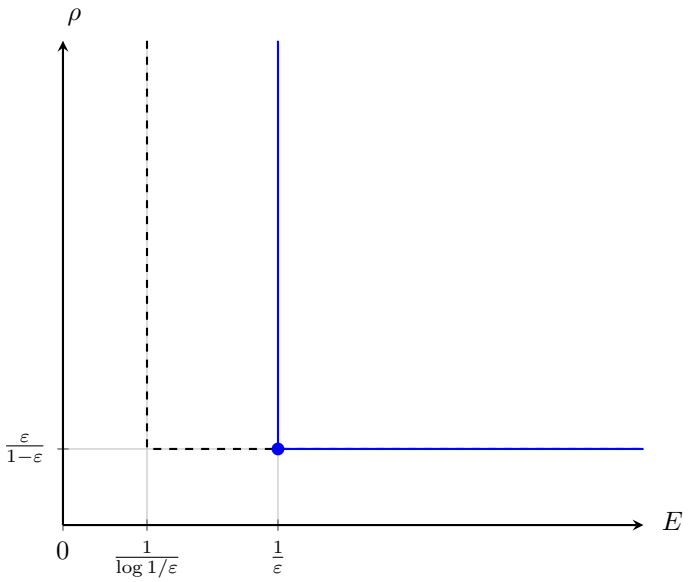


Figure 4-5: The redundancy-wiring complexity trade-off for redundant circuit with one type of element in the probabilistic error correction setting.

Theorem 4.10 (Redundancy-wiring complexity trade-off). *If $|\mathcal{X}| = 1$, then any ε-reliable redundant circuit requires*

$$\begin{cases} \rho(\mathcal{G}) & \geq \varepsilon/(1-\varepsilon) \\ E(\mathcal{G}) & \geq -1/\log \varepsilon \end{cases}. \tag{4.7}$$

In addition, for $|\mathcal{X}| = 1$, there exist different sequences of ε-reliable redundant circuits $\{\mathcal{C}_k, k \in \mathbb{Z}^+\}$ that achieve

$$(\rho, E) = (\varepsilon/(1-\varepsilon), 1/(1-\varepsilon)) \tag{4.8}$$

and

$$(\rho, E) = (\infty, -1/\log \varepsilon) \tag{4.9}$$

respectively.

The redundancy-wiring complexity trade-off in Theorem 4.10 is illustrated in Fig. 4-5. The design corresponds to (4.8) achieves minimum circuit redundancy ρ, but has a wiring complexity larger than the lower bound in (4.7). By contrast, the design corresponds to (4.9) achieves minimum wiring complexity, but has infinite circuit redundancy.

Remark 4.3. *Since*

$$\frac{1}{1-\varepsilon} - 1 \leq \frac{1}{\log(1/\varepsilon)} \leq \frac{1}{1-\varepsilon},$$

the gap between achievability and converse in E is at most 1.

To prove Theorem 4.10, We first show that there exists redundant circuit designs via random graph that achieves $(\rho, E) = (\varepsilon/(1-\varepsilon), 1/(1-\varepsilon))$ in the following lemma.

Lemma 4.11 (Redundant circuit via random bipartite graph). *Let the interconnect $\mathcal{G}_k$ of $\mathcal{C}_k$ be a random bipartite graph $\mathcal{G}(m, k, p_k)$ with edge probability*

$$p_k = \frac{\log k + w(k)}{m},$$

where $w(k) \to \infty$ as $k \to \infty$, then the sequence of redundant circuit $\{\mathcal{C}_k, k \in \mathbb{Z}^+\}$ achieves $(\rho, E) = (\varepsilon/(1-\varepsilon), 1/(1-\varepsilon))$.

Then we show with proper parameters, redundant circuit $\mathcal{C}_{\text{dr}}$ in Fig. 4-4 achieves $(\rho, E) = (\infty, -1/\log \varepsilon)$.

Lemma 4.12 (Probabilistic error correction of $\mathcal{C}_{\text{dr}}$). *The redundant circuit $\mathcal{C}_{\text{dr}}$ in Fig. 4-4 with $t = -\log \varepsilon$ achieves $(\rho, E) = (\infty, -1/\log \varepsilon)$.*

Finally we show the converse via the following two results.

Lemma 4.13 (Lower bound on redundancy). *Any ε-reliable redundant circuit requires $\rho \geq \varepsilon/(1-\varepsilon)$.*

Lemma 4.14 (Lower bound on redundancy). *Any ε-reliable redundant circuit requires $E \geq \frac{1}{\log(1/\varepsilon)}$.*

We defer all proofs in this section to Appendix A.5.

■ 4.4.2 Analysis of redundant circuit with multiple element types

In this section we extend the results in Section 4.4.1 to redundant circuit with multiple element types. Our first result shows a result analogous to Theorem 4.10 holds in the general-purpose setting.

Theorem 4.15. *For general-purpose setting, any ε-reliable redundant circuit satisfies*

$$\rho \geq \frac{\bar{n}(\mathcal{S})\varepsilon}{1-\varepsilon},$$
$$E(\mathcal{G}) \geq \frac{\bar{n}(\mathcal{S})}{\log(1/\varepsilon)},$$

where $\bar{n}(\mathcal{S})$ is defined in (4.3). In addition, there exist different sequences of ε-reliable redundant circuit $\{\mathcal{C}_k, k \in \mathbb{Z}^+\}$ that achieve

$$(\rho, E) = (\bar{n}(\mathcal{S})\varepsilon/(1-\varepsilon), \bar{n}(\mathcal{S})/(1-\varepsilon))$$

and

$$(\rho, E) = (\infty, \bar{n}(\mathcal{S})/\log(1/\varepsilon))$$

respectively.

We omit the proof due to its similarity to the proof of Theorem 4.10.

Regarding application-specific setting, while it is easy to propose redundant circuit designs, the performance analysis is more difficult. We leave this topic for future exploration.

■ 4.5 Concluding remarks

In this chapter we analyze the design of redundant circuit in both deterministic and probabilistic setting, focusing on the trade-off between circuit redundancy and wiring complexity. Our results indicate that to correct $t = \varepsilon k$ errors, in the deterministic setting $|\mathcal{E}| = \Theta\left(k^2\right)$ is necessary, while in the probabilistic setting $|\mathcal{E}| = k \log k$. Therefore, by allowing an asymptotically vanishing probability of error, we can reduce the wiring complexity dramatically from $\Theta\left(k^2\right)$ to $\Theta\left(k \log k\right)$. In addition to showing the advantage of considering probabilistic error correction, we also propose concrete redundant circuit designs such as $\mathcal{C}_{\text{dr}}$ in Fig. 4-4 and a design based on the random bipartite graph in Lemma 4.11, which achieves near-optimal redundancy-wiring complexity trade-off in the probabilistic error correction setting, as indicated in Theorem 4.10.

The results in this chapter represent a preliminary investigation of the problem of redundant circuit design, While our model enables mathematical analysis, there are a few important aspects in practical circuit design that need to be incorporated. The most important aspect is probably placement and routing, as our bipartite graph model does not capture the geometry location of circuit elements. Besides, in practice the component failure probability needs to be estimated, and may be correlated due to the clustering of fabrication defects. Therefore, a more realistic error model is called for. Nonetheless, our results provide insights on the fundamental problem of redundant system design subject to redundancy sharing constraints, and serves as a starting point for more realistic design and analysis.

Furthermore, a few research directions could offer new opportunities in reliable digital circuit design. First, in systems like Field-programmable gate array (FPGA), the elements are configurable, and hence it would be interesting to explore the case when functional and/or redundant elements are configurable, which may offer better performance trade-offs and lead to new designs. Second, in our model we assume homogeneous circuit components, where the redundancy and functional elements have the same reliability. It may be possible that building less reliable redundant elements at a lower cost (in terms of area, power, etc.) actually leads to better performance trade-off between reliability and resource usage. Finally, in our probabilistic setting we show it is beneficial to tolerate an asymptotically vanishing probability of error. However, it may be even more beneficial to tolerate a fixed probability of error, say 0.2. This calls for non-asymptotic analysis of the performance of redundant circuits.

Part II

Scheduling parallel tasks with variable response time

Chapter 5

On scheduling parallel tasks

Just as fault-tolerant computing aims to create a reliable whole out of less-reliable parts, large online services need to create a predictably responsive whole out of less-predictable parts.

J. Dean and L. A. Barroso, *The tail at scale*
Communications of the ACM, 2013

In this chapter we investigate the role of replication in scheduling a computation job that consists of a collection of parallel tasks[1], where there is no interdependency among these tasks, and the completion time of each task is stochastic. This problem occurs in many contexts, such as cloud computing and crowd sourcing. We are interested in the latency of obtaining results for the entire collection of task, which is determined by the completion time of the task that finishes last. In particular, we aim to understand how *replication* (executing the same task more than once) reduces the latency, and impacts resource usage, which provides guidance to system users on when and how to use replication. To the best of our knowledge, we establish the first theoretical analysis of task replication, with the system model and relevant performance measures proposed.

The rest of the chapter is organized as follows. After introducing the motivating applications in Section 5.1, we review the prior practical system implementations in Section 5.2 and point out the need for a corresponding theory. Then we formulate the problem in Section 5.3 and propose the performance measures of interest. Based on our formulation, in Section 5.4 we demonstrate that replication can indeed be very useful via a simple example. Then in Section 5.5, we describe the two approaches we use to analyze the scheduling problem. The detailed analysis are presented in Chapter 6 and Chapter 7 respectively. Finally, we conclude this chapter with a discussion on the implication of our results in scheduling policy design in Section 5.5.

■ 5.1 Motivating applications

The problem of executing a collection of tasks in parallel appears in many distributed systems. In this section we mention two emerging applications: cloud computing and crowd sourcing.

Executing parallel tasks in cloud computing

One of the typical scenarios in cloud computing is large scale computation in a data center with a large number of computers, which is pioneered by companies like Google with the support from distributed computing frameworks such as MapReduce [36] and

[1]Note that in some literature the usage of "task" and "job" is reversed, i.e., a task is consists of a collection of jobs.

	50%ile latency	95%ile latency	99%ile latency
One task finishes	1ms	5ms	10ms
All tasks finish	40ms	87ms	140ms

Table 5.1: Execution times for a job consists many tasks in the Google data center.

Percolator [37], and distributed storage systems such as Google File System [38] and BigTable [39].

Executing a large collection of parallel tasks is an important category of computation in data centers, which is sometimes called "*embarrassingly parallel*" computation [40]. While appearing to be simplistic, embarrassingly parallel computation happens (either in part or in whole) in many non-trivial application, such as the "Map" stage of MapReduce, genetic algorithms, and the tree growth step of random forest. Furthermore, researchers aim to design algorithms that can be run in an embarrassingly parallel fashion so that they can be applied in a distributed setting easily, such as parallel alternating direction method of multipliers (ADMM) [41], and Markov Chain Monte-Carlo (MCMC) [42].

One key challenge for executing a large collection of tasks is the long overall response time. Due to co-hosting, virtualization and other hardware and network variations [43], the task execution time of computing nodes in a data center is subject to stochastic variations. At a result, latency determined by the slowest component could degrade the performance significantly. For example, Table 5.1, paraphrased from [43, Table 1], shows that for a single task, the 99-percentile latency (10ms) could be 10 times its median latency (1ms), and as a result, the latency of execution the entire collection of tasks, even *in parallel*, could increase to 140ms, much longer than the latency of a single task!

Executing parallel tasks in crowd sourcing

Crowdsourcing is a process that distributes tasks to a group of people, with the objective to achieve some common goal. Crowdsourcing exists in many forms. For example, crowdsourcing platforms such as Amazon Mechanical Turk [44] or oDesk [45] have been built to allow requesters post tasks at various prices and workers solve these tasks for a corresponding amount of compensation.

In many crowd sourcing scenarios, tasks are independent among each other and are assigned to different workers. Each worker may take a different amount of time to complete his/her task, and these completion times may be model as random variables. Therefore, again we have a collection of parallel tasks with stochastic execution time.

■ 5.2 Related prior work

The idea of replicating tasks has been recognized by system designers for parallel computation [46, 47], and is first adopted in cloud computing via the "backup tasks" in MapReduce [36]. A line of system work [48–51] further develops this idea to handle various performance variability issues in data centers.

While task replication has been adopted in practice, to the best of our knowledge, it has not been mathematically analyzed, and the trade-off between latency reduction and additional resource usage is unclear. By contrast, for scheduling without task replication, there

exists a considerable amount of work on *stochastic scheduling*, i.e., scheduling problems with stochastic processing time (e.g., [52–61]). In addition, existing system work [48–51] usually investigates the latency of all tasks from all different jobs in the system, while we analyze the latency of tasks from the same job, which is more meaningful from a user's perspective.

Finally, we note that using replication or redundancy to reduce latency has recently attracted attention in other application contexts such as data transfer [62–65].

■ 5.3 Problem formulation

In this section we first introduce the additional notation in this part of the thesis in Section 5.3.1, then we describe the system model in Section 5.3.2, and finally we propose the performance metrics in Section 5.3.3.

■ 5.3.1 Notation

In addition to the notation introduced in Section 1.1, we introduce the following notation in this part of the thesis.

We denote the upper end point of a c.d.f. F_X by

$$\omega\left(F_X\right) \triangleq \sup\left\{x : F_X(x) < 1\right\}. \tag{5.1}$$

Note that for $A = \min\{B, C\}$,

$$\omega\left(F_A\right) = \min\left\{\omega\left(F_B\right), \omega\left(F_C\right)\right\}.$$

We define the complement c.d.f., which may be more convenient to use than the c.d.f. sometimes.

$$\bar{F}_X(x) \triangleq 1 - F_X(x).$$

We denote one-sided limits from above and below $a \in \mathbb{R}$ as $x \to a^+$ and $x \to a^-$.

We denote the empirical cumulative distribution function based on $X_1, X_2, \ldots, X_n$ as $F_n(x)$, and define the p-quantile of a distribution as

$$x_p \triangleq F^{-1}\left(p\right) \triangleq \inf\left\{x \in \mathbb{R} : F(x) \geq p\right\}.$$

For $0 \leq p \leq 1$, we use $\bar{p}$ as a shorthand of the quantity $1 - p$.

We use $[a, \mathbf{t}]$ and $[\mathbf{t}, a]$ to denote the vector resulting from inserting an element a to the head and tail respectively of the vector $\mathbf{t}$.

■ 5.3.2 System model

Motivated by applications in Section 5.1, we are interested in the problem of executing a collection of *parallel tasks*. Since the resources in cloud computing and crowd sourcing systems are usually requested on-demand, we consider the case of having access to an unlimited pool of *computing nodes*. In the cloud computing context, each computing node is a computer (or a virtual machine), while in the crowd sourcing context, each node is a worker. We assume the executing time of each task on a computing node is i.i.d., where the variation of execution time comes from the variability in the computing nodes, as

discussed in Section 5.1. Mathematically, we are given a collection of n tasks, each with i.i.d. execution time distribution F_X.

A *scheduling policy* or *scheduler* assigns tasks to different nodes, possibly at different time instants. In this chapter, we focus our attention on the case that a scheduler can take the following actions:

1. AddTaskToNode: the scheduler requests a node to use from the pool of available nodes and sends a task to run on the node.

2. TerminateTask: the scheduler shuts down all nodes that are running the same task.

Remark 5.1. *While we restrict our attention to two key scheduler actions, AddTaskToNode and TerminateTask, in practice more actions may be available, such as TerminateNode, which offer additional degrees of freedom in scheduling policy design. An example of the available scheduler options is listed in the documentation of the Google Cluster Data [66].*

We assume instantaneous *node completion feedback* is available from each node, notifying the scheduler when it finishes executing the assigned task. With node completion feedback information, assuming we always terminate all copies of task i when the earliest copy of task i finishes, the performance of a scheduling policy is determined by the times for action AddTaskToNode only.

Remark 5.2. *Instantaneous node completion feedback is a reasonable approximation as in the regime that latency is of interest, the task execution time is usually much longer than the delay in obtaining feedback, such as the transmission delay in a data center.*

Disjoint and joint scheduling policies

We categorize scheduling policies into two categories, disjoint and joint policies. A disjoint scheduling policy schedules each task without taking the *execution dynamics* of other tasks, i.e., the start and finish times of other tasks, into account, and hence can determine the *possible* tasks starting times at time $t = 0$. We can express it as

$$\pi_{\mathrm{SL}} = \left[(i, t_{i,j}), i \in [n], t_{i,j} \in \mathbb{R}_+, j \in \mathbb{Z}^+\right],$$

where the policy launches the j-th copy of task i if task i is *not terminated by time* $t_{i,j}$[2].

By contrast, a joint scheduling policy takes the execution dynamics of other tasks into account, and hence may change the starting time vector during the execution process.

Remark 5.3. *While the disjoint scheduling policy takes less information into account and hence could be potentially less efficient, it allows better resource provisioning as we know the entire starting vector at $t = 0$. This may be of interest in certain cloud computing environments. Moreover, as we shall see later, understanding the disjoint scheduling policy is useful for analyzing joint scheduling policy as well.*

[2]Strictly speaking, a disjoint policy may have starting times that are random variables that do not depend on other tasks. In this thesis we choose to not consider this case.

■ 5.3.3 Performance metrics

Section 5.2 shows that task replication is useful in speeding up the computation of a collection of parallel tasks, which is rather intuitive. However, this is at the cost of adding more computation resources. Therefore, it is crucial to understand the trade-off between computation speed-up and additional usage of computation resources. For this purpose, we define two corresponding key performance measures, latency and resource usage, and analyze their trade-off in a variety of scenarios later, which provide insights on when and how replication is useful.

Latency is the time that at least one copy of every task in the job finishes running. More specifically, given n tasks, let the latency for task i be T_i, the latency of the computation be T, and $X_{i,j}, 1 \leq j \leq r_i + 1$ be the execution time of the j-th copy of task i if it is not terminated, where $X_{i,j} \overset{i.i.d.}{\sim} F_X$, then given a scheduling policy π that launches the j-th copy of task i at $t_{i,j}$, $1 \leq i \leq n, 1 \leq j \leq r_i + 1$, where $r_i + 1$ is the number of copies for task i, we have the following expression for latencies:

$$T_i(\pi) \triangleq \min_{1 \leq j \leq r_i+1} (t_{i,j} + X_{i,j}), \tag{5.2}$$

$$T(\pi) \triangleq \max_i T_i(\pi). \tag{5.3}$$

Note that in (5.2) we take the minimum as we only need to wait for the earliest copy of task i to finish, while in (5.3) we take the maximum as we need to wait for all tasks in the job to finish.

The modeling of resource usage depends on the application, and we propose two versions of the cost function for cloud computing and crowd sourcing respectively.

Cost for cloud users: For a user of a public cloud such as the Amazon Web Service (AWS), which charges the user by time and by node, we use the notion of *total running time*, which is the sum of the amount of running times for all computing nodes. More specifically, for the j-th node that runs task i, if the node starting time $t_{i,j} \leq T_i$, then it runs for $T_i - t_{i,j}$ seconds, otherwise it does not run at all. Hence, the running time for this node is $|T_i - t_{i,j}|^+$. Therefore,

$$C_{\text{cloud}}(\pi) \triangleq \sum_{i=1}^{n} \sum_{j=1}^{r_i+1} |T_i - t_{i,j}|^+ . \tag{5.4}$$

Cost for crowd sourcing In crowd sourcing, it is common that workers are paid a fixed amount of compensation for a given task. Therefore, the resource usage cost is proportional to the total number of computing nodes (workers), leading to the following cost:

$$C_{\text{crowd}}(\pi) \triangleq \sum_{i=1}^{n} \sum_{j=1}^{r_i+1} \mathbb{1}\{T_i < t_{i,j}\} . \tag{5.5}$$

For brevity, we may omit the π in $T(\pi)$ and $C(\pi)$ if it is clear from context.

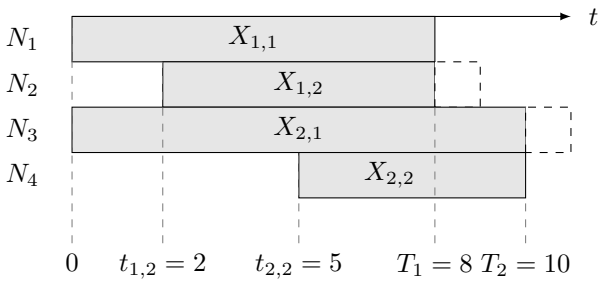


Figure 5-1: Example illustrating a scheduling policy and its performance measures. For the two given tasks, task 1 has latency 8 and task 2 has latency 10, leading to an overall latency of 10. The cloud user cost is 29, while the crowd sourcing cost is simply 4.

Fig. 5-1 shows an example illustrating a scheduling policy and its corresponding latency and costs. Given two tasks, we launch task 1 on nodes N_1 and N_2 at $t_{1,1} = 0$ and $t_{1,2} = 2$ respectively, and task 2 on nodes N_3 and N_4 at $t_{2,1} = 0$ and $t_{2,2} = 5$ respectively. The running time $X_{1,1} = 8$ and $X_{1,2} = 7$, and since node 1 finishes the task first at time $t = 8$, $T_1 = 8$ and node 2 is terminated before it finishes executing. Similarly, node 3 is terminated as node 4 finishes task 2 first at time $T_2 = 10$. The latency of this computation is $T = \max\{T_1, T_2\} = 10$. For a cloud user, the cost is the sum of all the nodes' actual running time, i.e., $C_{\text{cloud}} = 8 + 6 + 10 + 5 = 29$, while for a crowd sourcing user, the cost is simply the number of nodes that execute the task, i.e., $C_{\text{crowd}} = 4$.

Both of these performance measures are random variables, and in this thesis we choose to use their expected values as the performance metrics of interest, which correspond to the long-term average performance of scheduling policies.

■ 5.4 Motivating example

In this section we consider the following example, which shows in certain scenarios, task replication reduces both $\mathbb{E}[T]$ and $\mathbb{E}[C]$, even for a single task.

Let the execution time X satisfy

$$X = \begin{cases} 2 & \text{w.p. } 0.9 \\ 7 & \text{w.p. } 0.1 \end{cases}.$$

If we launch one task and wait for its completion, then the latency distribution is illustrated in Fig. 5-2a, and

$$T = 2 \times 0.9 + 7 \times 0.1 = 2.5, \tag{5.6}$$

$$C_{\text{cloud}} = T = 2.5. \tag{5.7}$$

If we launch a task at time $t_1 = 0$ and then launch a replicated task at time $t_2 = 2$ if the first one has not finished running by then, we have the latency distribution in Fig. 5-2b,

(a) P_T without replication

(b) P_T with replication at $t = 2$

Figure 5-2: Latency distribution under two different scheduling policies.

and in this case,

$$T = 2 \times 0.9 + 4 \times 0.09 + 7 \times 0.01 = 2.23,$$
$$C_{\text{cloud}} = 2 \times 0.9 + (4 + 2) \times 0.09 + (7 + 5) \times 0.01 = 2.46.$$

As we see here, introducing replication actually reduces both expected cloud computing cost and expected execution time!

■ 5.5 Two approaches for analyzing scheduling policies

With the example in Section 5.4 demonstrating the usefulness of replication, in this part of the thesis, we attempt to understand the trade-off between latency and resource usage of scheduling parallel tasks in more general scenarios. We conduct our analysis via the two different approaches, in Chapter 6 and Chapter 7 respectively.

In Chapter 6, we first investigate a specific class of policies called the *single-fork policy*, which include the `backup tasks` option in MapReduce as a special case. We conduct our investigation by assuming the execution time distribution are continuous, which allow us to leverage tools from order statistics and extreme value theory and obtain asymptotic characterizations of latency and resource usage. From these characterizations we obtain guidance on when to replicate and how much to replicate. Furthermore, we define another class of scheduling policies called the *multi-fork policy* in Section 6.4 that supports replication at multiple time instants. We show that the performance of a multi-fork policy can be analyzed by decomposing it to a scheduling with single-fork policies in multiple stages.

In Chapter 7, we investigate general scheduling policies by adopting discrete execution time distributions, and characterize the trade-off between latency and resource usage via a cost function that take both factors into account. Our analysis on the cost function reduces the search space for the optimal policy, and we also propose heuristic scheduling policies that have lower computational complexity.

Both of these approaches provide us with trade-offs between the latency and resource usage, leading to the following *scheduling policy design flow*:

1. estimate the execution time distribution from the traces of historical task execution times (e.g., Google Cluster Data [66]);

2. choose the one of the above analysis approaches to derive the trade-off between latency and resource usage;

3. depending on the latency or resource usage constraint, choose a proper point to operate on the trade-off curve

4. find the scheduling policy corresponding to this point.

For both approaches, we rely on the assumption that the execution time distribution is given, while in practice this needs to be estimated. Therefore, there is an issue in terms of the robustness of the scheduling policy, i.e., its performance when the execution time distribution is inaccurate. On this issue, we first note that there exists vast literature on the inference for distribution and order-statistics(e.g., [67–70], and we can leverage existing inference methods, especially robust ones (semi-parametric or non-parametric methods), to obtain the execution time distribution and the corresponding order statistics.

Besides the plug-in approach of estimate-then-schedule, an interesting research direction is to develop adaptive scheduling algorithms that directly works with unknown execution time distribution. This shares some similarity to the celebrated multi-arm bandit problem, as there is a trade-off between the exploration of execution time distribution and the exploitation of this execution time distribution.

Chapter 6

Design and analysis of forking policies

In this section we investigate two classes of scheduling policies, the *single-fork* policy and the *multi-fork policy*. Both classes of policies replicate all unfinished tasks at certain time instants, and we call these two classes of policies *forking policies*. These policies include certain practical implementations, such as the `backup task` option in Google MapReduce, as special cases. In our analysis, we model execution times of parallel tasks as continuous random variables because this enables us to leverage tools from order statistics, especially extreme value theory, to derive the trade-off between latency and resource usage analytically.

The rest of the chapter is organized as follows. We first define the single-fork policy in Section 6.1 and express its performance measures in terms of order statistics. Then we introduce some results from order statistics Section 6.2 that help to analyze these order statistics. With these results, we analyze single-fork policies in Section 6.3. Finally, we show that our analysis of single-fork policy can essentially be applied to its extension, the multi-fork policy in Section 6.4.

6.1 Single-fork policy and its performance measures

In this section we define the single-fork policy mathematically and express its performance measures in terms of order statistics.

Definition 6.1 (Single-fork scheduling policy). *Given n tasks with i.i.d. execution time distribution F_X, a* single-fork scheduling policy $\pi_{\mathrm{SF}}(p, r, l; n, F_X)$ *launches all n task at time $t = 0$ via the* AddTaskToNode *operation, wait until there are pn unfinished tasks, and choose one of the following two actions:*

replicate without relaunching: *launch r new copies for all unfinished tasks;*

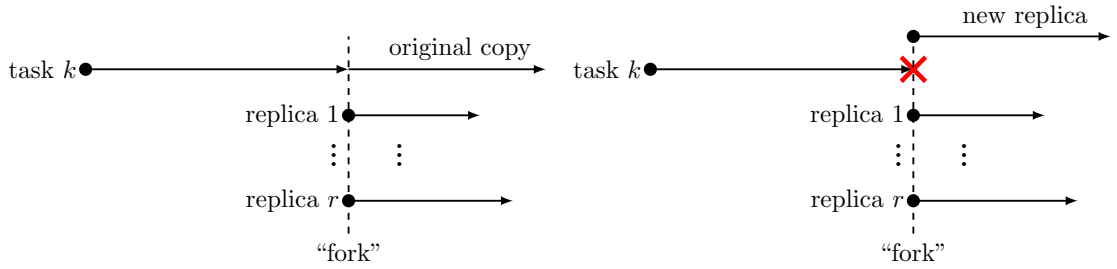
replicate with relaunching: *terminate the current copy of all unfinished tasks and launch $r + 1$ new copies for each of these tasks.*

Then for each task, the result from the copy that finishes earliest is used and all the other copies are terminated via the TerminateTask *operation.*

We use $l = 0$ to denote the case of relaunching and $l = 1$ the case of no relaunching. We omit F_X in $\pi_{\mathrm{SF}}(p, r, l; n, F_X)$ for brevity when it is clear from the context.

As we shall see, relaunch or not impacts the performance of a scheduling policy, and these two actions are illustrated in Fig. 6-1.

We note that $p = 0$ corresponds to the case of simply running n tasks in parallel, while $r = 0$ does not necessarily correspond to that because with $l = 0$ we relaunch pn tasks. We also note that a single-fork policy is a joint policy (cf. Section 5.3.2 for definition).



(a) Single fork with no relaunching (b) Single fork with relaunching

Figure 6-1: Illustration of single-fork policies with and without relaunching.

Example 6.1 (Backup tasks in MapReduce). *Our single-fork policy captures the* `backup tasks` *option in the Google MapReduce framework [36]. With this option, MapReduce initially distributes a collection of tasks to available computing nodes, and later re-executes the last few of the unfinished tasks to reduce the computation delay due to "stragglers" (computers that run unusually slow). Empirically, this scheduling policy works well. For example, it reduces the latency by* $1/3$ *in distributed sort [36]. Following our notation,* $r = 1, p = 0.1$ *and* $l = 1$ *is used in [36].*

Below we analyze the latency and resource usage for the single-fork policy, and show that both quantities can be represented as functions of various order statistics. We defer all proofs in this section to Appendix B.2.

The latency of a single fork policy $\pi_{\mathrm{SF}}(p, r, l; n)$ can be decomposed into two parts:

- $T^{(1)}(\pi_{\mathrm{SF}})$: the time to execute the first $\bar{p}n$ tasks,
- $T^{(2)}(\pi_{\mathrm{SF}})$: the time to execute the rest of the pn tasks with replication.

It is straightforward that

$$T^{(1)}(\pi_{\mathrm{SF}}) = X_{\bar{p}n:n}.$$

At time $T^{(1)}$, there are pn unique tasks remaining, and we denote their remained running time after replicating each of them r times by $Y_j, 1 \le j \le pn$. The analysis of these new running times $\{Y_j\}$ can be facilitated via a concept called conditional excess distribution.

Definition 6.2 (conditional excess distribution). *Given* $X \sim F_X$ *and* t*, the* conditional excess distribution $F_{\Delta_{X,t}}(x)$ *is defined via its complement* $\bar{F}_{\Delta_{X,t}}(x)$*, i.e.,*

$$\bar{F}_{\Delta_{X,t}}(x) = \frac{\bar{F}_X(t+x)}{\bar{F}_X(t)}, \quad 0 \le x \le \omega(F_X) - t. \tag{6.1}$$

For each of the remaining task j, the additional running time for the original copy is Δ_{X_j,T^*}, where $T^* = l \cdot T^{(1)}$. For the newly launched r copies, it takes $X'_{1:r}$ for the earliest copy to finish, where $X'_i \overset{i.i.d.}{\sim} F_X, 1 \le i \le r$. Therefore, Y_j is the minimum of two random variables, Δ_{X_j,T^*} and $X'_{1:r}$. We summarize these results in the following lemma.

Lemma 6.1. *All* $Y_j \overset{i.i.d.}{\sim} F_Y$*, and* $\bar{F}_Y(y)$ *is a function* $g(F_X, r, l)$ *that satisfies*

$$\bar{F}_Y(y) = g(F_X, r, l) \triangleq \bar{F}_{\Delta_{X,T^*}}(y) \left(\bar{F}_X(y)\right)^r, \tag{6.2}$$

where $T^* = l \cdot T^{(1)}$ *and the expression for* $\bar{F}_{\Delta_{X,T^*}}(y)$ *is given in (6.1).*

Noting the latency for $\pi_{\mathrm{SF}}(p, r, l; n)$ satisfies

$$\begin{aligned} T(\pi_{\mathrm{SF}}) &= T^{(1)}(\pi_{\mathrm{SF}}) + T^{(2)}(\pi_{\mathrm{SF}}) \\ &= X_{\bar{p}n:n} + Y_{pn:pn}, \end{aligned}$$

we can have the expected latency expression in Theorem 6.2.

Theorem 6.2 (Expected latency). *The latency of a single fork policy* $\pi_{\mathrm{SF}}(p, r, l; n, F_X)$ *satisfies*

$$\mathbb{E}[T(\pi_{\mathrm{SF}})] = \mathbb{E}[X_{\bar{p}n:n}] + \mathbb{E}[Y_{pn:pn}], \tag{6.3}$$

where Y *is specified in (6.2).*

Similarly, the cost of a single fork policy $\pi_{\mathrm{SF}}(p, r, l; n)$ can be decomposed into two parts:

- $C^{(1)}(\pi_{\mathrm{SF}})$: the cost to execute the first $\bar{p}n$ tasks,
- $C^{(2)}(\pi_{\mathrm{SF}})$: the cost to execute the rest of the pn tasks with replication.

This leads to the expected cost expression in Theorem 6.3.

Theorem 6.3 (Expected cost). *The cost of a single fork policy* $\pi_{\mathrm{SF}}(p, r, l; n)$ *satisfies*

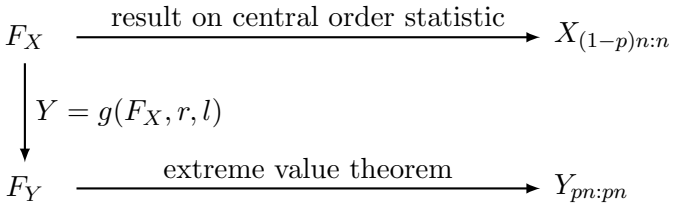
$$\mathbb{E}[C_{\mathrm{cloud}}(\pi_{\mathrm{SF}})] = \sum_{i=1}^{\bar{p}n} \mathbb{E}[X_{i:n}] + pn\mathbb{E}[X_{\bar{p}n:n}] + (r+1)(pn)\mathbb{E}[Y], \tag{6.4}$$

$$C_{\mathrm{crowd}}(\pi_{\mathrm{SF}}) = n + npr. \tag{6.5}$$

Setting $p = 0$ we obtain the performance of no task replications:

$$\begin{aligned} T &= X_{n:n}, \\ C_{\mathrm{cloud}} &= \sum_{i=1}^{n} X_{i:n} = \sum_{i=1}^{n} X_i, \\ C_{\mathrm{crowd}} &= n. \end{aligned}$$

Now we proceed to Section 6.2, where we introduce results in order statistics that are useful in analyzing (6.3) and (6.4). In particular, we will introduce results on central order statistics and the *extreme value theorem*, which help to analyze the two terms in (6.3), as shown below.



6.2 Order statistics: definitions and results

In this section we introduce some definitions in order statistics and some results that play important roles in the performance analysis of scheduling policies later in Section 6.3. In particular, we show that order statistics in different regimes have different concentration behavior in Sections 6.2.1 to 6.2.3 respectively.

The notation for order statistics is first introduced in Section 1.1 on Page 18.

6.2.1 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 distributed with the p-th quantile of X as its mean, as indicated by the following result.

Theorem 6.4 (Theorem 10.3 in [68]). *Given* $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F$, *if* $0 < p < 1$ *and* $0 < f(x_p) < \infty$, *where* $x_p = F^{-1}(p)$, *then for* $k = k(n)$ *such that* $k = np + o(\sqrt{n})$,

$$X_{k:n} \xrightarrow{P} \mathcal{N}\left(x_p, \frac{p(1-p)}{nf^2(x_p)}\right)$$

where $f(\cdot)$ *is the p.d.f. corresponds to* F *and* $\xrightarrow{P}$ *denotes convergence in probability as* $n \to \infty$.

Therefore, when n is large, $X_{k:n}$ is tightly concentrated around x_p.

6.2.2 Extreme order statistics

Extreme value theory (EVT) is an asymptotic theory regarding the sample extremes, i.e., minima and maxima. It shows that if a distribution belongs to a certain class (domain of attraction in Theorem 6.6), then its maxima can be well characterized asymptotically. More specifically, given n i.i.d. random variables $X_1, X_2, \ldots, X_n$, if a non-degenerate limit distribution of the sample extreme $X_{n:n}$ exists after proper shifting and scaling, then the distribution of this sample extreme belongs to a class of distributions G_γ, which is called *extreme value distributions*.

Theorem 6.5 (Fisher-Tippett-Gnedenko theorem, Theorem 1.1.3 in [71]). *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*

$$\mathbb{P}\left[(X_{n:n} - b_n)/a_n \leq x\right] \to G_\gamma(x) \tag{6.6}$$

as $n \to \infty$ *and* $G_\gamma(\cdot)$ *is a non-degenerate distribution, then*

$$G_\gamma(x) = \exp\left(-(1+\gamma x)^{-1/\gamma}\right), 1+\gamma x > 0,$$

with γ *real and where for* $\gamma = 0$ *the right-hand side is interpreted as* $\exp(-e^{-x})$.

If a distribution function satisfies (6.6) with G_γ for some $\gamma \in \mathbb{R}$, we say the distribution function F is in the *domain of attraction* of G_γ, and denote it as $F \in \mathrm{DA}(G_\gamma)$.

The behavior of $G_\gamma(\cdot)$ depends primarily on the sign of γ. Theorem 6.6 specifies the necessary and sufficient condition for a distribution function F to be in the domain of attraction that corresponds to γ in a certain range. Furthermore, Corollary 6.7 provides the normalization coefficients and the resulting limiting distribution.

Theorem 6.6 (Domain of attractions, Theorem 1.2.1 in [71]). *A distribution function* F *is in the domain of attraction of the extreme value distribution* G_γ *if and only if*

1. *For* $\gamma = 0$*: 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. *For* $\gamma > 0$*:* $\omega(F) = \infty$ *and*

$$\lim_{x \to \infty} \frac{\bar{F}(tx)}{\bar{F}(x)} = t^{-1/\gamma}, \quad t > 0;$$

3. *For* $\gamma < 0$*:* $\omega(F) < \infty$ *and*

$$\lim_{x \to 0^+} \frac{\bar{F}(\omega(F) - tx)}{\bar{F}(\omega(F) - x)} = t^{-1/\gamma}, \quad t > 0;$$

where $\omega(\cdot)$ *is defined in (5.1).*

Corollary 6.7 (Corollary 1.2.4 in [71]). *The constants* a_n*,* b_n*, and the distribution function* $G_\gamma(\cdot)$ *in (6.6) satisfy*

1. *For* $\gamma = 0$*,*

$$a_n = \eta\left(F^{-1}(1-1/n)\right),$$
$$b_n = F^{-1}(1-1/n),$$

and $G_\gamma(x)$ *with* $\gamma = 0$ *is called the* Gumbel distribution*,*

$$\Lambda(x) = \exp\{-\exp(-x)\}. \tag{6.7}$$

2. *For* $\gamma > 0$,

$$\begin{aligned} a_n &= F^{-1}(1 - 1/n), \\ b_n &= 0, \end{aligned}$$

and $G_\gamma(x)$ *with* $\gamma > 0$ *is called the* Fréchet distribution *with* $\alpha = 1/\gamma > 0$,

$$\Phi_\alpha(x) = \begin{cases} 0 & x \leq 0 \\ \exp\left\{-x^{-\alpha}\right\} & x > 0 \end{cases}. \tag{6.8}$$

3. *For* $\gamma < 0$,

$$\begin{aligned} a_n &= \omega\left(F\right) - F^{-1}(1 - 1/n), \\ b_n &= \omega\left(F\right), \end{aligned}$$

and $G_\gamma(x)$ *with* $\gamma < 0$ *is called the* reversed-Weibull distribution *with* $\alpha = -1/\gamma > 0$,

$$\Psi_\alpha(x) = \begin{cases} \exp\left\{-\left(-x\right)^{\alpha}\right\} & x < 0, \\ 1 & x \geq 0. \end{cases} \tag{6.9}$$

Remark 6.1. *We called the distribution in (6.9) the* reversed Weibull *distribution as the ordinary Weibull distribution arises in reliability applications has distribution function*

$$F(y; \alpha) = \begin{cases} 1 - e^{-y^\alpha} & y \geq 0 \\ 0 & y < 0 \end{cases},$$

which indicates a strictly positive support*.*

We say a distribution function $F \in \mathrm{DA}\left(\Lambda\right)$ if $F \in \mathrm{DA}\left(G_\gamma\right)$ with $\gamma = 0$ in Theorem 6.6, and $F \in \mathrm{DA}\left(\Phi_\alpha\right)$ for $\gamma = 1/\alpha > 0$ and $F \in \mathrm{DA}\left(\Psi_\alpha\right)$ for $\gamma = -1/\alpha < 0$. As we have seen, being in a different domain of attraction leads to different asymptotic behavior for the extreme values.

Remark 6.2. *Intuitively,* $F \in \mathrm{DA}\left(\Lambda\right)$ *corresponds to the case that* $\bar{F}$ *has an exponentially decaying tail,* $F \in \mathrm{DA}\left(\Phi_\alpha\right)$ *corresponds to the case that* $\bar{F}$ *has heavy tail (such as polynomially decaying), and* $F \in \mathrm{DA}\left(\Psi_\alpha\right)$ *corresponds to the case that* $\bar{F}$ *has a short tail with finite upper bound.*

We can characterize the limit distribution of the sample extreme $X_{1:n}$ analogously via Theorem 6.6 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.

■ 6.2.3 Intermediate order statistics

When $k \to \infty$ and $n - k \to \infty$, such that k/n approaches 0 or 1, $X_{k:n}$ is an *intermediate order statistic* and its asymptotic result falls somewhere between the extreme and central order cases, and one of the characterizations is provided in Theorem 6.8.

Theorem 6.8 (Theorem 10.8.1 in [68]). *Given* $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F$, *if* F *is in* $\mathrm{DA}(\Lambda), \mathrm{DA}(\Phi_\alpha)$ *or* $\mathrm{DA}(\Psi_\alpha)$, *then as* $k \to \infty$ *and* $k/n \to 0$, *with* $p_n = 1 - k/n$, *then let* $x_{p_n} = F^{-1}(p_n)$,

$$X_{n-k+1:n} \xrightarrow{d} \mathcal{N}\left(x_{p_n}, \frac{k}{nf^2(x_{p_n})}\right),$$

where $f(\cdot)$ *is the p.d.f. corresponds to* F *and* $\xrightarrow{d}$ *denotes convergence in distribution as* $n \to \infty$.

■ 6.3 Single-fork policy analysis

With the setup in Section 6.1 and the results in Section 6.2, in this section we analyze the asymptotic characterizations of these performance metrics in Section 6.3.1, and finally apply these characterizations to two canonical execution time distributions, Pareto and shifted exponential, to draw insights for system design in Section 6.3.2. We defer all proofs in this section to Appendix B.2.

■ 6.3.1 Performance characterization

In this section we apply results presented in Section 6.2 to calculate the asymptotic expected latency and resource usage expressions in Theorem 6.2 and Theorem 6.3.

We introduce an assumption that greatly simplifies our analysis.

Simplifying assumption: From now on we assume $T^{(1)} = x_{\bar{p}}$, because by Theorem 6.4 and Theorem 6.8, for any $0 < p < 1$, $T^{(1)}$ converges to $x_{\bar{p}}$ quickly as $n \to \infty$. As we shall see later from the simulation results, the inaccuracy introduced by this assumption is negligible. As a result, we denote $T^* = l \cdot x_{\bar{p}}$.

With this assumption, Lemma 6.9 below indicates F_Y introduced in Lemma 6.1 is in the same domain of attraction as F_X, as indicated in Lemma 6.9.

Lemma 6.9 (Domain of attraction for F_Y). *Given a single fork policy* $\pi_{\mathrm{SF}}(p, r, l; n)$ *with* $0 < p < 1$,

1. *if* $F_X \in \mathrm{DA}(\Lambda)$, *then* $F_Y \in \mathrm{DA}(\Lambda)$;
2. *if* $F_X \in \mathrm{DA}(\Phi_\alpha)$, *then* $F_Y \in \mathrm{DA}\left(\Phi_{(r+1)\alpha}\right)$;
3. *if* $F_X \in \mathrm{DA}(\Psi_\alpha)$, *then* $F_Y \in \mathrm{DA}\left(\Psi_{(lr+1)\alpha}\right)$.

Knowing the domain of attraction for F_Y, we can apply Theorem 6.5 to analyze the latency of the second stage $Y_{pn:pn}$, and obtain Theorem 6.10 on its expected value.

Theorem 6.10. *When $n \to \infty$,*

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

where $\tilde{a}_{pn}$ and $\tilde{b}_{pn}$ are the normalizing constants of F_Y for its domain of attraction (cf. Corollary 6.7), γ_{EM} is the Euler-Mascheroni constant, and $\Gamma(\cdot)$ is the Gamma function, i.e.,

$$\Gamma(t) \triangleq \int_0^\infty x^{t-1}e^{-x}\,dx. \tag{6.10}$$

The coefficients $\tilde{a}_{pn}$ and $\tilde{b}_{pn}$ for F_Y can be evaluated numerically or sometimes analytically via Corollary 6.7. Therefore, we can calculate (6.2), which allows us to see how the expected latency varies as we change the scheduling parameters such as p, r and l, a topic we investigate in the next section.

■ 6.3.2 Scheduling examples

In this section we calculate the trade-off of latency and resource usage for two canonical execution time distributions, Pareto distribution and Shifted Exponential distribution (cf. (6.13) for definition). The former has a heavy (polynomially decaying) tail while the latter an exponentially decaying tail. We focus on the cloud user cost defined in (5.4) as the calculation of crowd sourcing cost is straightforward.

For both execution time distributions, we analyze the cases of relaunching and no relaunching, and show that for the Pareto distribution, relaunching leads to smaller latency in certain regime, while for the Shifted Exponential distribution, no relaunching always achieves smaller latency.

Pareto execution time

A Pareto distribution $\mathsf{Pareto}\left(\alpha, x_m\right)$ satisfies

$$F(x;\alpha,x_m) \triangleq \begin{cases} 1 - \left(\frac{x_m}{x}\right)^\alpha & x \geq x_m, \\ 0 & x < x_m. \end{cases} \tag{6.11}$$

Pareto distribution is a heavy-tail distribution, which corresponds well to both the task size and task execution time distribution in data centers [43, 72].

First from (6.11) we obtain

$$x_{\bar{p}} = F_X^{-1}\left(1 - p;\alpha,x_m\right) = x_m p^{-1/\alpha},$$

Regarding $T^{(2)}$, when there is relaunching, by Lemma B.3, $Y \sim \mathsf{Pareto}\left((r+1)\alpha, x_m\right)$, and

$$\mathbb{E}\left[Y_{pn:pn}\right] = \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right)(pn)^{\frac{1}{(r+1)\alpha}}x_m.$$

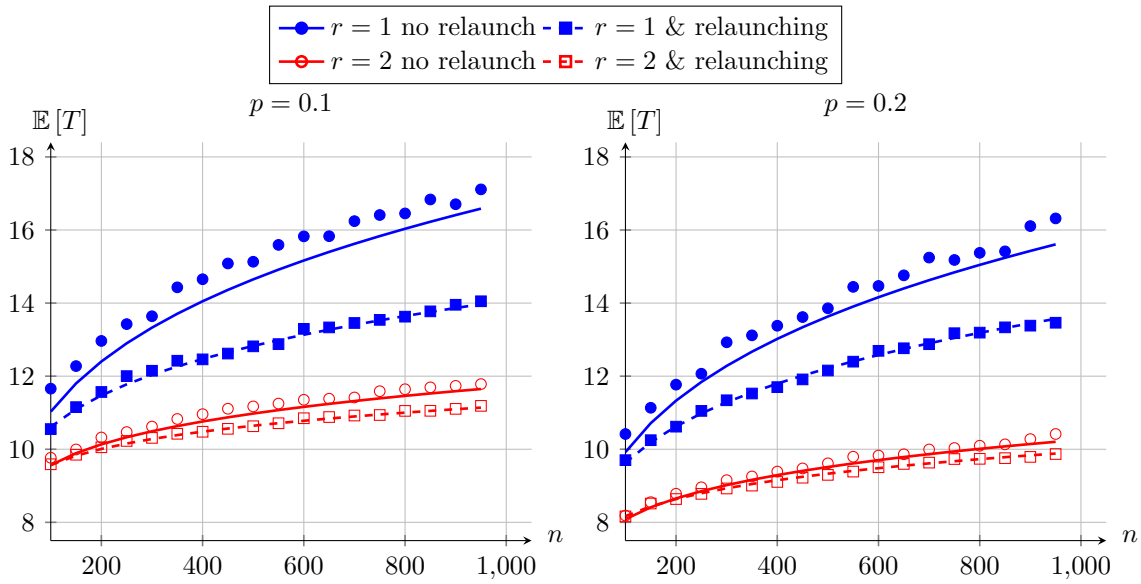


Figure 6-2: Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the Pareto distribution $\mathsf{Pareto}(2, 2)$.

When there is no relaunching, $Y = \min\{\mathsf{Pareto}(\alpha, x_{\bar{p}}) - x_{\bar{p}}, \mathsf{Pareto}(r\alpha, x_m)\}$, which leads to

$$\mathbb{E}[Y_{pn:pn}] = \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right)\tilde{a}_{pn},$$

and derivations in Appendix B.3.1 shows that $\tilde{a}_n$ satisfies

$$n^{1/\alpha} = \frac{1}{x_m^r x_{\bar{p}}}(x_{\bar{p}}\tilde{a}_n^r + \tilde{a}_n^{r+1}). \tag{6.12}$$

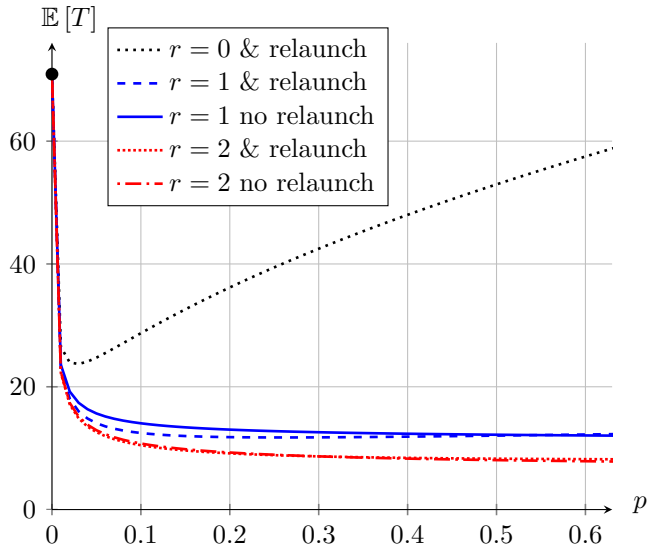
Fig. 6-2 compares the latency obtained from simulation and analytical calculations for Pareto distribution, indicating latency obtained from analytical calculation is very close to the actual performance for $n \geq 100$, especially for the case with relaunching ($l = 0$).

Remark 6.3 (Job size in data centers). *Analysis of real-world trace data shows that it is common for a job to contain hundreds or even thousands of tasks [72].*

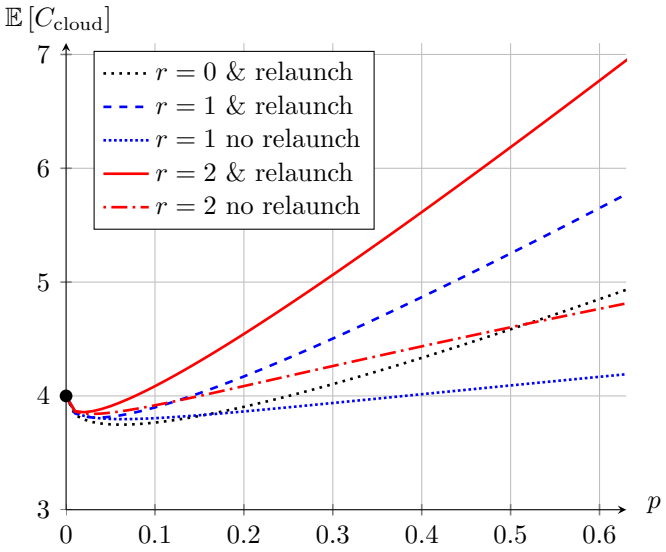
With the cloud computing cost in Appendix B.3.1, we plot the latency and cloud user cost as p changes in Fig. 6-3, leading the following observation.

Observation 6.11. *A small amount of replication can reduce latency significantly, and may lead to lower cloud user cost than the case of no replication.*

From Fig. 6-4 we draw the following two observations.



(a) Expected latency $\mathbb{E}[T]$.



(b) Expected cloud user cost $\mathbb{E}[C_{\text{cloud}}]$.

Figure 6-3: Expected latency and cloud user cost for a Pareto execution time distribution $\mathsf{Pareto}(2, 2)$, given $n = 400$.

Observation 6.12. *For a given n, when p is small enough, relaunching leads to lower latency than no relaunching.*

Observation 6.13. *With relaunching, earlier replication (larger p) may not reduce latency, while without relaunching, earlier replication always reduces latency.*

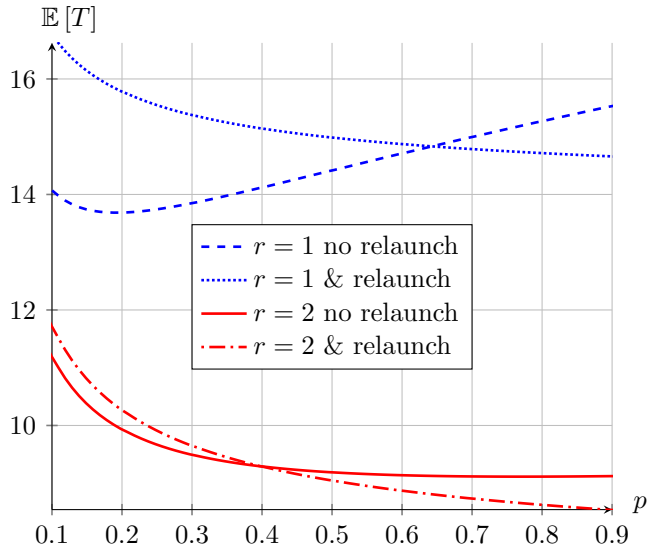


Figure 6-4: The expected latency for relaunching and no relaunching ($n = 1000$) for the Pareto distribution $\mathsf{Pareto}(2, 2)$.

Relaunching too early could increase latency, because in this case it may terminate tasks that are run by "normal" nodes, which betrays the purpose of terminating a "straggler" and launching the task on another normal node.

Finally, in the cloud computing scenario, we can characterize the trade-off between latency and cost shown in Fig. 6-5, leading the observations below.

Observation 6.14. *In general, no relaunching achieves better trade-off between latency and cloud user cost due to the huge cloud user cost overhead in relaunching.*

Observation 6.15. *The optimal different replication level r depends on the latency requirement.*

For example, to achieve latency of 8, $r = 3$ with no relaunching results in the lowest cloud user cost, while to achieve latency 10, $r = 2$ is more preferable in terms of cloud user cost.

Shifted Exponential execution time

While the exponential distribution has been popular in various latency analysis, especially queuing theory, it may not be suitable for modeling the execution time of a task, as a task seldom finishes instantaneously, and usually it is lower bounded by a constant delay due to machine start-up or task initialization. Therefore, we model the execution time of tasks as the sum of a constant x_m and an exponential random variable with parameter λ, and we called this distribution a *Shifted Exponential* distribution $\mathsf{SExp}(x_m, \lambda)$, which has c.d.f.

$$F(x) = 1 - e^{-\lambda(x - x_m)}, x \geq x_m. \tag{6.13}$$

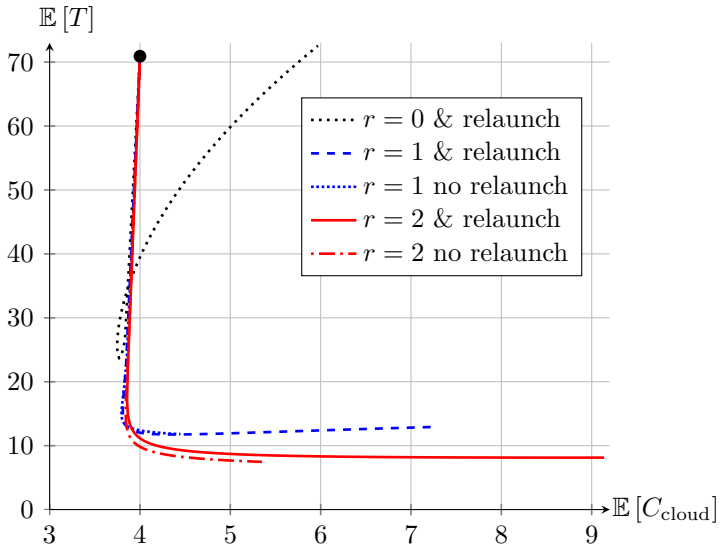


Figure 6-5: The trade-off between expected latency $\mathbb{E}[T]$ and normalized expected cloud user cost $\mathbb{E}[C_{\text{cloud}}]/n$ for $\mathsf{Pareto}(2,2)$ and $n = 400$, by varying p in the range of $[0,1]$.

The Exponential distribution $\mathsf{Exp}(\lambda) \sim \mathsf{SExp}(0,\lambda)$.

Let $E_1, E_2, \ldots, E_n \overset{i.i.d.}{\sim} \mathsf{Exp}(\lambda)$, it is not hard to see that

$$X_{k:n} = x_m + E_{k:n}.$$

From (6.13), we derive

$$x_{\bar{p}} = x_m + \frac{\ln(1/p)}{\lambda}. \tag{6.14}$$

When there is relaunching, by Lemma B.1, $Y = X_{1:r+1} = x_m + \mathsf{Exp}((r+1)\lambda)$, and

$$\mathbb{E}[Y] = x_m + \frac{1}{(r+1)\lambda} \tag{6.15}$$

$$\mathbb{E}[Y_{pn:pn}] = x_m + \frac{\ln(pn) + \gamma_{\text{EM}}}{(r+1)\lambda} \tag{6.16}$$

When there is no relaunching, $Y = \min\{\mathsf{Exp}(\lambda), x_m + \mathsf{Exp}(r\lambda)\}$, and the calculations in Appendix B.3.2 show that

$$\mathbb{E}[Y] = \frac{1}{\lambda}\left(1 - e^{-\lambda x_m}\right) + \frac{1}{(r+1)\lambda} e^{-\lambda r x_m} \tag{6.17}$$

$$\mathbb{E}[Y_{pn:pn}] = \frac{r}{r+1} x_m + \frac{\ln(pn) + \gamma_{\text{EM}}}{\lambda(r+1)}. \tag{6.18}$$

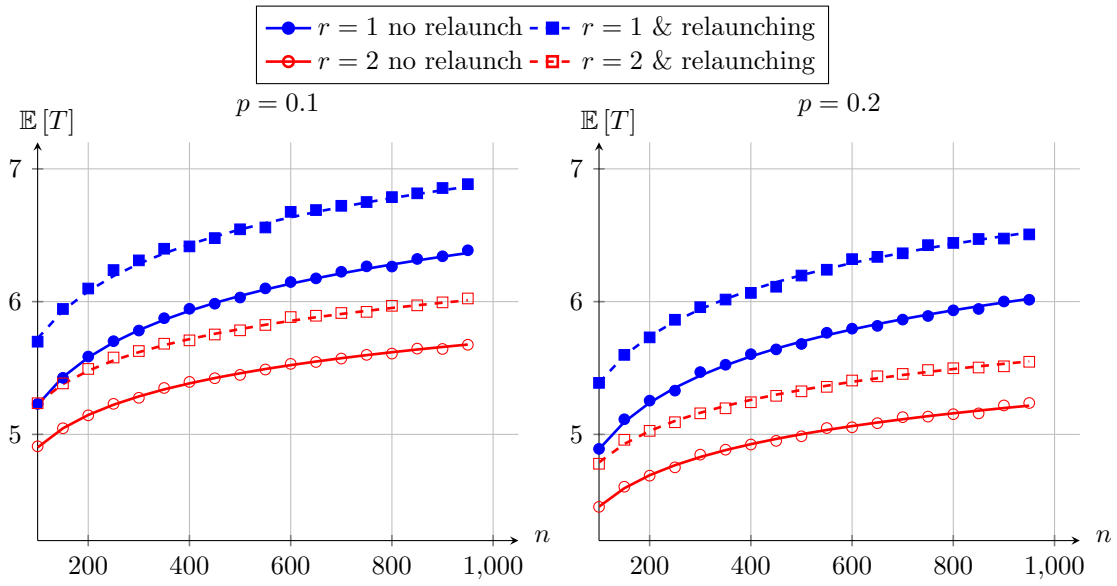


Figure 6-6: Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the Shifted Exponential distribution $\mathsf{SExp}(1, 1)$.

Fig. 6-6 compares the latency obtained from simulation and analytical calculations for the Shifted Exponential distribution, which again demonstrates the effectiveness of the asymptotic theory.

For the Shifted Exponential execution time distribution, we draw the following observations.

Observation 6.16. *Given the same p and r, relaunching leads to larger latency than no relaunching.*

Observation 6.16 follows directly after comparing (6.16) and (6.18).

Observation 6.17. *Replicating earlier (larger p) leads to smaller latency, relaunch or not.*

Following (6.14), (6.16) and (6.18),

$$\begin{aligned}\mathbb{E}[T] &= \mathbb{E}\left[T^{(1)}\right] + \mathbb{E}\left[T^{(2)}\right] \\ &= \frac{2r+l}{r+l} x_m + \frac{1}{(r+1)\lambda}\left(\ln n - r\ln p + \gamma_{\text{EM}}\right),\end{aligned}$$

indicating the larger p, the smaller the latency.

Observation 6.18. *Given r, replicating later (smaller p) reduces cloud user cost.*

The derivations in Appendix B.3.2 show that

$$\lambda\mathbb{E}[C_{\text{cloud}}] = \begin{cases} n + pn\left[\lambda x_m + r\left(1 - e^{-\lambda x_m}\right)\right] & l = 1 \\ n + pn\lambda(r+2)x_m & l = 0 \end{cases}.$$

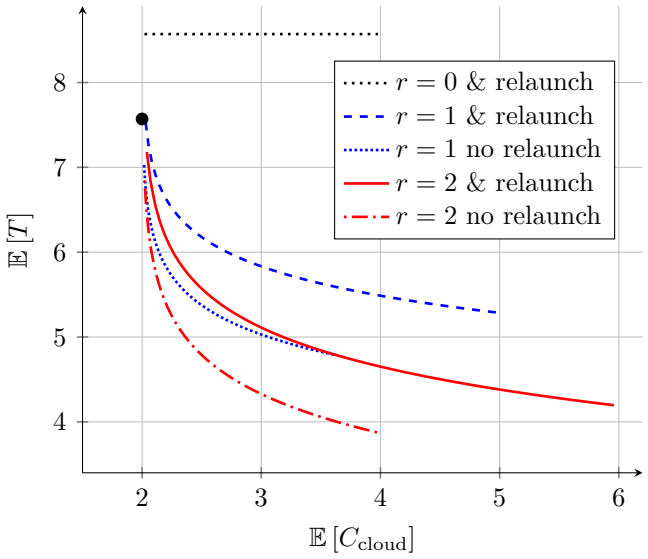


Figure 6-7: The trade-off between expected latency $\mathbb{E}[T]$ and normalized expected cloud user cost $\mathbb{E}[C_{\text{cloud}}]/n$ for $\mathsf{SExp}(1,1)$ and $n = 400$, by varying p in the range of $[0.05, 0.95]$.

Therefore, when $x_m > 0$, using small p, which corresponds to less replication reduces cloud user cost. However, when $x_m = 0$, $\lambda\mathbb{E}[C_{\text{cloud}}] = n$, which is independent of p and r.

Remark 6.4. *When $x_m = 0$, we can see that replicating jobs does not increase cloud user cost, but reduces latency, which suggests we should replicate as many copies as possible. This counter-intuitive result suggests that the Exponential distribution is indeed a poor model of the execution time distribution.*

Observation 6.19. *If $x_m > \beta^*/\lambda$, where $\beta^* > 0$ is the solution to*

$$\beta r + \beta - r + re^{-\beta} = 0,$$

then relaunching leads to strictly larger latency and cloud computing cost than no relaunching.

In particular, $\beta^* < 1$, hence if $x_m \geq 1/\lambda$, then no relaunching achieves better trade-off between latency and cost than relaunching.

The detailed derivations for Observation 6.19 is shown in Appendix B.3.2, and Fig. 6-7 illustrates the trade-off for execution time distribution $\mathsf{SExp}(1,1)$ at $n = 400$.

■ 6.4 Multi-fork policy analysis

In Section 6.3, the single-fork policy replicate tasks at one time instant to speed up the computation, i.e., fork once. In this section, we consider the policy that forks multiple times, i.e., the multi-fork policy.

Definition 6.3 (Multi-fork scheduling policy). *Given the execute time distribution* F_X, $\mathbf{p} = [p_1, p_2, \ldots, p_k]$, $\mathbf{r} = [r_1, r_2, \ldots, r_k]$, *and* $\mathbf{l} = [l_1, l_2, \ldots, l_k]$, *where* $p_1 > p_2 > \ldots p_k$, *a* multi-fork scheduling policy $\pi_{\mathrm{MF}}(n, \mathbf{p}, \mathbf{r}, \mathbf{l}; F_X)$ *replicates each the remaining tasks* r_i *times when there are* np_i *unique tasks still unfinished, and takes the result from the earliest finished copy.*

Similar to the single-fork policy, when replicating an unfinished task, the scheduler can choose to either terminate the current copy of unfinished task and relaunch a new copy (relaunch, $l_i = 0$*), or let the current copy continue to run (no relaunch,* $l_i = 1$*).*

Our key result is that a multi-fork policy can be decomposed as multiple single-fork policies, leading to the performance characterization in Theorem 6.20.

Theorem 6.20 (Multi-fork policy performance). *The latency and cost of a multi-fork scheduling policy* $\pi_{\mathrm{MF}}(n, \mathbf{p}, \mathbf{r}, \mathbf{l}; F_X)$ *satisfy*

$$\begin{aligned} T\left(\pi_{\mathrm{MF}}\left(n, \mathbf{p}, \mathbf{r}, \mathbf{l}; F_X\right)\right) &= \sum_{i=1}^{k} T^{(1)}\left(\pi_{\mathrm{SF}}\left(nq_{i-1}, q_i, r_i, l_i; F_{X^{(i)}}\right)\right) \\ &\quad + T^{(2)}\left(\pi_{\mathrm{SF}}\left(nq_k, q_k, r_k, l_k; F_{X^{(k)}}\right)\right) \\ C\left(\pi_{\mathrm{MF}}\left(n, \mathbf{p}, \mathbf{r}, \mathbf{l}; F_X\right)\right) &= \sum_{i=1}^{k} C^{(1)}\left(\pi_{\mathrm{SF}}\left(nq_{i-1}, q_i, r_i, l_i; F_{X^{(i)}}\right)\right) \\ &\quad + C^{(2)}\left(\pi_{\mathrm{SF}}\left(nq_k, q_k, r_k, l_k; F_{X^{(k)}}\right)\right) \end{aligned}$$

where $T^{(1)}(\cdot)$, $T^{(2)}(\cdot)$, $C^{(1)}(\cdot)$ *and* $C^{(2)}(\cdot)$ *are introduced in Section 6.1,*

$$F_{X^{(i)}} = \begin{cases} F_X & i = 1 \\ g\left(F_{X^{(i-1)}}, r_i, l_i\right) & 2 \le i \le k \end{cases},$$

$$q_i = \begin{cases} 1 & i = 0 \\ p_i / q_{i-1} & 1 \le i \le k \end{cases}.$$

Proof. Given a multi-fork scheduling policy $\pi_{\mathrm{MF}}(n, \mathbf{p}, \mathbf{r}, \mathbf{l}; F_X)$, after the first fork, we can view the remaining problem as a new scheduling problem with np_1 unique tasks, each with execution time distribution $F_{X^{(2)}} = g(F_X, r_1, l_1)$. Applying this recursively after each fork completes the proof. □

We leave the topic of using Theorem 6.20 to design multi-fork policies that achieve better trade-off between latency and resource usage for future exploration.

■ 6.5 Concluding remarks

In this chapter we characterize the performance trade-offs of single-fork and multi-fork polices, which allow system designers or users to follow the procedure outlined in Section 5.5 to obtain scheduling policies.

The accuracy of our performance characterizations depends crucially on the accuracy of the execution time distribution F_X, especially its tail behavior, because the latency

of the second stage of a single-fork policy largely depends on the tail behavior of the execution time distribution. Therefore, for estimation purpose, it is important to capture the anomalies in task duration data.

In practice, we may have more knowledge about the states of different computing nodes and hence may model the execution time of the same task on different computing node differently. For example, faster computing nodes may correspond to execution time distribution with a smaller mean. While this makes the analysis more involved, as long as the execution time for the same task are independent among different machines, replication continues to help and our framework remains useful.

Finally, we note that while a single-fork policy introduced in this chapter is a joint scheduling policy (cf.Section 5.3.2 for definition), it is possible to obtain a corresponding disjoint policy by setting the replication time of tasks to $x_{\bar{p}}$[1]. By the "simplifying assumption" discussion in Section 6.3.1, the performance trade-off obtained from this disjoint policy is essentially the same as its joint counterpart.

[1]We only replicate if a task if not finished by then.

Chapter 7

Design and analysis of general scheduling policies

The investigation in Chapter 6, based on continuous execution time distribution, shows when and how task replication can be helpful for certain classes of practically useful scheduling policies. However, its implications are limited to the class of execution time distributions that can be analytical characterized and specific classes of scheduling policies, such as the single-fork and multi-fork policies. To remove these two limitations, in this chapter we investigate the scheduling problem with discrete execution time distributions, which allows more flexible modeling of the execution time distribution, and the finiteness in the support of the distribution allows analysis of more general scheduling policies.

The rest of the chapter is organized as follows. We first discuss the scenarios where discrete execution time distribution naturally occurs in Section 7.1. Based on this, we formulate the optimal scheduling problem as an optimization problem in Section 7.2. However, solving this optimization problem is not straightforward, and we tackle this optimization problem by narrowing down the search space, solving for special yet important cases, and proposing heuristic algorithms. We first analyze the simple yet important case of scheduling a single task optimally in Section 7.3, then proceed to scheduling multiple tasks in Section 7.4.

We focus on the analysis of the cloud user cost in this chapter as the analysis for crowd sourcing cost is straightforward, and use C to denote C_{cloud}.

■ 7.1 Discrete execution time distribution

In this chapter we model the execution time X as a discrete random variable, which corresponds to a p.m.f. P_X, i.e.,

$$X = \alpha_i \text{ w.p. } p_i, \quad 1 \le i \le l, \tag{7.1}$$

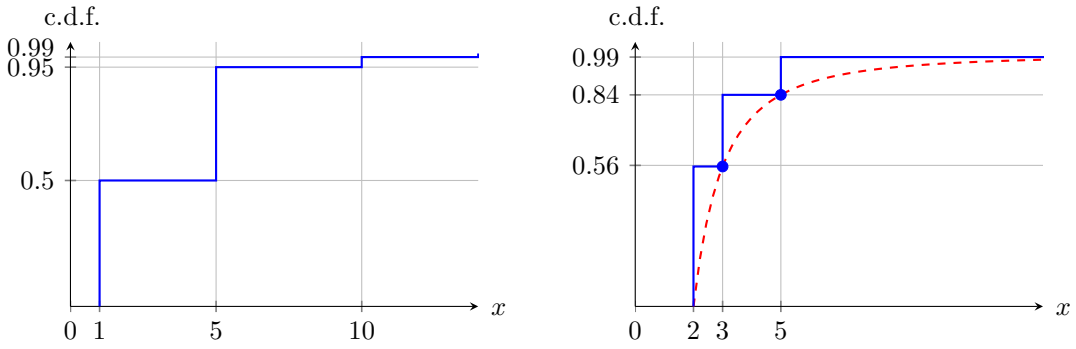
$$\text{or, } P_X(\alpha_i) = p_i. \tag{7.2}$$

where $p_i \in [0, 1]$ and $\sum_{i=1}^{l} p_i = 1$.

This modeling is motivated by the following observations:

1. In practice we often estimate the execution time distribution based on log files or traces, and it is sometimes more convenient to estimate a discrete distribution than a continuous one. For example, a simple estimation could be a histogram of the past execution time spans with certain bin size (e.g., 10 seconds), or from the percentile execution time table such as Table 5.1, as shown in Fig. 7-1a.

2. Given a continuous execution time X, we can use a discrete execution time $\hat{X}$ to derive the lower bound of its performance by setting the c.d.f. of $\hat{X}$ to be the upper envelope of $\bar{F}_X(\cdot)$. An example of for $X \sim \mathsf{Pareto}(2, 2)$ is shown in Fig. 7-1b.



(a) A discrete execution time distribution estimated from Table 5.1.

(b) Pareto distribution (dashed line) and its discrete approximation (solid line).

Figure 7-1: Examples for discrete execution time distribution.

3. A computing node may have multiple states (e.g., idle, with concurrent tasks, stragglers, ...), and in each state i the execution time may be tightly concentrated around some value α_i. Then the execution time distribution for this node over all states is simply (7.2) with probability p_i being the probability that it is in state i.

In particular, in this chapter we will frequently specialize our results to the case that P_X is a *bimodal distribution*, which corresponds to non-zero probability at two time spans, i.e.,

$$X = \begin{cases} \alpha_1 & \text{w.p. } p_1, \\ \alpha_2 & \text{w.p. } p_2 = 1 - p_1. \end{cases} \tag{7.3}$$

The bimodal distribution is observed in real systems, as pointed out by [73, Observation 3], which states task duration distributions are bimodal, with different task types having different task duration distributions. This model captures the phenomena of "stragglers" [36], which indicates the majority of computing nodes in the data centers finish execution in the normal time span, while a small fraction of the nodes takes exceedingly long to complete execution due to malfunctioning of one or multiple part of the data center, such as network congestion, software bugs, bad disk, etc.. In the bimodal distribution (7.3), α_1 can be viewed as the time span that a normal node takes to execute a task, and α_2 the time span that a straggler takes.

■ 7.2 Scheduling objective function

In addition to the trade-off between latency and costs, which are introduced in Section 5.3.3, we define the scheduling objective function, which is simply a convex combination of latency and cost:

$$J_\lambda(\pi) = \lambda \mathbb{E}\left[T(\pi)\right] + (1-\lambda)\mathbb{E}\left[C(\pi)\right], \tag{7.4}$$

where $0 \leq \lambda \leq 1$ reflects the relative importance of latency. As we shall see, sometimes this metric is more convenient as it allows us to define a *optimal policy*.

Definition 7.1 (Optimal and suboptimal policies). *Given λ, then the corresponding* optimal *scheduling policy π^* is*

$$\pi^* = \arg\min_{\pi} J_\lambda(\pi).$$

A scheduling policy π is said to be suboptimal *if there exists another policy π' such that $J_\lambda(\pi') < J_\lambda(\pi)$ for any $\lambda \in [0, 1]$.*

Note that there may exist policies that are neither optimal nor suboptimal.

If we can find the optimal policy π^* for any given λ, then we can schedule tasks optimally. However, this optimization problem is not straightforward because the scheduling objective function is non-convex, and the search space is infinite-dimensional, as we can launch any number of computing nodes at any time.

■ 7.3 Scheduling a single task

In this section we present our results regarding the optimal scheduling for a single task. While appearing simplistic, single-task scheduling is practically useful if we cannot divide a job into multiple parallel tasks. In addition, it is impossible to scheduling multiple tasks optimally if we do not even understand how to schedule a single task optimally.

We defer all proofs to Appendix B.4.

In the single-task setting, the disjoint and joint scheduling policies defined in Section 5.3.2 are equivalent. Therefore, for the single-task scenario, we can focus on the disjoint scheduling policy without any loss of generality, and represent a scheduling policy by its starting time vectors, i.e.,

$$\pi = \mathbf{t} = [t_1, t_2, \ldots, t_m],$$

where t_j is the time that the task starts on computing node j.

Remark 7.1. *Note that the starting time vector $[t_1, \ldots, t_m]$ is equivalent to $[t_1, t_2, \ldots, t_m, \alpha_l, \ldots, \alpha_l]$ as tasks scheduled to start at α_l will never be launched. We use these two representations interchangeably in this chapter.*

The performance metrics defined in Section 5.3.3, latency T and cloud computing cost C_{cloud}, can now be expressed as

$$T(\pi) = \min_{1 \le j \le m} t_j + X_j, \tag{7.5}$$

$$C(\pi) \triangleq C_{\text{cloud}}(\pi) = \sum_{j=1}^{m} |T - t_j|^+, \tag{7.6}$$

where $X_j \overset{i.i.d.}{\sim} P_X$.

■ 7.3.1 Computing the trade-off between latency and cost

In this section we provide methods to compute the trade-off between expected latency $\mathbb{E}[T]$ and expected cost $\mathbb{E}[C]$, which is equivalent to solving the optimization problem

in Section 7.2. Our results reduce the search space of the optimal starting time vector from infinite-dimensional to a finite set, making the computation of the trade-off between latency and cost feasible.

Given the execution time distribution P_X and a starting time vector $\mathbf{t} = [t_1, \ldots, t_m]$, we first show an important property of $\mathbb{E}[T(\mathbf{t})]$ and $\mathbb{E}[C(\mathbf{t})]$ in Theorem 7.1.

Theorem 7.1. *$\mathbb{E}[T(\mathbf{t})]$ and $\mathbb{E}[C(\mathbf{t})]$ are piecewise linear functions of $\mathbf{t}$.*

A further refinement of Theorem 7.1 leads to Theorem 7.2, which indicates the optimal starting time vector $\mathbf{t} \in [0, \alpha_l]^m$ is located in a finite set, which is composed by a constrained integer combination of the support of P_X.

Theorem 7.2. *The starting time vector $\mathbf{t} = [t_1, \ldots, t_m]$ that minimizes J_λ satisfies that*

$$t_j^* \in \mathcal{V}_m, \tag{7.7}$$

where $\mathcal{V}_m$ is a finite set such that

$$\mathcal{V}_m \triangleq \left\{ v : v = \sum_{j=1}^{l} \alpha_j w_j, 0 \le v \le \alpha_l, \sum_{j=1}^{l} |w_j| \le m, w_j \in \mathbb{Z} \right\}. \tag{7.8}$$

Theorem 7.2 directly leads to Corollary 7.3.

Corollary 7.3. *If p.m.f. P_X satisfies that $\alpha_j = k_j \alpha, 1 \le j \le l, k_j \in \mathbb{Z}^+$, then the optimal starting time vector $\mathbf{t}^*$ satisfies*

$$t_j \in \mathcal{V}_m \subset \{0, \alpha, 2\alpha, \ldots, \alpha_l = k_m \alpha\},$$

where $|\mathcal{V}_m| \le k_m + 1$.

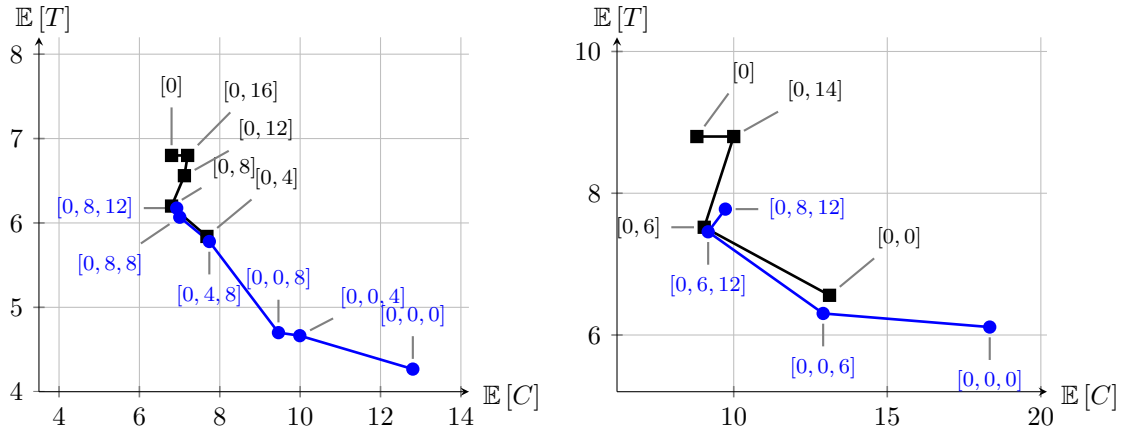
Given Theorem 7.2, we can calculate the $\mathbb{E}[T]$ and $\mathbb{E}[C]$ for all starting time vectors with length m that satisfy (7.7), then discard suboptimal ones, leading to the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off as shown in Fig. 7-2, which are plotted for the following two sample execution times with $m = 3$.

$$X = \begin{cases} 4 & \text{w.p. } 0.6 \\ 8 & \text{w.p. } 0.3 \\ 20 & \text{w.p. } 0.1 \end{cases}, \tag{7.9}$$

$$X' = \begin{cases} 6 & \text{w.p. } 0.8 \\ 20 & \text{w.p. } 0.2 \end{cases}. \tag{7.10}$$

■ 7.3.2 Heuristic policy search algorithm

While Theorem 7.2 reduces the search space of the optimal scheduling policy, the number of policies to evaluate is still exponential in m. In this section we introduce a heuristic single-task scheduling algorithm in Algorithm 3 that has much lower complexity.



(a) Execution time X in (7.9)

(b) Execution time X' in (7.10)

Figure 7-2: Examples of the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off region with $m = 3$. The label of each point indicates the starting time vector, and the region is defined by two piecewise linear segments marked by squares and dots respectively.

We first show that the optimal choice of the $(i+1)$-th element of the starting time vector is dependent on the starting times before it, i.e., $t_1, t_2, \ldots, t_i$, via Theorem 7.4. In particular, the optimal value belongs to a set $\mathcal{U}$ that we called *corner points* that is defined below in Definition 7.2.

Definition 7.2 (Corner points). *Given* $\mathbf{t} = [t_1, t_2, \ldots, t_i]$*, let*

$$\begin{aligned}\mathcal{U}_1 &\triangleq \{0, \alpha_1, \ldots, \alpha_l\},\\ \mathcal{U}_{i+1}(t_1, \ldots, t_i) &\triangleq \bigcup_{u \in \mathcal{U}_i(t_1,\ldots,t_{i-1})} \Big\{ u + b_1 t_i - b_2 \alpha_j : 0 \le u + b_1 t_i - b_2 \alpha_j \le \alpha_l,\\ &\qquad\qquad 1 \le j \le l, b_1, b_2 \in \{0,1\} \Big\}, \quad i \ge 1,\end{aligned}$$

and we called $\mathcal{U}_{i+1}$ *the corner points given* $\mathbf{t}$*.*

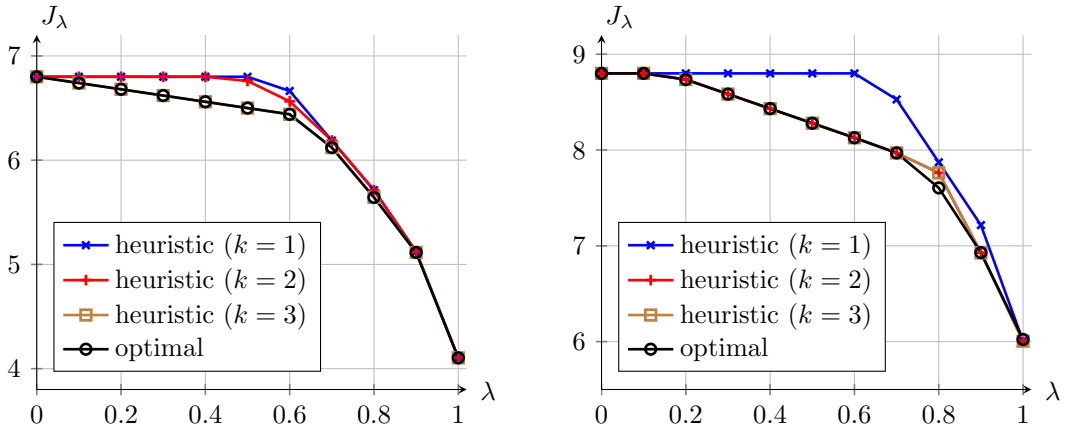
Theorem 7.4. *Given* $\mathbf{t} = [t_1, t_2, \ldots, t_i]$ *and the corner points* $\mathcal{U}_{i+1}(\mathbf{t})$*, then the optimal scheduling policy with* $i+1$ *starting times*

$$\mathbf{t}' = [t_1, t_2, \ldots, t_i, t_{i+1}]$$

satisfies $t_{i+1} \in \mathcal{U}_{i+1}$*.*

Finally, we have the following simple observation that, again, help to reduce the search space of scheduling policy.

Lemma 7.5. *Starting a computing node at any time* $\alpha_l - \alpha_1 \le t \le \alpha_l$ *is suboptimal.*



(a) Execution time X in (7.9). (b) Execution time X' in (7.10).

Figure 7-3: Cost comparisons between the heuristic scheduling policies obtained via Algorithm 3 and optimal scheduling policy obtained via Theorem 7.2, given the starting time vector has length $m = 4$, for different execution time distributions.

With these insights, we propose a heuristic algorithm that builds the starting time vector $[t_1, t_2, \cdots t_m]$ iteratively, with the constraint that t_i's are in non-decreasing order. Given a starting time vector $[t_1, \cdots t_i]$, this algorithm compares the policies $[t_1, \cdots t_i, t_{i+1}]$ where t_{i+1} can be one of first k corner points in $\mathcal{U}(t_1, \ldots, t_i)$, and choose the policy t_{i+1} that leads to the minimum cost. As we increase k, the algorithm compares a larger space of policies and hence achieves a lower cost, as illustrated by the comparisons in Fig. 7-3 for execution times defined in both (7.9) and (7.10) . The example also demonstrates that for the given p.m.f.s, a small k is sufficient to achieve near-optimal trade-off.

Algorithm 3 k-step heuristic algorithm for single-task scheduling

Initialize $t_1 = 0$ and $\mathbf{t} = [t_1]$
for $i = 2, \ldots m$ **do**
  $U^+(\mathbf{t}) \leftarrow$ sorted elements of $\mathcal{U}(\mathbf{t})$ which are $\geq t_{i-1}$
  $\pi_0 \leftarrow [\mathbf{t}, \alpha_l]$, policy that keeps the computing node unused
  for $j = 1, \ldots, k$ **do**
    $\pi_j \leftarrow [\mathbf{t}, U^+(\mathbf{t})[j]]$
  end for
  $j^* \leftarrow \arg\min_{j \in 0,1,\cdots k} J_\lambda(\pi_j)$
  $t_i \leftarrow U^+(\mathbf{t})[j]$ and $\mathbf{t} \leftarrow [\mathbf{t}, t_i]$
end for

■ 7.3.3 Bimodal execution time distribution

While results in Section 7.3.1 characterize the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off and suggest good scheduling policies, a sharper characterization on when and why task replication helps is useful. In this section, we analyze the special yet important case of bimodal execution

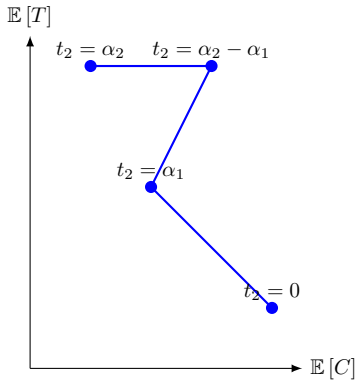


Figure 7-4: The $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off for bimodal execution with two computing nodes, which corresponds to starting time vector $\mathbf{t} = [t_1 = 0, t_2]$.

time distribution (cf. (7.3)), and present a complete characterization of the scenario of scheduling with two computing nodes ($m = 2$). Then for the case of general number of nodes, we analyze the specific strategy that we only introduce task replication at a single time instant, which is similar to the single-fork policy defined in Section 6.1.

Scheduling with two computing nodes

We present results for scheduling one task with two nodes, which is the simplest non-trivial example. The scheduling policy can be represented as the vector $\mathbf{t} = [t_1 = 0, t_2]$, and we provide a complete characterization of the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off in Fig. 7-4, leading to Theorem 7.6.

Theorem 7.6. *Given P_X is a bimodal distribution and at most two nodes, the optimal policy $\mathbf{t} = [t_1 = 0, t_2]$ satisfies $t_2 \in \{0, \alpha_1, \alpha_2\}$.*

In Theorem 7.7, we provide further insights by showing the suboptimality (cf. Definition 7.1) of certain scheduling policies as the execution time distribution P_X varies, which is characterized by the ratio of its fast and slow response time, α_1/α_2, and the probability that it finishes at its fast response time, p_1.

Theorem 7.7. *Given the bimodal execution time and two computing nodes,*

(a) $[0, \alpha_2 - \alpha_1]$ *is always suboptimal;*

(b) $[0, \alpha_1]$ *is suboptimal if* $\frac{\alpha_1}{\alpha_2} > \frac{p_1}{1+p_1}$*;*

(c) $[0, \alpha_2]$ *is suboptimal if* $\frac{\alpha_1}{\alpha_2} < \frac{2p_1-1}{4p_1-1}$*;*

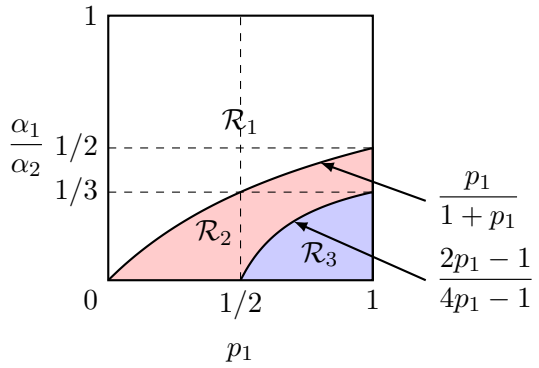


Figure 7-5: Bimodal two computing node. $\mathcal{R}_1$ is the range of parameters that $\mathbf{t} = [0, \alpha_1]$ is strictly suboptimal, $\mathcal{R}_3$ is the range $\mathbf{t} = [0, \alpha_2]$ is strictly suboptimal, which means no task replication is strictly suboptimal.

Given λ, we can find the optimal policy by comparing the ratio $\frac{1-\lambda}{\lambda}$ to the thresholds,

$$\tau_1 = \frac{\alpha_1 p_1(3-2p_1) + \alpha_2(1-p_1)(1-2p_1)}{(\alpha_2-\alpha_1)(1-p_1)p_1} \tag{7.11}$$

$$\tau_2 = \frac{1+2p_1(1-p_1)}{p_1(1-p_1)} \tag{7.12}$$

$$\tau_3 = \frac{\alpha_1(4p_1-1) + \alpha_2(1-2p_1)}{\alpha_2-2\alpha_1)p_1} \tag{7.13}$$

(d) If $\frac{\alpha_1}{\alpha_2} > \frac{p_1}{1+p_1}$, then policy $[0, \alpha_2]$ is optimal if $\frac{1-\lambda}{\lambda} \le \tau_1$, and $[0, 0]$ is optimal otherwise.

(e) If $\frac{2p_1-1}{4p_1-1} \le \frac{\alpha_1}{\alpha_2} \le \frac{p_1}{1+p_1}$, then policy $[0, \alpha_1]$ is optimal if $\tau_3 < \frac{1-\lambda}{\lambda} \le \tau_2$, policy $[0, \alpha_2]$ is optimal if $\frac{1-\lambda}{\lambda} \le \tau_3$, and $[0, 0]$ is optimal otherwise.

(f) If $\frac{\alpha_1}{\alpha_2} < \frac{2p_1-1}{4p_1-1}$, then policy $[0, \alpha_1]$ is optimal if $\frac{1-\lambda}{\lambda} \le \tau_2$, and $[0, 0]$ is optimal otherwise.

Theorem 7.7 is summarized in Fig. 7-5.

Scheduling with multiple computing nodes

For scheduling with m computing nodes, an analysis on general scheduling policy is involved and hence we restrict our attention to the special case of scheduling with replication at only one time instant, which we called the *one-time replication* policy. We show that even this simple strategy often improves the trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$.

Theorem 7.8 (Optimal one-time replication scheduling). *Given bimodal execution time, m computing nodes, and the following form of starting time vector,*

$$\mathbf{t} = [t_1 = 0, t_2 = 0, \ldots, t_{m-d} = 0, t_{m-d+1} = t, \ldots, t_m = t],$$

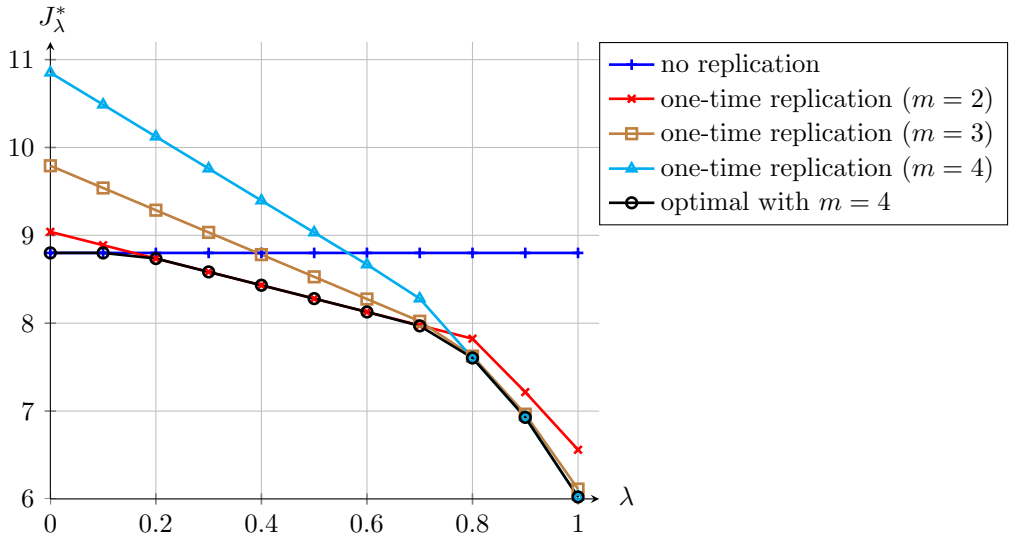


Figure 7-6: The performance of one-time replication polices for the execution time X' in (7.10) with $m = 2, 3, 4$. The optimal policy with $m = 4$ is obtained from Theorem 7.2.

which assumes after $t_1 = 0$*, we only introduce task replication once at time* t*, then the optimal scheduling policy satisfies*

$$t = \alpha_1,$$
$$d = \begin{cases} 0 & \text{when } \bar{\lambda}/(\lambda + m\bar{\lambda}) < p_2^{m-1} p_1 \\ m - 1 & \text{when } \bar{\lambda}/(\lambda + m\bar{\lambda}) > p_1 \\ \lceil a \rceil - 1, \text{ or } \lceil a \rceil & \text{otherwise} \end{cases},$$

where

$$a = m - 1 - \frac{\log(1-\lambda) - \log\left[\lambda + m(1-\lambda)\right] - \log p_1}{\log(1 - p_1)}.$$

As we can see from Fig. 7-6, in this example, replication helps unless λ, the relative importance of latency, is really small, as $m \geq 2$ results in higher costs than the case of no replication. In addition, more computing nodes is useful only when λ is large.

Given a bimodal distribution and m, we can compute the optimal d and hence the optimal cost according to Theorem 7.8. Fig. 7-6 shows that, for the given execution time defined in (7.10), the one-time replication policy with the appropriate number of m coincides with the optimal scheduling policy obtained from Theorem 7.2, i.e., it is possible to achieve optimal performance with replication at one time instant. However, we note this may not be true for other execution time distributions, such as (7.9), because Fig. 7-2a suggests for a certain λ, $[0, 8, 12]$ would the optimal starting time vector, which corresponds to replication at two time instants.

■ 7.4 Scheduling multiple tasks

In this section we investigate the scheduling of multiple tasks. We first show that it is crucial to take the interaction of different tasks into account in Theorem 7.9, then extend our algorithm in Algorithm 3 for multi-task scheduling. All proofs are deferred to Appendix B.5.

Theorem 7.9 (Separation is suboptimal). *Given m tasks, applying the optimal one-task scheduling policy for each of them individually is suboptimal.*

Given the complexity of searching for optimal scheduling policy in the single-task case, we again aim to search for scheduling policy via a heuristic algorithm. In particular, we aim to find a good policy that takes the interaction among tasks into account. To achieve this, we apply Algorithm 3, but using the cost function for the multi-task case, where T and C are defined in (5.3) and (5.4) respectively. This search procedure produces a starting time vector $\mathbf{t} = [t_1, t_2, \ldots, t_m]$, and at each time t_i, we launch an additional copy for each of the unfinished tasks.

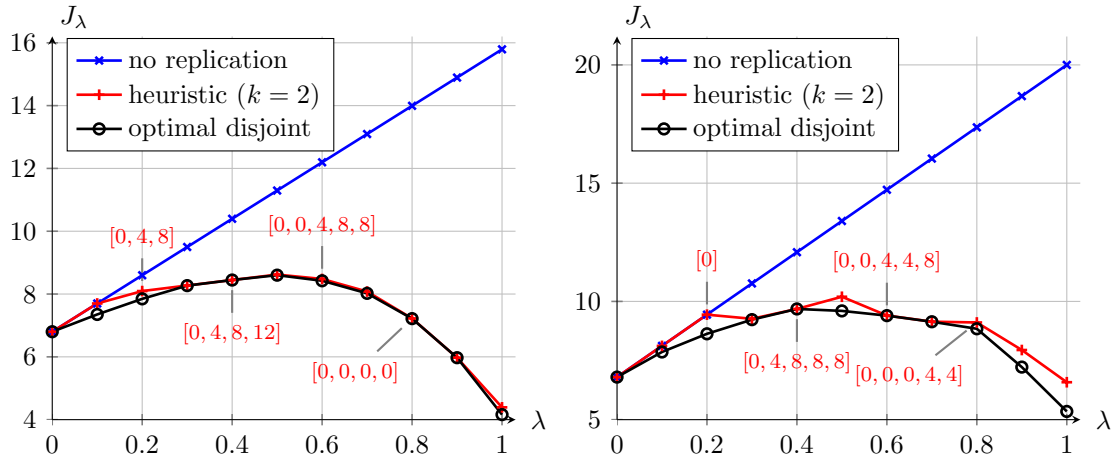
Fig. 7-7 shows an example for the execution time in (7.9). The scheduling policy with replication reduces J_λ, especially when λ is large. We also see that as the number of tasks n increases, the cost J_λ increases as the impact of the slowest task gets more severe. Fig. 7-7 also indicates that when λ is not too big, it may be beneficial to introduce replication at multiple time instants, as in this case, we are more concerned with cloud computing cost C_{cloud} and hence introducing replication gradually is preferred. By contrast, when λ is close to 1, a good scheduling policy should introduce replication early to cut down latency as soon as possible.

Our proposed policy via searching algorithm is a disjoint scheduling policy (cf. Section 5.3.2 for definitions). One may extend it by running the searching algorithm at each time instant. For example, at time $t = 0$ we obtain the starting time vector $t^{(0)}$. At time $t = \alpha_1$ we can re-run the search algorithm given the number of unfinished tasks and obtain an updated starting time vector $t^{(1)}$, etc.. This policy is a joint scheduling policy and is likely to achieve better performance than its disjoint counterpart.

■ 7.5 Concluding Remarks

In this chapter we analyze scheduling policy when tasks share a discrete execution time distribution. For the case of scheduling a single task, we obtain results that enable the computation of the trade-off between expected latency $\mathbb{E}[T]$ and expected cost $\mathbb{E}[C]$, which essentially enables us to schedule a single task optimally. In addition, we propose a heuristic scheduling policy with lower computation complexity and achieve near-optimal costs for the demonstrated examples.

While in single-task scheduling we only need to consider disjoint scheduling policies, in multi-task scheduling we also need to consider joint scheduling policies. This category of policies is more involved to analyze as the time to replicate additional copies of a task could be a random variable that depends on the completion of other tasks. We first show that in general disjoint scheduling policies are suboptimal, and then extend the heuristic scheduling policy for single-task scheduling to the multi-task case. It would be of great



(a) $N = 10$ tasks

(b) $N = 100$ tasks

Figure 7-7: Cost of the heuristic scheduling policy in Algorithm 3 for execution time X in (7.9) with $k = 2$. The starting time vectors for $\lambda = 0.2, 0.4, 0.6$ and 0.8 are labeled in the plots. The optimal disjoint policy is search from all possible starting time vectors that satisfy (7.8).

interest to gain more understanding on the performance limit of joint scheduling policies, which may lead to better scheduling policy designs.

In addition better understanding of scheduling policies, considering a richer set of scheduler actions may further improve performance, as mentioned in Section 5.3.2. In particular, the action TerminateNode can be used to terminate a specific copy of a task preemptively, which may help to achieve better trade-off between latency and resource usage.

Part III

Approximate sorting with noisy comparisons

Chapter 8

The approximate sorting problem with noisy comparisons

> *Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise.*
>
> John Tukey, *The future of data analysis*
> Annals of Mathematical Statistics, 1962

In this part of the thesis we investigate the problem of approximate sorting with noisy comparisons. While similar problems have appeared in a variety of contexts, our investigation is motivated by the application of crowd-based ranking, which suggests that the number of comparisons is an important performance metric, leading to the problem of minimum-comparison approximate sorting. In this chapter we first describe the motivating application in Section 8.1, then define the problem of minimum-comparison approximate sorting in Section 8.2, and finally discuss the related existing works in Section 8.3. In Chapter 9 we present a rate-distortion theory of permutation spaces, a theory that is of interest in its own and provides an information theoretic lower bound for the approximate sorting problem with both noiseless and noisy comparisons.

■ 8.1 Motivating application: crowd-based ranking

In this section we first introduce the notion of crowdsourcing, and then describe crowd-based ranking.

Crowdsourcing is a process that distributes tasks to a group of people, with the objective to achieve some common goal. It exists in many forms. Explicitly, there exist crowdsourcing platforms such as Amazon Mechanical Turk or oDesk, where requesters post simple tasks at various prices and workers solve these tasks for some amount of compensation. Implicitly, crowdsourcing occurs in many group decision making processes, such as movie ranking and the college admission process, where the opinions of a group of people are polled to reach a decision. Many examples of crowdsourcing, especially online ones, share the common characteristic that can be summarized in the following question:

> How can we solve a complex problem by gathering inaccurate answers to a list of simple questions?

In this part of the thesis we aim to answer the above question in the context of *crowd-based ranking*. Ranking plays an important role in many aspects of our life, as it summarizes detailed features of a set of objects to a sequence of numbers, and makes it possible to evaluate complex information straightforwardly. In crowd-based ranking, the opinions of experts/users are queried and aggregated to produce a ranking. One of the

most salient example may be the Internet Movie Database (`IMDb.com`), where users provide ratings for movies they have watched, and these information is aggregated to produce the top 250 movies of all time, an extremely difficult task for most users to accomplish on their own.

In crowd-based ranking, the queries are usually in two formats: ratings and comparisons. In *ranking via ratings*, users assign numerical value (rating score) to each item, as in the Internet Movie Database example, while in *ranking via comparisons*, users provide comparison results for a pair of items in terms of preference. Again, this may happen either explicitly or implicitly, in which binary actions such as clicking a link or voting up or down can be used as the comparison outcomes.

In this part of the thesis we focus on the problem of ranking via comparisons. Noting that one key feature of crowd-based ranking is that *query cost is significant*, because human queries are expensive and/or time consuming. In addition, inaccuracies are inherent in the crowdsourcing process, so we should aim to rank approximately rather than exactly. Based on these observations, we aim to analyze approximate ranking algorithms that used the minimum number of comparisons, leading the problem of *minimum-comparisons approximate sorting* defined in Section 8.2.

■ 8.2 Minimum-comparison approximate sorting

In this section we first introduce the notation and facts used in this part of the thesis in Section 8.2.1, then setup the minimum-comparison approximate sorting problem in Section 8.2.2.

■ 8.2.1 Notation and facts

In addition to the notation introduced in Section 1.1, we introduce the following notation in this part of the thesis.

Let $\mathcal{S}_n$ denote the symmetric group of n elements. We write the elements of $\mathcal{S}_n$ as arrays of natural numbers with values ranging from $1, \ldots, n$ and every value occurring only once in the array. For example, $\sigma = [3, 4, 1, 2, 5] \in \mathcal{S}_5$. This is also known as the *vector notation* for permutations. For a permutation σ, we denote its permutation inverse by σ^{-1}, where $\sigma^{-1}(x) = i$ when $\sigma(i) = x$, and $\sigma(i)$ is the i-th element in array σ. For example, the permutation inverse of $\sigma = [2, 5, 4, 3, 1]$ is $\sigma^{-1} = [5, 1, 4, 3, 2]$. We denote the *identity permutation* by Id, i.e., $\mathrm{Id} \triangleq [1, 2, \ldots, n]$. Given a metric $d : \mathcal{S}_n \times \mathcal{S}_n \to \mathbb{R}^+ \cup \{0\}$, we define a *permutation space* $\mathcal{X}(\mathcal{S}_n, d)$.

We let $[a : b] \triangleq \{a, a + 1, \ldots, b - 1, b\}$ for any two integers a and b.

We introduce *Stirling's approximation*, a frequently-used technique: for $m \in \mathbb{Z}^+$,

$$\left(\frac{m}{e}\right)^m e^{\frac{1}{12m+1}} < \frac{m!}{\sqrt{2\pi m}} < \left(\frac{m}{e}\right)^m e^{\frac{1}{12m}}. \tag{8.1}$$

■ 8.2.2 Problem definition

Given a set of items $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ such that

$$v_{\sigma^{-1}(1)} \succ v_{\sigma^{-1}(2)} \succ \ldots \succ v_{\sigma^{-1}(n)},$$

where $a \succ b$ indicates a is preferred to b, then we say the permutation σ is the *true ranking* of these list of items, where $\sigma(i)$ provides the rank of item i, and $\sigma^{-1}(r)$ provides the index of the item with rank r.

If an algorithm produces a ranking π based on $f(n)$ pairwise comparisons of items in $\mathcal{V}$, we say this algorithm is an *approximate sorting* algorithm with *query complexity* $f(n)$ and distortion $d(\sigma, \pi)$, where $d(\cdot, \cdot)$ is some distance measure on the symmetric group $\mathcal{S}_n$.

We consider two types of comparisons, noiseless comparison and noisy comparison. Given a true ranking σ, for any $a, b \in \mathcal{S}_n$, a noiseless comparison satisfy

$$\operatorname{cmp}(a, b) = \begin{cases} 1 & a \succ_\sigma b \\ 0 & a \prec_\sigma b \end{cases},$$

where $\succ_\sigma$ indicates preference based on the ranking σ. A noisy comparison produces a random outcome, which equals the noiseless comparison with probability $1 - \varepsilon$, i.e.,

$$\widetilde{\operatorname{cmp}}(a, b) = \begin{cases} \mathsf{Bern}(1-\varepsilon) & a \succ_\sigma b \\ \mathsf{Bern}(\varepsilon) & a \prec_\sigma b \end{cases}, \tag{8.2}$$

Remark 8.1. *In certain applications, the error probability of the comparison may depends on the two items being compared, and we can adopt ranking models such as the Thurstone model [74] to model the error probability in pairwise comparisons.*

While algorithms for approximate sorting with both noiseless and noisy comparisons have been investigated, the fundamental performance limits of approximate sorting algorithms are not established. In Chapter 9, our analysis via rate distortion theory establishes the lower bounds of query complexity for approximate sorting with both noiseless comparisons and noisy comparisons, and is shown to be tight for the noiseless case. Furthermore, as our lower bound results indicate, for the number of comparisons in approximate sorting, asymptotic analysis in terms of the Big-O notation is often too coarse because the constant term in front of the dominant factor matters. We called an approximate sorting algorithm that achieves the optimal constant *constant optimal*.

■ 8.3 Related works and problems

Sorting is a rich subject, and in this section we describe various existing work that relate to the problem of approximate sorting with noisy comparisons.

As discussed below, while algorithms for minimum comparison sorting problem have been proposed, the problem of minimum-comparison approximate sorting problem with noisy comparisons is still open.

Minimum comparison sorting

Regular sorting with minimum number of comparisons has been studied in [75, Chapter 5.3]. A few well-known sorting algorithms, including *merge sort*, *tree selection sort* and *binary insertion sort*, are minimum comparison sorting algorithms.

Sorting under partial information and partial order production

The problem of *partial order production* aims to "arrange the elements in an unknown totally ordered set $\mathcal{V}$ into a target partially ordered set, by comparing a minimum number of pairs in $\mathcal{V}$" [76]. It was first proposed by [77], then [78] provides an information-theoretic lower bound in terms of $\log n!/e(P)$, where P is the target partially ordered set (poset) and $e(P)$ is the number of linear extensions of P. For this problem, a solution that achieves the information-theoretic lower bound was proposed in [76].

While the output of a partial order production algorithm is a poset, this poset corresponds to at least one linear extension, i.e., a fully ordered set that corresponds to a permutation. Therefore, any partial order production algorithm can be converted to an algorithm for the approximate sorting problem defined in Section 8.2.

The problem of *sorting under partial information* sorts a set of elements with some given outcomes of comparisons between some pairs of the elements, which can be seen as a complement of the problem of partial order production. It is an important problem that been well investigated in [79, 80].

Partial sorting with noiseless comparisons

The problem of *partial sorting* is first proposed in [81]. Given a set of n elements $\mathcal{V}$ and a set of indices $\mathcal{I} \subset \{1, 2, \ldots, n\}$, a partial sorting algorithm aims to arrange the elements into a list $[v_1, v_2, \ldots, v_n]$ such that for any $i \in \mathcal{I}$, all elements with indices $j < i$ are no greater than v_i, and all elements with indices $j' > i$ are no smaller than v_i. A partial sorting algorithm essentially selects all elements with ranks in the set $\mathcal{I}$, and hence is also called *multiple selection*. A lower bound on the query complexity for the *multiple selection* problem is proposed in [82], and the algorithm proposed by [83] achieves this lower bound.

In addition, [84] points out that multiple selection can be seen a special case of the problem of partial order production. Therefore, a variety of results on partial order production could be useful for the problem of partial sorting as well.

While it is not straightforward to extend the minimum comparison sorting algorithms above to partial sorting as they all build a global ordering from a local ordering and any local disorder leads to error propagation, it is straightforward to extend *quicksort* to partial sorting. Indeed, simply by stopping at a proper partition size, *quicksort* achieves $D = O\left(n^{1+\delta}\right)$ with $1.39(1-\delta)n \log_2 n + o\left(n \log_2 n\right)$ comparisons on average. The analysis follows from [75, Chapter 5.2.2] or [85]. A more general version of the algorithm is discussed in [86].

Finally, the soft heap idea [87, 88] can also be applied to partial sorting.

It is not hard to see that given the same distortion, the set of partial sorting algorithms is a subset of approximate sorting algorithms defined in Section 8.2, because approximate sorting algorithms do not require their outputs to be in the form specified by for the partial sorting problem.

Sorting and selection with noisy comparisons

There have been a variety of work on algorithms based on noisy comparisons.

Most of work adopts the "probably correct" performance measure, aiming to produce a correct output with $1 - q$ probability, which can be translated to approximate sorting in terms of average distortion. In [89], the problems of sorting and k-selection of n elements

with noisy comparisons and correct answer probability requirement at least $1 - q$ are investigated. The main idea is to repeat comparisons appropriately, and the analysis is based on analyzing a random walk on a binary search tree. In particular, for sorting, $\Theta(n \log n/q)$ noisy comparisons are needed, and for k-selection, $\Theta(n \log(m/q))$, where $m = \min\{k, n-k\}$ noisy comparisons are needed.

Another line of work focuses on sorting under different performance measures and/or constraints. In [90], the problem of noisy sorting without resampling (i.e., repeating is not helpful) is considered. An algorithm that finds the maximum likelihood order with $O(n \log n)$ sample complexity is provided, and with high probability achieves ℓ_1 distortion $\Theta(n)$. In [91], a variety of noisy comparison-based problems are investigated, again with the no resampling constraint. Relevant results include finding the maximum of n numbers in $O(n)$ time and approximately sort n numbers in $O(n^2)$ time such that each number's position deviates from its true rank by at most $O(\log n)$ positions.

To our best knowledge, all existing work on noisy comparisons algorithms adopt the common asymptotic complexity analysis measure and hence do not take the constant in front of the asymptotic term into account.

Sorting with distributional models

In classical sorting research it is implicitly assumed that all sorting outcome are equally likely, i.e., the permutation that corresponding to the ordering are uniformly distributed over the $\mathcal{S}_n$. However, in some applications, one may have prior knowledge about the distribution of the ordering, such as the Mallows model [92], a popular distributional model in ranking applications. The learning of Mallows model in [93] can be viewed as sorting under the Mallows distributional model.

Note that sorting with partial information can be viewed as sorting with a distributional model, where the distribution is uniform on a subset of $\mathcal{S}_n$ that agrees with the partial information. Therefore, sorting with distributional model is a more general formulation than sorting with partial information.

Information theoretic lower bound for algorithms

Using information-theoretic tools to derive lower bounds for algorithm complexity is a well-known technique. For the multiple-selection problem, [82] analyzes the information theoretic bound and claims the information-theoretic bound is "usually very weak". Later, results in [94] and [95] shows that the information-theoretic bound is tight up to a multiplicative constant for merging and sorting under partial information respectively (see also [96]).

Rank aggregation

Rank aggregation [97] investigates the problem of combining rankings of subsets of data into a ranking of the whole set of the data, where comparison-based ranking can be seen as a special case (each comparison produces a ranking of subset of size 2). Some recent work include the algorithm for aggregating (possibly conflicting) pairwise preferences to get an overall rank with minimum conflicts in [98], and rank aggregation with partial data in [99].

Most of rank aggregation investigations assumes a given set of rankings. One exception is [100], where the problem of active ranking via pairwise comparison is investigated. However, the problem setup is constrained as they assume all the objects can be embedded into a d-dimensional Euclidean space and the overall rankings is determined by the objects' relative distances to a common reference point in $\mathbb{R}^d$.

Learning to rank

Learning to rank is an important problem in search and information retrieval [101–105]. While the problem here is also finding a permutation based on some input (possibly pairwise comparisons), a loss function between the learned permutation and the true permutation is assumed. In the sorting setup, no such loss function is available, making the problem more difficult.

Chapter 9

A rate-distortion theory for permutation spaces

9.1 Introduction

As discussed in Chapter 8, in this part we are interested in the problem of minimum comparison approximate sorting algorithms with noisy comparisons, and in this chapter we derive the fundamental limits on the query complexity via analyzing the trade-off between rate and distortion for permutation spaces.

In addition to analyzing the query complexity of approximate sorting, a rate-distortion theory for permutation spaces also provides the fundamental limits on the lossy compression of permutations, which is of interest to the application of storing ranking data. In applications such as recommendation systems, it may be necessary to store the ranking of all users in the system, and hence the storage efficiency of ranking data is of interest. Furthermore, in many cases a rough knowledge of the ranking (e.g., finding one of the top five elements instead of the top element) is sufficient. Because a ranking of n items can be represented as a permutation of 1 to n, storing a ranking is equivalent to storing a permutation. This poses the question of the number of bits needed for permutation storage when a certain amount distortion can be tolerated.

Remark 9.1 (Compression is easier than sorting). *It is worth noting that compression is easier than sorting, in the sense that every comparison-based sorting algorithm corresponds to a compression scheme of the permutation space, while the reverse does not hold in general. In particular, the string of bits that represent comparison outcomes in any deterministic (approximate) sorting algorithm corresponds to a (lossy) representation of the permutation.*

Besides the above applications, the rate-distortion theory on permutation spaces is of technical interest on its own because the permutation space does not possess the product structure that a discrete memoryless source induces.

With the above motivations, we consider the rather fundamental problem of lossy compression in permutation spaces in this chapter, which provide insights on both approximate sorting and lossy compression of ranking data. Following the classical rate-distortion setup, we aim to answer the question that given a distortion measure $d(\cdot, \cdot)$ and the amount of distortion D allowed, what is the minimum number of the bits needed to describe the permutations?

The analysis of the lossy compression problem depends on the source distribution and the distortion measure. We mainly concern with the permutation spaces with a uniform distribution, and consider different distortion measures based on four distances in the permutation spaces: the Kendall tau distance, Spearman's footrule, Chebyshev distance and inversion-ℓ_1 distance. As we shall see in Section 9.2, each of these distortion measures has its own operational meaning that may be useful in different applications.

Besides characterizing the trade-off between rate and distortion, we also show that under the uniform distribution over the permutation space, there are close relationships between all the distortion measures of interest in this chapter. We use these relations to establish the equivalence of source codes in permutation spaces with different distortion measures. Our results indicate that, while these distance measures usually have different intended applications, an optimal coding scheme for one distortion measure is effectively optimal for other distortion measures. For each distortion measure, we provide simple and constructive achievability schemes, leading to explicit source code designs with low complexity.

Finally, we turn our attention to non-uniform distribution over the permutation spaces, as in some application we may have prior knowledge about the permutation data, which can be captured in a non-uniform distribution. There are a variety of distributional models, such as the Bradley-Terry model [106], the Luce-Plackett model [107,108], and the Mallows model [92] that rise in different contexts. Among these, we choose the Mallows model due to its richness and applicability in various ranking applications [93,109,110]. We analyze the lossless and lossy compression of the permutation space under the Mallows model and with the Kendall tau distance as the distortion measure, and characterize its entropy and end points of its rate-distortion function.

Our analysis provides an information-theoretic lower bound on query complexity for *all* approximate sorting algorithms that achieve a certain distortion, while as discussed in Section 8.3, the lower bound in [82] only holds for the class of partial sorting algorithms that achieve the same distortion. The multiple selection algorithm proposed in [83] turns out to be optimal for the general approximate sorting problem as well and hence our information-theoretic lower bound is tight.

The rest of the chapter is organized as follows. We first present the problem formulation in Section 9.2. We then show that there exist close relationships between the distortion measures of interest in this chapter in Section 9.3. In Section 9.4, we derive the rate-distortion functions for different permutation spaces. In Section 9.5, we provide achievability schemes for different permutation spaces under different regimes. After that, we turn our attention to non-uniform distributional model over the permutation space and analyze the lossless and lossy compression for Mallows model in Section 9.6. Finally we apply our rate-distortion theory results and provide lower bounds on the query complexity of approximate sorting with both noiseless and noisy comparisons in Section 9.7 and conclude in Section 9.8.

■ 9.2 Problem formulation

In this section we discuss aspects of the problem formulation of the rate-distortion problem for permutation spaces. We first introduce the distortions of interest in Section 9.2.1, and then provide a mathematical formulation of the rate-distortion problem in Section 9.2.2.

■ 9.2.1 Distortion measures

For distortion measures, it is natural to use distance measures on the permutation set $\mathcal{S}_n$, and there exist many possibilities [111]. In this chapter we choose a few distortion measures of interest in a variety of application settings, including Spearman's footrule (ℓ_1

distance between two permutation vectors), Chebyshev distance (ℓ_∞ distance between two permutation vectors), Kendall tau distance and the inversion-ℓ_1 distance.

Given a list of items with values $v_1, v_2, \ldots, v_n$ such that $v_{\sigma^{-1}(1)} \succ v_{\sigma^{-1}(2)} \succ \ldots \succ v_{\sigma^{-1}(n)}$, where $a \succ b$ indicates a is preferred to b, then we say the permutation σ is the ranking of these list of items, where $\sigma(i)$ provides the rank of item i, and $\sigma^{-1}(r)$ provides the index of the item with rank r. Note that sorting via pairwise comparisons is simply the procedure of rearranging $v_1, v_2, \ldots, v_n$ to $v_{\sigma^{-1}(1)}, v_{\sigma^{-1}(2)}, \ldots, v_{\sigma^{-1}(n)}$ based on preferences from pairwise comparisons.

Given two rankings σ_1 and σ_2, we measure the total deviation of ranking and maximum deviation of ranking by Spearman's footrule and Chebyshev distance respectively.

Definition 9.1 (Spearman's footrule). *Given two permutations $\sigma_1, \sigma_2 \in \mathcal{S}_n$, the* Spearman's footrule *between σ_1 and σ_2 is*

$$d_{\ell_1}(\sigma_1, \sigma_2) \triangleq \|\sigma_1 - \sigma_2\|_1 = \sum_{i=1}^{n} |\sigma_1(i) - \sigma_2(i)|.$$

Definition 9.2 (Chebyshev distance). *Given two permutations $\sigma_1, \sigma_2 \in \mathcal{S}_n$, the* Chebyshev distance *between σ_1 and σ_2 is*

$$d_{\ell_\infty}(\sigma_1, \sigma_2) \triangleq \|\sigma_1 - \sigma_2\|_\infty = \max_{1 \le i \le n} |\sigma_1(i) - \sigma_2(i)|.$$

The Spearman's footrule in $\mathcal{S}_n$ is upper bounded by $\lfloor n^2/2 \rfloor$ (cf. Lemma C.6) and the Chebyshev distance in $\mathcal{S}_n$ is upper bounded by $n - 1$.

Given two lists of items with ranking σ_1 and σ_2, let $\pi_1 \triangleq \sigma_1^{-1}$ and $\pi_2 \triangleq \sigma_2^{-1}$, then we define the number of pairwise adjacent swaps on π_1 that changes the ranking of π_1 to the ranking of π_2 as the Kendall tau distance.

Definition 9.3 (Kendall tau distance). *The* Kendall tau distance $d_\tau(\sigma_1, \sigma_2)$ *from one permutation σ_1 to another permutation σ_2 is defined as the minimum number of transpositions of pairwise adjacent elements required to change σ_1 into σ_2.*

The Kendall tau distance is upper bounded by $\binom{n}{2}$.

Example 9.1 (Kendall tau distance). *The Kendall tau distance for $\sigma_1 = [1, 5, 4, 2, 3]$ and $\sigma_2 = [3, 4, 5, 1, 2]$ is $d_\tau(\sigma_1, \sigma_2) = 7$, as one needs at least 7 transpositions of pairwise adjacent elements to change σ_1 to σ_2. For example,*

$$\begin{aligned}\sigma_1 &= [1, 5, 4, 2, 3] \\ &\to [1, 5, 4, 3, 2] \to [1, 5, 3, 4, 2] \to [1, 3, 5, 4, 2] \\ &\to [3, 1, 5, 4, 2] \to [3, 5, 1, 4, 2] \to [3, 5, 4, 1, 2] \\ &\to [3, 4, 5, 1, 2] = \sigma_2.\end{aligned}$$

Being a popular global measure of disarray in statistics, Kendall tau distance also has a natural connection to sorting algorithms. In particular, given a list of items with values

$v_1, v_2, \ldots, v_n$ such that $v_{\sigma^{-1}(1)} \succ v_{\sigma^{-1}(2)} \succ \ldots \succ v_{\sigma^{-1}(n)}$, $d_\tau\left(\sigma^{-1}, \mathrm{Id}\right)$ is the number of swaps needed to sort this list of items in a bubble-sort algorithm [75].

Finally, we introduce a distortion measure based on the concept of inversion vector, another measure of the order-ness of a permutation.

Definition 9.4 (inversion, inversion vector). *An* inversion *in a permutation $\sigma \in \mathcal{S}_n$ is a pair $(\sigma(i), \sigma(j))$ such that $i < j$ and $\sigma(i) > \sigma(j)$.*

We use $I_n(\sigma)$ to denote the total number of inversions in $\sigma \in \mathcal{S}_n$, and

$$K_n(k) \triangleq |\{\sigma \in \mathcal{S}_n : I_n(\sigma) = k\}| \tag{9.1}$$

to denote the number of permutations with k inversions.

Denote $i' = \sigma(i)$ and $j' = \sigma(j)$, then $i = \sigma^{-1}(i')$ and $j = \sigma^{-1}(j')$, and thus $i < j$ and $\sigma(i) > \sigma(j)$ is equivalent to $\sigma^{-1}(i') < \sigma^{-1}(j')$ and $i' > j'$.

A permutation $\sigma \in \mathcal{S}_n$ is associated with an inversion vector $\mathbf{x}_\sigma \in \mathcal{G}_n \triangleq [0:1] \times [0:2] \times \cdots \times [0:n-1]$, *where $\mathbf{x}_\sigma(i'), 1 \le i' \le n-1$ is the number of inversions in σ in which $i'+1$ is the first element. Mathematically, for $i' = 2, \ldots, n$,*

$$\mathbf{x}_\sigma(i'-1) = \left|\left\{j' \in [n] : j' < i', \sigma^{-1}(j') > \sigma^{-1}(i')\right\}\right|.$$

Let $\pi \triangleq \sigma^{-1}$, then the *inversion vector* of π, $\mathbf{x}_\pi$, measures the deviation of ranking σ from Id. In particular, note that

$$\begin{aligned}\mathbf{x}_\pi(k) &= \left|\left\{j' \in [n] : j' < k, \pi^{-1}(j') > \pi^{-1}(k)\right\}\right| \\ &= \left|\left\{j' \in [n] : j' < k, \sigma(j') > \sigma(k)\right\}\right|\end{aligned}$$

indicates the number of elements that have *larger ranks* and *smaller item indices* than that of the element with index k. In particular, the rank of the element with index n is $n - \mathbf{x}_\pi(n-1)$.

Example 9.2. *Given 5 items such that $v_4 \succ v_1 \succ v_2 \succ v_5 \succ v_3$, then the inverse of the ranking permutation is $\pi = [4, 1, 2, 5, 3]$, with inversion vector $\mathbf{x}_\pi = [0, 0, 3, 1]$. Therefore, the rank of the v_5 is $n - \mathbf{x}_\pi(n-1) = 5 - 1 = 4$.*

It is well known that mapping from $\mathcal{S}_n$ to $\mathcal{G}_n$ is one-to-one and straightforward [75].

With these, we define the inversion-ℓ_1 distance.

Definition 9.5 (inversion-ℓ_1 distance). *Given two permutations $\sigma_1, \sigma_2 \in \mathcal{S}_n$, we define the inversion-ℓ_1 distance, ℓ_1 distance of two inversion vectors, as*

$$d_{\mathbf{x},\ell_1}(\sigma_1, \sigma_2) \triangleq \sum_{i=1}^{n-1} |\mathbf{x}_{\sigma_1}(i) - \mathbf{x}_{\sigma_2}(i)|. \tag{9.2}$$

Example 9.3 (inversion-ℓ_1 distance). *The inversion vector for permutation $\sigma_1 = [1, 5, 4, 2, 3]$ is $\mathbf{x}_{\sigma_1} = [0, 0, 2, 3]$, as the inversions are $(4, 2), (4, 3), (5, 4), (5, 2), (5, 3)$. The inversion vector for permutation $\sigma_2 = [3, 4, 5, 1, 2]$ is $\mathbf{x}_{\sigma_2} = [0, 2, 2, 2]$, as the inversions are $(3, 1), (3, 2), (4, 1)$,*

$(4,2),(5,1),(5,2)$. *Therefore,*

$$d_{\mathbf{x},\ell_1}(\sigma_1,\sigma_2) = d_{\ell_1}([0,0,2,3],[0,2,2,2]) = 3.$$

As we shall see in Section 9.3, all these distortion measures are related to each other. While the operational significance of the inversion-ℓ_1 distance may not be as clear as other distortion measures, some of its properties provide useful insights in the analysis of other distortion measures.

Remark 9.2. *While Spearman's footrule and Chebyshev distance operates on the* ranking domain*, inversion vector and Kendall tau distance could be seen to operate on* the inverse of the ranking domain*.*

Remark 9.3. *The ℓ_1, ℓ_∞ distortion measures above can be readily generalized to weighted versions to incorporate different emphasis on different parts of the ranking.*

In particular, using a weighted version that only puts non-zero weight to the first k components of the permutation vector corresponds to the case that we only the distortion of the top-k items (top-k selection problem).

■ 9.2.2 Rate-distortion problems

With the distortions defined in Section 9.2.1, in this section we define the rate-distortion problems under both average-case distortion and worst-case distortion.

Definition 9.6 (Codebook for average-case distortion). *An (n, D_n) source code $\bar{\mathcal{C}}_n \subseteq \mathcal{S}_n$ for $\mathcal{X}(\mathcal{S}_n, d)$ under average-case distortion is a set of permutations such that for a σ that is drawn from $\mathcal{S}_n$ according to a distribution P on $\mathcal{S}_n$, there exists an encoding mapping $f_n : \mathcal{S}_n \to \bar{\mathcal{C}}_n$ that*

$$\mathbb{E}_P[d(f_n(\sigma),\sigma)] \le D_n. \tag{9.3}$$

The mapping $f_n : \mathcal{S}_n \to \bar{\mathcal{C}}_n$ can be assumed to satisfy

$$f_n(\sigma) = \arg\min_{\sigma' \in \bar{\mathcal{C}}_n} d(\sigma', \sigma)$$

for any $\sigma \in \mathcal{S}_n$.

Definition 9.7 (Codebook for worst-case distortion). *The codebook for permutations under worst-case distortion can be defined analogously to Definition 9.6, except (9.3) now becomes*

$$\max_{\sigma \in \mathcal{S}_n} d(f_n(\sigma),\sigma) \le D. \tag{9.4}$$

We use $\hat{\mathcal{C}}_n$ to denote a (n, D_n) source code under the worst-case distortion.

In this chapter we mostly focus on the case P is uniformly distributed over the symmetric group $\mathcal{S}_n$, except in Section 9.6, where a distribution rising from the Mallows model is used.

Definition 9.8 (Rate function). *Given a source code $\mathcal{C}_n$ and a distortion D_n, let $A(n, D_n)$ be the minimum size of $\mathcal{C}_n$, and we define the minimal rate for distortions D_n as*

$$R(D_n) \triangleq \frac{\log A(n, D_n)}{\log n!}.$$

In particular, we denote the minimum rate of the codebook under average-case and worst-case distortions by $\bar{R}(D_n)$ and $\hat{R}(D_n)$ respectively.

Similar to the classical rate-distortion setup, we are interested in deriving the trade-off between distortion level D_n and the rate $R(D_n)$ as $n \to \infty$. In this work we show that for the distortions $d(\cdot, \cdot)$ and the sequences of distortions $\{D_n, n \in \mathbb{Z}^+\}$ of interest, $\lim_{n\to\infty} R(D_n)$ exists.

For Kendall tau distance and inversion-ℓ_1 distance, a close observation shows that in regimes such as $D_n = O(n)$ and $D_n = \Theta\left(n^2\right)$, $\lim_{n\to\infty} R(D_n) = 1$ and $\lim_{n\to\infty} R(D_n) = 0$ respectively. In these two regimes, the trade-off between rate and distortion is really shown in the higher order terms in $\log A(n, D_n)$, i.e.,

$$r(D_n) \triangleq \log A(n, D_n) - \log n! \lim_{n\to\infty} R(D_n). \tag{9.5}$$

For convenience, we categorize the distortion D_n under Kendall tau distance or inversion-ℓ_1 distance into three regimes. We say D is small when $D = \Theta(n)$, moderate when $D = \Theta\left(n^{1+\delta}\right), 0 < \delta < 1$, and large when $D = \Theta\left(n^2\right)$[1].

We choose to omit the higher order term analysis for $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$ because its analysis is essentially the same as $\mathcal{X}(\mathcal{S}_n, d_\tau)$, and the analysis for $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty})$ is still open.

Note that the higher order terms $r(D_n)$ may behave differently under average and worst-case distortions, and in this chapter we restrict our attention to the worst-case distortion.

■ 9.3 Relationships between distortion measures

In this section we show all four distortion measures defined in Section 9.2.1 are closely related to each other, which is summarized in (9.6) and (9.7). These relationships implies a variety of equivalence in the lossy compression schemes, which we exploit to derive the rate-distortion functions in Section 9.4.

For any $\sigma_1 \in \mathcal{S}_n$ and σ_2 is randomly uniformly chosen from $\mathcal{S}_n$,

$$nd_{\ell_\infty}(\sigma_1, \sigma_2) \geq d_{\ell_1}(\sigma_1, \sigma_2) \geq d_\tau\left(\sigma_1^{-1}, \sigma_2^{-1}\right) \geq d_{\mathbf{x},\ell_1}\left(\sigma_1^{-1}, \sigma_2^{-1}\right), \tag{9.6}$$

$$nd_{\ell_\infty}(\sigma_1, \sigma_2) \overset{w.h.p.}{\lesssim} d_{\ell_1}(\sigma_1, \sigma_2) \lesssim d_\tau\left(\sigma_1^{-1}, \sigma_2^{-1}\right) \overset{w.h.p.}{\lesssim} d_{\mathbf{x},\ell_1}\left(\sigma_1^{-1}, \sigma_2^{-1}\right), \tag{9.7}$$

where $x \lesssim y$ indicates $x < c \cdot y$ for some constant $c > 0$, and $\overset{w.h.p.}{\lesssim}$ indicates $\lesssim$ with high probability.

[1] In the small distortion region with $R(D_n) = 1$, $r(D_n)$ is negative while in the large distortion region where $R(D_n) = 0$, $r(D_n)$ is positive.

The following sections provide detailed arguments for (9.6) and (9.7) by analyzing the relationship between different pairs of distortion measures.

Spearman's footrule and Chebyshev distance

Let σ_1 and σ_2 be any permutations in $\mathcal{S}_n$, then by definition,

$$d_{\ell_1}(\sigma_1, \sigma_2) \leq n \cdot d_{\ell_\infty}(\sigma_1, \sigma_2), \tag{9.8}$$

and additionally, a scaled Chebyshev distance lower bounds the Spearman's footrule with high probability.

Theorem 9.1. *For any $\pi \in \mathcal{S}_n$, let σ be a permutation chosen uniformly from $\mathcal{S}_n$, then*

$$\mathbb{P}\left[c_1 \cdot n \cdot d_{\ell_\infty}(\pi, \sigma) \leq d_{\ell_1}(\pi, \sigma)\right] \geq 1 - O(1/n) \tag{9.9}$$

for any positive constant $c_1 < 1/3$.

Proof. See Appendix C.2.1. □

Spearman's footrule and Kendall tau distance

The following theorem is a well-known result on the relationship between Kendall tau distance and ℓ_1 distance of permutation vectors.

Theorem 9.2 ([112]). *Let σ_1 and σ_2 be any permutations in $\mathcal{S}_n$, then*

$$d_{\ell_1}(\sigma_1, \sigma_2)/2 \leq d_\tau(\sigma_1^{-1}, \sigma_2^{-1}) \leq d_{\ell_1}(\sigma_1, \sigma_2). \tag{9.10}$$

inversion-ℓ_1 distance and Kendall tau distance

We show that the inversion-ℓ_1 distance and the Kendall tau distance are closely related in Theorem 9.3 and Theorem 9.4, which helps to establish the equivalence of the rate-distortion problem later.

Theorem 9.3. *Let σ_1 and σ_2 be any permutations in $\mathcal{S}_n$, then for $n \geq 2$,*

$$\frac{1}{n-1} d_\tau(\sigma_1, \sigma_2) \leq d_{\mathbf{x},\ell_1}(\mathbf{x}_{\sigma_1}, \mathbf{x}_{\sigma_2}) \leq d_\tau(\sigma_1, \sigma_2). \tag{9.11}$$

Proof. See Appendix C.2.2. □

Remark 9.4. *The lower and upper bounds in Theorem 9.3 are tight in the sense that there exist permutations σ_1 and σ_2 that satisfy the equality in either lower or upper bound. For equality in lower bound, when $n = 2m$, let*

$$\sigma_1 = [1, 3, 5, \ldots, 2m-3, 2m-1, 2m, 2m-2, \ldots, 6, 4, 2],$$
$$\sigma_2 = [2, 4, 6, \ldots, 2m-2, 2m, 2m-1, 2m-3, \ldots, 5, 3, 1],$$

then $d_\tau(\sigma_1, \sigma_2) = n(n-1)/2$ and $d_{\mathbf{x},\ell_1}(\sigma_1, \sigma_2) = n/2$, as

$$\mathbf{x}_{\sigma_1} = [0, 0, 1, 1, 2, 2, \ldots, m-2, m-2, m-1, m-1],$$
$$\mathbf{x}_{\sigma_2} = [0, 1, 1, 2, 2, 3, \ldots, m-2, m-1, m-1, m].$$

For equality in upper bound, note that $d_\tau(\mathrm{Id}, \sigma) = d_{\mathbf{x},\ell_1}(\mathrm{Id}, \sigma)$.

Theorem 9.3 shows that in general $d_\tau(\sigma_1, \sigma_2)$ is not a good approximation to $d_{\mathbf{x},\ell_1}(\sigma_1, \sigma_2)$ due to the $1/(n-1)$ factor. However, Theorem 9.4 shows that Kendall tau distance scaled by a constant actually provides a lower bound to the inversion-ℓ_1 distance with high probability.

Theorem 9.4. *For any* $\pi \in \mathcal{S}_n$*, let* σ *be a permutation chosen uniformly from* $\mathcal{S}_n$*, then*

$$\mathbb{P}\left[c_2 \cdot d_\tau(\pi, \sigma) \le d_{\mathbf{x},\ell_1}(\pi, \sigma)\right] \ge 1 - O(1/n) \tag{9.12}$$

for any positive constant $c_2 < 1/2$.

Proof. See Appendix C.2.3. □

■ 9.4 Trade-offs between rate and distortion

In this section we present the main results of this chapter—the trade-offs between rate and distortion in permutation spaces. Throughout this section we assume the permutations are uniformly distributed over $\mathcal{S}_n$.

We first present Theorem 9.5, which shows the equivalence of lossy source codes under different distortion measures. This indicates that for all the distortion measures in this chapter, the lossy compression scheme for one measure preserves distortion under other measures under average-case distortion, and hence all compression schemes can be used interchangeably, given appropriate transformation of the permutation representation and scaling the distortion. Then with these equivalence relationships, Theorem 9.6 shows that all distortion measures in this chapter essentially share the same rate distortion function. Last, in Section 9.4.2, we present results on the trade-off between rate and distortion for $\mathcal{X}(\mathcal{S}_n, d_\tau)$ and $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$ when the distortion leads to degenerate rates $R(D_n) = 0$ and $R(D_n) = 1$.

■ 9.4.1 Rate distortion functions

Theorem 9.5 (Equivalence of lossy source codes). *Under* both average-case and worst-case distortion*, a following source code on the left hand side implies a source code on the right hand side:*

1. $(n, D_n/n)$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty}) \Rightarrow (n, D_n)$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$,
2. (n, D_n) *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\ell_1}) \Rightarrow (n, D_n)$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_\tau)$,
3. (n, D_n) *source code for* $\mathcal{X}(\mathcal{S}_n, d_\tau) \Rightarrow (n, 2D_n)$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$,
4. (n, D_n) *source code for* $\mathcal{X}(\mathcal{S}_n, d_\tau) \Rightarrow (n, D_n)$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$.

Furthermore, under average-case distortion*, a following source code on the left hand side implies a source code on the right hand side:*

5. (n, D_n) *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\ell_1}) \Rightarrow (n, D_n/(nc_1) + O(1))$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty})$ *for any* $c_1 < 1/3$,

6. (n, D_n) *source code for* $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1}) \Rightarrow (n, D_n/c_2 + O(n))$ *source code for* $\mathcal{X}(\mathcal{S}_n, d_\tau)$ *for any* $c_2 < 1/2$.

The relationship between source codes is summarized in Fig. 9-1.

The proof is based on the relationships between various distortion measures investigated in Section 9.3 and we defer the details in Appendix C.3.1.

As we show below, for the uniform distribution on $\mathcal{S}_n$, the rate-distortion function is the same for both average- and worst-case, apart from the terms that are asymptotically negligible.

Theorem 9.6 (Rate distortion functions). *For permutation spaces* $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$, $\mathcal{X}(\mathcal{S}_n, d_\tau)$, *and* $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$,

$$\begin{aligned}\bar{R}(D_n) &= \hat{R}(D_n) \\ &= \begin{cases} 1 & \text{if } D_n = O(n), \\ 1-\delta & \text{if } D_n = \Theta\left(n^{1+\delta}\right), \quad 0 < \delta \le 1. \end{cases}\end{aligned} \tag{9.13}$$

For the permutation space $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty})$,

$$\begin{aligned}\bar{R}(D_n) &= \hat{R}(D_n) \\ &= \begin{cases} 1 & \text{if } D_n = O(1), \\ 1-\delta & \text{if } D_n = \Theta\left(n^{\delta}\right), \quad 0 < \delta \le 1. \end{cases}\end{aligned} \tag{9.14}$$

The rate-distortion functions for all these spaces are summarized in Fig. 9-2.

Proof sketch. The achievability comes from the compression schemes proposed in Section 9.5. The converse for $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$ can be shown via the geometry of permutation spaces in Appendix C.1. Then because a D-ball in $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$ has the largest volume (cf. (9.6)), a converse for other permutation spaces can be inferred.

The rest of the proof follows from the simple fact that an achievability scheme for the worst-case distortion is also an achievability scheme for the average-case distortion, and a converse for the average-case distortion is also a converse for the worst-case distortion.

We present the detailed proof in Appendix C.3.2. □

Because the rate distortion functions under average-case and worst-case distortion coincide, if we require

$$\lim_{n\to\infty} \mathbb{P}\left[d(f_n(\sigma), \sigma) > D_n\right] = 0 \tag{9.15}$$

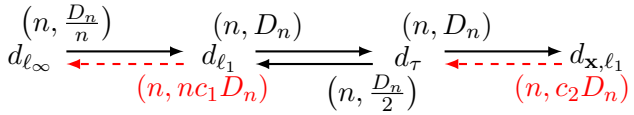


Figure 9-1: Relationship between source codes. An arrow indicates a source code in one space implies a source in another space, where the solid arrow indicates for both average-case and worst-case distortions, and the dashed arrow indicates for average-case only.

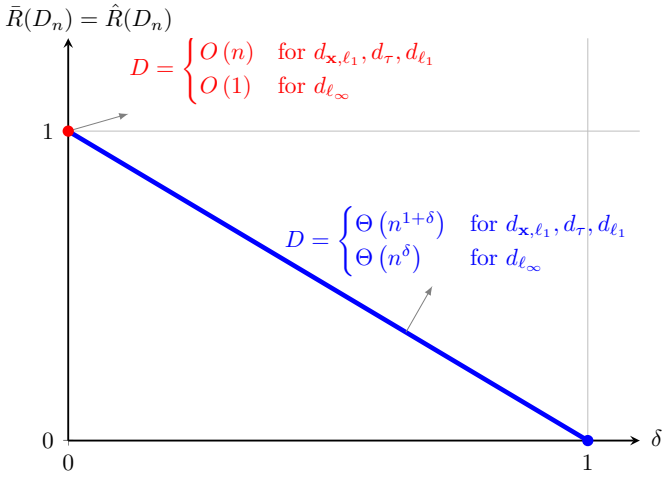


Figure 9-2: Rate-distortion function for permutation spaces $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$, $\mathcal{X}(\mathcal{S}_n, d_\tau)$, $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$, and $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty})$.

instead of $\mathbb{E}[d(f_n(\sigma), \sigma)] \leq D_n$ in Definition 9.6, then the asymptotic rate-distortion trade-off remains the same.

Given the number of elements n and a distortion level D, we can compute the number of bits needed by first computing δ via the asymptotic relationship $\log D / \log n - 1$ (for permutation spaces $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$, $\mathcal{X}(\mathcal{S}_n, d_\tau)$, and $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$) or $\log D / \log n$ (for permutation space $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty})$), then obtain the number of bits needed via $(1-\delta) n \log_2 n$.

■ 9.4.2 Higher order term analysis

As mentioned in Section 9.2, for small- and large-distortion regimes it is of interest to understand the trade-off between rate and distortion via the higher order term defined in (9.5). In this section we present the analysis for both regimes in permutation spaces $\mathcal{X}(\mathcal{S}_n, d_\tau)$ and $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$.

Theorem 9.7. *In the permutation space $\mathcal{X}(\mathcal{S}_n, d_\tau)$, when $D_n = an^\delta, 0 < \delta \leq 1$, for the worst-case distortion, $\underline{r_\tau^{\mathrm{s}}(D_n)} \leq r(D_n) \leq \overline{r_\tau^{\mathrm{s}}(D_n)}$, where*

$$\underline{r_\tau^{\mathrm{s}}(D_n)} = \begin{cases} -a(1-\delta)n^\delta \log n + O\left(n^\delta\right), & 0 < \delta < 1 \\ -n\left[\log \frac{(1+a)^{1+a}}{a^a}\right] + o(n), & \delta = 1 \end{cases}, \tag{9.16}$$

$$\overline{r_\tau^{\mathrm{s}}(D_n)} = \begin{cases} -n^\delta \frac{a \log 2}{2} + O(1), & 0 < a < 1 \\ -n^\delta \frac{\log \lfloor 2a \rfloor !}{\lfloor 2a \rfloor} + O(1), & a \geq 1 \end{cases}. \tag{9.17}$$

When $D_n = bn^2, 0 < b \leq 1/2$, $\underline{r_\tau^{\mathrm{l}}(D_n)} \leq r(D_n) \leq \overline{r_\tau^{\mathrm{l}}(D_n)}$, where

$$\underline{r_\tau^{\mathrm{l}}(D_n)} = \max\left\{0, n \log 1/\left(2be^2\right)\right\}, \tag{9.18}$$

$$\overline{r_\tau^{\mathrm{l}}(D_n)} = n \log \lceil 1/(2b) \rceil + O(\log n). \tag{9.19}$$

Remark 9.5. *Some of the results above for $\mathcal{X}(\mathcal{S}_n, d_\tau)$, since their first appearances in the conference version [113], have been improved subsequently by [114]. More specifically, for the small distortion regime, [114, Lemma 7, Lemma 10] provides an improved upper bound and show that $r^{\mathrm{s}}_\tau(D_n) = \underline{r^{\mathrm{s}}_\tau(D_n)}$ in (9.16). For the large distortion regime, [114, Lemma 11] shows a lower bound that is tighter than (9.18).*

Theorem 9.8. *In the permutation space $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$, when $D_n = an^\delta, 0 < \delta \le 1$,*

$$\underline{r^{\mathrm{s}}_{\mathbf{x},\ell_1}(D_n)} \le r(D_n) \le \overline{r^{\mathrm{s}}_{\mathbf{x},\ell_1}(D_n)},$$

where $\underline{r^{\mathrm{s}}_{\mathbf{x},\ell_1}(D_n)} = \underline{r^{\mathrm{s}}_\tau(D_n)} - n^\delta \log 2$ (cf. (9.16)) and

$$\overline{r_{\mathbf{x},\ell_1}(D_n)} = \begin{cases} -\lfloor n^\delta \rfloor \log(2a-1) & a > 1 \\ -\lceil an^\delta \rceil \log 3 & 0 < a \le 1 \end{cases}.$$

When $D_n = bn^2, 0 < b \le 1/2$,

$$\underline{r^{\mathrm{l}}_{\mathbf{x},\ell_1}(D_n)} \le r(D_n) \le \overline{r^{\mathrm{l}}_{\mathbf{x},\ell_1}(D_n)},$$

where $\underline{r^{\mathrm{l}}_{\mathbf{x},\ell_1}(D_n)} = \underline{r^{\mathrm{l}}_\tau(D_n)}$ (cf. (9.18)) and $\overline{r^{\mathrm{l}}_{\mathbf{x},\ell_1}(D_n)} = n \log \lceil 1/(4b) \rceil + O(1)$.

Proof. The achievability are presented in Section 9.5.4 and Section 9.5.5. For converse, note that

$$|\mathcal{C}_n| N(D_n) \ge n!,$$

where $N(D_n)$ is the maximum of the size of balls with radius D_n in the corresponding permutation space (cf. Appendix C.1 for definitions), then a lower bound on $|\mathcal{C}_n|$ follows from the upper bound on $N(D_n)$ in Lemma C.3 and Lemma C.5. We omit the details as it is analogous to the proof of Theorem 9.6. □

The bounds to $r(D_n)$ of both Kendall tau distance and inversion-ℓ_1 distance in both small and large distortion regimes are shown in Fig. 9-3 and Fig. 9-4.

■ 9.5 Compression schemes

Though the permutation space has a complicated structure, in this section we show two rather straightforward compression schemes, sorting subsequences and component-wise scalar quantization, are optimal as they achieve the rate-distortion functions in Theorem 9.6. We first describe these two key compression schemes in Section 9.5.1 and Section 9.5.2 respectively. Then in Sections 9.5.3 to 9.5.5, we show that by simply applying these building block algorithms with the proper parameters, we can achieve the corresponding trade-offs between rate and distortion shown in Section 9.4.

The equivalence relationships in Theorem 9.5 suggest these two compression schemes achieve the same asymptotic performance. In addition, it is not hard to see that in general sorting subsequences has higher time complexity (e.g., $O(n \log n)$ for moderate distortion regime) than the time complexity of component-wise scalar quantization (e.g.,

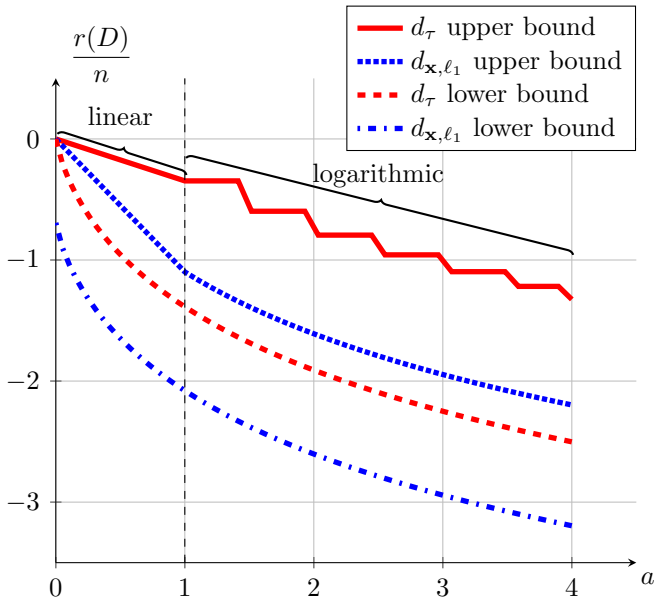


Figure 9-3: Higher-order trade-off between rate and distortion in the small distortion regime with $D = an$. The zig-zag of the d_τ upper bound in the range of $a \geq 1$ is due to the flooring in (9.17).

$O(n)$ for moderate distortion regime). However, these two compression schemes operate on the permutation (or inverse permutation) domain and the inverse vector of permutation domain respectively, and the time complexity to convert a permutation from its vector representation to its inversion vector representation is $\Theta(n \log n)$ [75, Exercise 6 in Section 5.1.1]. Therefore, the cost of representation transformation of permutations should be taken into account when selecting the compression scheme.

■ 9.5.1 Quantization by sorting subsequences

In this section we describe the basic building block for lossy source coding in permutation space $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$, $\mathcal{X}(\mathcal{S}_n, d_{\ell_\infty})$ and $\mathcal{X}(\mathcal{S}_n, d_\tau)$: sorting the subsequences, either of the given permutation σ or of its inverse σ^{-1}. This operation reduces the number of possible permutations and thus the code rate, but introduces distortion. By choosing the proper number of subsequences with proper lengths, we can achieve the corresponding rate-distortion function.

More specifically, we first consider the space $\mathcal{X}(\mathcal{S}_n, d_\tau)$ and a code obtained by the sorting the first k subsequences with length m, $2 \leq m \leq n$, $km \leq n$:

$$\mathcal{C}(k, m, n) \triangleq \{f_{k,m}(\sigma) : \sigma \in \mathcal{S}_n\}$$

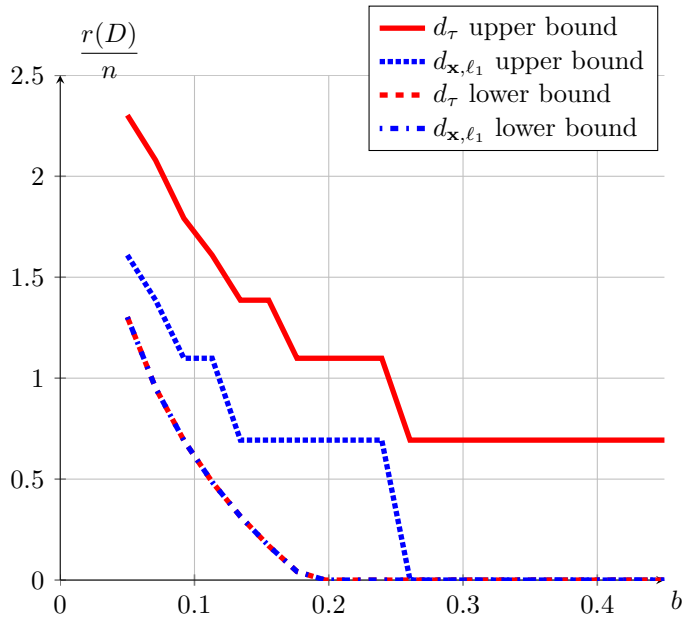


Figure 9-4: Higher-order trade-off between rate and distortion in the large distortion regime with $D = bn^2$. The lower bounds for d_τ and $d_{\mathbf{x},\ell_1}$ are identical.

where $\sigma' = f_{k,m}(\sigma)$ satisfies

$$\begin{aligned} \sigma'[im+1:(i+1)m] &= \operatorname{sort}\left(\sigma[im+1:(i+1)m]\right), && 0 \le i \le k, \\ \sigma'(j) &= \sigma(j), && j > km, \end{aligned}$$

and $\sigma[a:b]$ is a shorthand notation for the vector $[\sigma(a), \sigma(a+1), \ldots, \sigma(b)]$. This procedure is illustrated in Fig. 9-5.

Then $|\mathcal{C}(k,m,n)| = n!/\left(m!^k\right)$, and we define the (log) size reduction as

$$\begin{aligned} \Delta(k,m) &\triangleq \log \frac{n!}{|\mathcal{C}(k,m,n)|} = k \log m! \\ &\overset{(a)}{=} k\left[m\log(m/e) + \frac{1}{2}\log m + O\left(\frac{1}{m}\right)\right], \end{aligned}$$

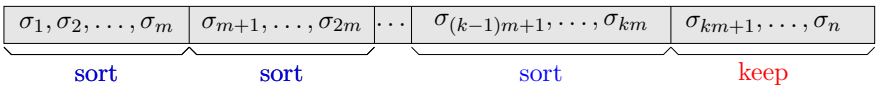


Figure 9-5: Quantization by sorting subsequences.

where (a) follows from Stirling's approximation in (8.1). Therefore,

$$\Delta(k,m) = \begin{cases} km\log m + o\left(km\log m\right) & m = \Omega\left(1\right) \\ k\log m! & m = \Theta\left(1\right) \end{cases}.$$

We then calculate the worst-case and average-case distortions:

$$\hat{D}_{d_\tau}\left(k,m\right) = k\frac{m(m-1)}{2} \leq km^2/2 \tag{9.20}$$

$$\bar{D}_{d_\tau}\left(k,m\right) = k\frac{m(m-1)}{4} \leq km^2/4 \tag{9.21}$$

where (9.20) is from (C.11).

Similarly, for permutation space $\mathcal{X}\left(\mathcal{S}_n, d_{\ell_1}\right)$ and $\mathcal{X}\left(\mathcal{S}_n, d_{\ell_\infty}\right)$, we consider sorting subsequences in the *inverse permutation* domain, where

$$\mathcal{C}'(k,m,n) \triangleq \left\{\pi^{-1} : \pi = f_{k,m}(\sigma^{-1}), \sigma \in \mathcal{S}_n\right\}.$$

It is straightforward that $\mathcal{C}'(k,m,n)$ has the same cardinality as $\mathcal{C}(k,m,n)$ and hence code rate reduction $\Delta(k,m)$. And the worst-case and average-case distortions satisfy

$$\hat{D}_{\ell_\infty}\left(k,m\right) = m-1 \tag{9.22}$$

$$\bar{D}_{\ell_\infty}\left(k,m\right) \leq m-1 \tag{9.23}$$

$$\hat{D}_{\ell_1}\left(k,m\right) = k\left\lfloor m^2\right\rfloor/2 \leq km^2/2 \tag{9.24}$$

$$\bar{D}_{\ell_1}\left(k,m\right) = k(m^2-1)/3, \tag{9.25}$$

where (9.24) comes from Lemma C.6 and (9.25) comes from (C.10).

Remark 9.6. *Due to the close relationship between Kendall tau distance and Spearman's footrule shown in (9.10), there exists an equivalent construction via the inverse permutation σ^{-1} of a permutation $\sigma \in \mathcal{S}_n$:*

1. *Construct a vector $a(\sigma)$ such that for $1 \leq i \leq k$,*

$$a(i) = j \text{ if } \sigma^{-1}(i) \in [(j-1)m+1, jm], 1 \leq j \leq k.$$

   Then a contains exactly m values of integers j.

2. *Form a permutation π' by replacing the length-m subsequence of a that corresponds to value j by vector $[(j-1)m+1, (j-1)m+2, \ldots, jm]$.*

It is not hard to see that the set of $\left\{\pi'^{-1}\right\}$ forms a codebook with the same size with distortion upper bounded by $km^2/2$.

■ 9.5.2 Component-wise scalar quantization

To compress in the space of $\mathcal{X}\left(\mathcal{S}_n, d_{\mathbf{x},\ell_1}\right)$, component-wise scalar quantization suffices, due to the product structure of inversion vector space $\mathcal{G}_n$.

More specifically, to quantize the k points in $[0:k-1]$, where $k=2,\cdots,n$, uniformly with m points, the maximal distortion is

$$\hat{D}_{\mathbf{x},\ell_1}(k,m)=\lceil (k/m-1)/2 \rceil, \tag{9.26}$$

Conversely, to achieve distortion $\hat{D}_{\mathbf{x},\ell_1}$ on $[0:k-1]$, we need

$$m=\left\lceil k/\left(2\hat{D}_{\mathbf{x},\ell_1}+1\right)\right\rceil \tag{9.27}$$

points.

■ 9.5.3 Compression in the moderate distortion regime

In this section we provide compression schemes in the moderate distortion regime, where for any $0<\delta<1$, $D_n=\Theta\left(n^{\delta}\right)$ for $\mathcal{X}(\mathcal{S}_n,d_{\ell_\infty})$ and $D_n=\Theta\left(n^{1+\delta}\right)$ for $\mathcal{X}(\mathcal{S}_n,d_{\ell_1})$, $\mathcal{X}(\mathcal{S}_n,d_{\tau})$ and $\mathcal{X}(\mathcal{S}_n,d_{\mathbf{x},\ell_1})$. While Theorem 9.5 indicates a source code for $\mathcal{X}(\mathcal{S}_n,d_{\ell_\infty})$ can be transformed into source codes for other spaces under both average-case and worst-case distortions, we develop explicit compression schemes for each permutation spaces as the transformation of permutation representations incur additional computational complexity and hence may not be desirable.

Permutation space $\mathcal{X}(\mathcal{S}_n,d_{\ell_\infty})$

Given distortion $D_n=\Theta\left(n^{\delta}\right)$, we apply the sorting subsequences scheme in Section 9.5.1 and choose $m=D_n+1$, which ensures the maximal distortion is no more than D_n, and $k=\lfloor n/m \rfloor$, which indicates

$$\begin{aligned} km=\lfloor n/m\rfloor m &= n+O\left(n^{\delta}\right)\\ \log m &= \delta\log n+o(1)\\ \Delta(k,m) &= km\log m+o(km\log m)\\ &= \delta n\log n+O(n). \end{aligned}$$

Permutation spaces $\mathcal{X}(\mathcal{S}_n,d_{\ell_1})$ and $\mathcal{X}(\mathcal{S}_n,d_{\tau})$

Given distortion $D_n=\Theta\left(n^{1+\delta}\right)$, we apply the sorting subsequences scheme in Section 9.5.1 and choose

$$\begin{aligned} m &= (1/\alpha)\lfloor D_n/n\rfloor \le D_n/(n\alpha)\\ k &= \lfloor n/m\rfloor, \end{aligned}$$

then

$$\begin{aligned} km &= n-\left|O\left(n^{\delta}\right)\right|\\ D &\le \alpha km^2\le D_n\\ \Delta(k,m) &= \delta n\log n-n\log(\alpha e)+o(n), \end{aligned}$$

where the constant α depends on the distortion measure and whether we are considering worst-case or average-case distortion, as shown in (9.20) and (9.21) and (9.24) and (9.25), and is summarized in Table 9.1.

Permutation space $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$

Given distortion $D_n = \Theta\left(n^{1+\delta}\right)$, we apply the component-wise scalar quantization scheme in Section 9.5.2 and choose the quantization error of the coordinate with range $[0 : k-1]$ $D^{(k)}$ to be

$$D^{(k)} = \frac{kD}{(n+1)^2},$$

then

$$\begin{aligned} m_k &= \left\lceil k/\left(2 + D^{(k)} + 1\right)\right\rceil = \left\lceil \frac{k(n+1)^2}{2kD_n + (n+1)^2}\right\rceil \\ &\le \left\lceil \frac{(n+1)^2}{2D_n}\right\rceil, \end{aligned}$$

and the overall distortion and the codebook size satisfy

$$\begin{aligned} D &= \sum_{k=2}^{n} = \frac{(n-1)(n+2)}{(n+1)^2} D_n \le D_n, \\ \log|C_n| &= \sum_{k=2}^{n} \log m_k \le n \log\left\lceil \frac{(n+2)^2}{2D_n}\right\rceil \\ &= (1-\delta) n \log n + O(n). \end{aligned}$$

■ 9.5.4 Compression in the small distortion regime

In this section we provide compression schemes in the small distortion regime for $\mathcal{X}(\mathcal{S}_n, d_\tau)$ and $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$, where for any $0 < \delta < 1$, $D_n = an^\delta$.

Permutation space $\mathcal{X}(\mathcal{S}_n, d_\tau)$

When $a \ge 1$, let $m = \lfloor 2a \rfloor$ and $k = \left\lfloor n^\delta/m \right\rfloor$, then

$$\begin{aligned} \Delta(k,m) &= k \log m! \\ &\ge (n^\delta/m - 1)\log m! = \frac{\log \lfloor 2a \rfloor!}{\lfloor 2a \rfloor} n^\delta + O(1). \end{aligned}$$

α	average-case	worst-case
$\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$	1/3	1/2
$\mathcal{X}(\mathcal{S}_n, d_\tau)$	1/4	1/2

Table 9.1: The values of α for different compression scenarios.

And the worst-case distortion is upper bounded by

$$km^2/2 \leq \frac{n^\delta m}{2} \leq an^\delta = D_n.$$

When $0 < a < 1$, let $m = 2$ and $k = \lfloor D_n/2 \rfloor$, then

$$\Delta(k, m) = k \log m! = \left\lfloor \frac{D_n}{2} \right\rfloor \log 2 = \frac{a \log 2}{2} n^\delta + O(1).$$

And the worst-case distortion is no more than $km^2/2 \leq D_n$.

Permutation space $\mathcal{X}\left(\mathcal{S}_n, d_{\mathbf{x},\ell_1}\right)$

When $a > 1$, let

$$m_k = \begin{cases} k & k \leq n - \lfloor n^\delta \rfloor \\ \lceil k/(2a-1) \rceil & k > n - \lfloor n^\delta \rfloor \end{cases}, \quad k = 2, \ldots, n$$

then the distortion $D^{(k)}$ for each coordinate k satisfies

$$D^{(k)} \leq \begin{cases} a & k \leq \lceil n^\delta \rceil \\ 0 & k > \lceil n^\delta \rceil \end{cases}, k = 2, 3, \ldots, n,$$

and hence overall distortion is $\sum_{k=2}^{n} D^{(k)} = (\lfloor n^\delta \rfloor)a \leq D_n$. In addition, the codebook size

$$\left|\hat{\mathcal{C}}_n\right| = \prod_{k=2}^{n} m_k \leq (1/(2a-1))^{\lfloor n^\delta \rfloor} \prod_{k=2}^{n} k.$$

Therefore, $\log \left|\hat{\mathcal{C}}_n\right| \leq \log n! - \lfloor n^\delta \rfloor \log(2a-1) + O(\log n)$.

When $a \leq 1$, let

$$m_k = \begin{cases} \lceil k/3 \rceil & k < \lceil D_n \rceil \\ k & k \geq \lceil D_n \rceil \end{cases}, \quad k = 2, \ldots, n$$

and apply uniform quantization on the coordinate k of the inversion vector with m_k points, Then the distortion $D^{(k)}$ for each coordinate k satisfies

$$D^{(k)} \leq \begin{cases} 1 & k < \lceil D_n \rceil \\ 0 & k \geq \lceil D_n \rceil \end{cases}, k = 2, 3, \ldots, n,$$

and hence overall distortion is $\sum_{k=2}^{n} D^{(k)} = \lceil D_n \rceil - 1 \leq D_n$. In addition, the codebook size

$$\begin{aligned}\left|\hat{\mathcal{C}}_n\right| &= \prod_{k=2}^{n} m_k \leq \prod_{k=2}^{\lceil D_n \rceil - 1} (k+3)/3 \prod_{k=\lceil D_n \rceil}^{n} k \\ &= \frac{1}{3^{\lceil D_n \rceil - 1}} \lceil D_n \rceil \left(\lceil D_n \rceil + 1\right)\left(\lceil D_n \rceil + 2\right) \prod_{k=5}^{n-1} k.\end{aligned}$$

Therefore, $\log \left|\hat{\mathcal{C}}_n\right| \leq \log n! - \left\lceil a n^{\delta} \right\rceil \log 3 + O\left(\log n\right)$.

■ 9.5.5 Compression in the large distortion regime

In this section we provide compression schemes in the small distortion regime for $\mathcal{X}\left(\mathcal{S}_n, d_\tau\right)$ and $\mathcal{X}\left(\mathcal{S}_n, d_{\mathbf{x},\ell_1}\right)$, where for any $0 < \delta < 1$, $D_n = bn^2$.

Permutation space $\mathcal{X}\left(\mathcal{S}_n, d_\tau\right)$

Let $k = \lceil 1/(2b) \rceil$ and $m = \lfloor n/k \rfloor$, then

$$\begin{aligned}\Delta(k,m) &= k \log m! \geq k \log(n/k-1)! \\ &\geq k[n/k \log(n/k) - n/k \log e + O\left(\log n\right)] \\ &= n \log(n/e) - n \log \lceil 1/(2b) \rceil + O\left(\log n\right).\end{aligned}$$

Hence $\hat{r}(D_n) = \log n! - \Delta(k,m) \leq \log \lceil 1/(2b) \rceil + O\left(\log n\right)$. And the worst-case distortion is upper bounded by

$$km^2/2 \leq n^2/(2k) \leq n^2/(1/b) = bn^2.$$

Permutation space $\mathcal{X}\left(\mathcal{S}_n, d_{\mathbf{x},\ell_1}\right)$

Let $m_k = \lceil k/(4b(k-1)+1) \rceil, \quad k = 2, \ldots, n$. The distortion $D^{(k)}$ for each coordinate k satisfies

$$D^{(k)} = \left\lceil \frac{1}{2}\left(\frac{k}{m} - 1\right) \right\rceil \leq \lceil 2b(k-1) \rceil, k = 2, 3, \ldots, n,$$

and hence overall distortion $\sum_{k=2}^{n} D^{(k)} \leq \sum_{k=2}^{n} 2b(k-1) + 1 \leq (b + 1/n)n(n-1)$. In addition, the codebook size

$$\left|\hat{\mathcal{C}}_n\right| = \prod_{k=2}^{n} m_k \leq \prod_{k=2}^{n} \left\lceil \frac{k-1}{4b(k-1)} \right\rceil \leq \left\lceil \frac{1}{4b} \right\rceil^{n-1}.$$

Therefore, $\log \left|\hat{\mathcal{C}}_n\right| \leq n \log \lceil 1/(4b) \rceil + O\left(1\right)$.

■ 9.6 Compression of permutation space with Mallows model

In this section we depart from the uniform distribution assumption and investigate the compression of a permutation space with a distributional model—Mallows model [92], a

distributional model with a wide range of applications such ranking, partial ranking, and even algorithm analysis (see [115, Section 2e] and the references therein). In the context of storing user ranking data, Mallows model (or more generally, the mixture of Mallows model) captures the phenomenon that user rankings are often similar to each other. In the application of approximate sorting, Mallows model may be used to model our prior knowledge that permutations that are similar to the reference permutation are more likely.

Definition 9.9 (Mallows model). *We denote a* Mallows model *with reference permutation (mode)* π *and parameter* q *as* $\mathcal{M}(\pi, q)$*, where for each permutation* $\sigma \in \mathcal{S}_n$*,*

$$\mathbb{P}[\sigma; \mathcal{M}(\pi, q)] = \frac{q^{d_\tau(\sigma,\pi)}}{Z_q},$$

where normalization $Z_q = \sum_{\sigma\in\mathcal{S}_n} p^{d_\tau(\sigma,\pi)}$*. In particular, when the mode* $\pi = \mathrm{Id}$*,* $Z_q = [n]_p!$ *[115, (2.9)], where* $[n]_q!$ *is the q-factorial* $[n]_q! = [n]_q[n-1]_q \dots [1]_q$ *and* $[n]_q$ *is the q-number*

$$[n]_q \triangleq \begin{cases} \frac{1-q^n}{1-q} & q \neq 1 \\ n & q = 1 \end{cases}.$$

As we shall we, the entropy of the permutation space with Mallows model is in general $\Theta(n)$, implying lower space for storage and *potentially* lower query complexity for sorting. Since the Mallows model is specified via the Kendall tau distance, we use Kendall tau distance as the distortion measure, and focus our attention to average-case distortion.

Noting the Kendall tau distance is right-invariant [111], for the purpose of compression, without loss of generality, we can assume the mode $\pi = \mathrm{Id}$, and denote the Mallows model by $\mathcal{M}(q) \triangleq \mathcal{M}(\mathrm{Id}, q)$.

■ 9.6.1 Repeated insertion model

The Mallows model can be generated through a process named *repeated insertion model* (RIM), which is introduced in [116] and later applied in [93].

Definition 9.10 (Repeated insertion model). *Given a reference permutation* $\pi \in \mathcal{S}_n$ *and a set of insertion probabilities* $\{p_{i,j}, 1 \le i \le n, 1 \le j \le i\}$*, RIM generates a new output* σ *by repeated inserting* $\pi(i)$ *before the* j*-th element in* σ *with probability* $p_{i,j}$ *(when* $j = i$*, we append* $\pi(i)$ *at the end of* σ*).*

Remark 9.7. *Note that the insertion probabilities at step* i *is independent of the realization of earlier insertions.*

If we denote the sampling at each step of the RIM process by $a_i, 1 \le i \le n$, then a vector $\mathbf{a} = [a_1, a_2, \dots, a_n]$ has an one-one correspondence to a permutation, and we called this vector $\mathbf{a}$ an *insertion vector*.

Lemma 9.9. *Given a RIM with reference permutation* $\pi = \mathrm{Id}$ *and insertion vector* $\mathbf{a}_\sigma$*, then the corresponding permutation* σ *satisfies*

$$\mathbf{a}_\sigma(i) = i - \tilde{\mathbf{x}}_\sigma(i),$$

where $\tilde{\mathbf{x}}_\sigma$ *is an* extended inversion vector *that simply prepends a 0 to an inversion vector* $\mathbf{x}_\sigma$*:*

$$\tilde{\mathbf{x}}_\sigma(i) = \begin{cases} 0 & i = 1 \\ \mathbf{x}_\sigma(i-1) & 2 \le i \le n \end{cases}$$

Therefore,

$$\begin{aligned} d_\tau(\sigma, \mathrm{Id}) &= d_{\mathbf{x},\ell_1}(\sigma, \mathrm{Id}) \\ &= \sum_{i=1}^{n} (i - \mathbf{a}_\sigma(i)) = \binom{n+1}{2} - \sum_{i=1}^{n} \mathbf{a}_\sigma(i). \end{aligned}$$

Example 9.4. *For* $n = 4$ *and reference permutation* $\mathrm{Id} = [1,2,3,4]$*, if* $\mathbf{a} = [1,1,1,1]$*, then* $\sigma = [4,3,2,1]$*, which corresponds to* $\tilde{\mathbf{x}}_\sigma = [0,1,2,3]$*.*

Theorem 9.10 (Mallows model via RIM [93, 116]). *Given reference permutation* π *and*

$$p_{i,j} = \frac{q^{i-j}}{1 + q + \ldots + q^{i-1}}, 1 \le j \le i \le n,$$

RIM induces the same distribution as the Mallows model $\mathcal{M}(\pi, q)$*.*

This observation allows us to convert compressing the Mallows model to a standard problem in source coding.

Theorem 9.11. *Compressing a Mallows model is equivalent to compressing a vector source* $\mathbf{X} = [X_1, X_2, \ldots, X_n]$*, where* X_i *is a geometric random variable truncated at* $i-1, 1 \le i \le n$*, i.e.,*

$$\begin{aligned} \mathbb{P}[X_i = j] &= \frac{q^j}{\sum_{j'=0}^{i-1} q^{j'}} \\ &= \frac{q^j(1-q)}{1-q^i}, 0 \le j \le i-1 \end{aligned}$$

Proof. This follows directly from Lemma 9.9 and Theorem 9.10. □

■ 9.6.2 Lossless compression

We consider the lossless compression of Mallows model.

Lemma 9.12.

$$H(\mathcal{M}(q)) = H(\mathcal{M}(1/q))$$

Proof. This follows directly from Theorem 9.10. □

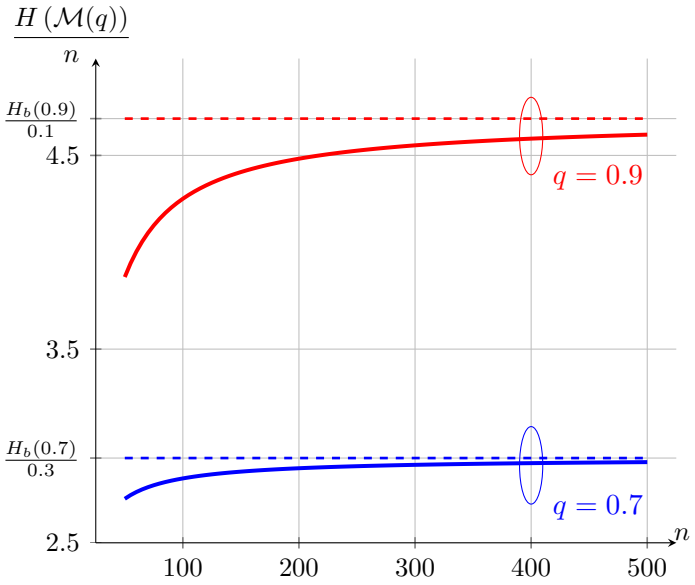


Figure 9-6: Entropy of the Mallows model for $q = 0.7$ and $q = 0.9$, where the dashed lines are the coefficients of the linear terms, $H_b(q)/(1-q)$.

Lemma 9.13 (Entropy of Mallows model).

$$H(\mathcal{M}(q)) = \sum_{k=1}^{n} H(X_k) = \begin{cases} \frac{H_b(q)}{1-q} n + g(n,q) & q \neq 1 \\ \log n! & q = 1 \end{cases},$$

where $H_b(\cdot)$ is the binary entropy function, $g(n,q) = \Theta(1)$, and $\lim_{q\to 0} g(n,q) = 0$.

The proof is presented in Appendix C.4.1. Fig. 9-6 shows plots of $H(\mathcal{M}(q))$ for different values of n and q.

Remark 9.8. *Doing entropy-coding for each $X_i, 1 \leq i \leq n$ is sub-optimal in general as the overhead is $O(1)$ for each i and hence $O(n)$ for $\mathbf{X}$, which is on the same order of the entropy $H(\mathcal{M}(q))$ when $q \neq 1$.*

■ 9.6.3 Lossy compression

By Theorem 9.11, the lossy compression of Mallows model is equivalent to the lossy compression of the independent non-identical source $\mathbf{X}$. However, it is unclear whether an analytical solution of the rate-distortion function for this source can be derived, and below we try to gain some insights via characterizing the typical set of Mallows model in Lemma 9.14, which implies that at rate 0, the average-case distortion is $\Theta(n)$, while under the uniform distribution, Theorem 9.6 indicates that it takes $n \log n + o(n \log n)$ bits to achieve average-case distortion of $\Theta(n)$!

Lemma 9.14 (Typical set of Mallows model). *There exists $c_0(q)$, a constant that depends on q only, such that for any $r_0 \geq c_0(q)n$,*

$$\lim_{n\to\infty} \mathbb{P}\left[B_\tau\left(r_0\right)\right] = 1.$$

The proof is presented in Appendix C.4.2.

Remark 9.9. *As pointed out in [116], Mallows model is only one specific distributional model that is induced by RIM. It is possible to generalize our analysis above to other distributional models that is also induced by RIM.*

■ 9.7 On the lower bound of query complexity in approximate sorting

In this section we discuss the implications of our rate-distortion results on the lower bound of query complexity of approximate sorting.

■ 9.7.1 Uniform distributional model

For the case of noiseless comparisons, since each comparison provides at most 1 bit of information, Theorem 9.5 indicates that to achieve a ranking distortion of $\Theta\left(n^{1+\delta}\right)$ in terms of Spearman's footrule, Kendall tau distance inversion-ℓ_1 distance, or $\Theta\left(n^{\delta}\right)$ in terms of Chebyshev distance, at least $(1-\delta)n\log n$ noiseless comparisons are needed.

For the case of noisy comparisons with error probability ε, as each noisy comparison provides at most $1-H_b\left(\varepsilon\right)$ bit of information, at least $(1-\delta)/(1-H_b\left(\varepsilon\right))n\log n$ comparisons are needed.

In both cases, our lower bound results indicate that for approximate sorting, the constant in front of the $n\log n$ term matters.

For the case of noiseless comparisons, it is not hard to see that the *multiple selection* algorithm proposed in [83] achieves the above query complexity-distortion trade-off for all distortion measures of interest, and hence our information-theoretic bound is tight! It is worth noting that while [82] indeed provides a lower bound on the query complexity for the *multiple selection* problem, our results is stronger because we show a lower bound on the query complexity for *all algorithms* that achieve a certain distortion level, while multiple selection is only one particular class of algorithms that achieves the distortion level.

For the case of noisy comparisons, we can specialize the noisy-comparison based algorithms Section 8.3 to obtain upper bounds. Letting $q = n^{\delta-1}$, the results in [89] indicate that $\Theta\left((2-\delta)n\log n\right)$ comparisons are sufficient to achieve distortion $O\left(n^{1+\delta}\right)$ in terms of Spearman's footrule, Kendall tau distance or inversion-ℓ_1 distance. The algorithms in [90, 91] correspond to the small distortion regime $D = O\left(n\right)$. However, since none of these algorithms analyze the constant in front of the $n\log n$ term, it is unclear if they achieve the information theoretic lower-bound and hence the tightness question of the information-theoretic lower bound in the presence of noisy comparisons remains open.

■ 9.7.2 Mallows distributional model

For the Mallows model, our entropy calculation in Lemma 9.13 shows that when $q \neq 1$, even for regular (non-approximate) sorting, information theory suggests a lower bound

of $nH_b(q)/(1-q) + \Theta(1)$ comparisons, which is much lower than the case of uniform distribution.

Lemma 9.14 reveals that under the Mallows distribution, any permutations that has super-linear Kendall tau distance to the mode is highly unlikely, and can be ignored for the purpose of distortion calculation. Therefore, when sorting under the Mallows model, we have a much smaller space to search for the true permutation, hinting there may be a sorting algorithm with query complexity as low as $nH_b(q)/(1-q)$.

■ 9.8 Concluding remarks

Entropy-based information-theoretic lower bound has been one useful tool in computational complexity analysis, and our analysis in this chapter demonstrates that besides entropy, rate-distortion theory is also useful in analyzing the computational complexity for algorithms, especially approximate algorithms. In particular, our lower bound for the query complexity of approximate sorting with noiseless comparisons is more general than existing lower bounds and is tight.

While the optimal algorithm for approximate sorting with noiseless comparisons is known, constant-optimal or near constant-optimal algorithms for approximate sorting with noisy comparisons are still open, and is of great practical interest in the context of crowd-based ranking, as discussed in Section 8.2. This will also help to understand whether the information theoretic lower bound we developed is tight.

In addition, our results on sorting with the Mallows model indicate that the search space for the true permutation is much smaller than the uniform case, and hence a lower query complexity may be achievable. While there exists algorithms [93] for sorting with the Mallows model, it is not easily amenable to query complexity analysis, and it is of great interest to investigate sorting algorithms under the Mallows or other common distributional models.

Finally, as distributional models play an important role in many areas, such as learning to rank [93] and algorithm analysis [115], a deeper understanding on the rate-distortion trade-off of non-uniform distributional models would be beneficial.

Chapter 10

Concluding remarks

While the subject of computing with unreliable resources have been researched in various contexts, it has some unique characteristics nowadays due to the large scale of modern computing systems, both posing new challenges and presenting new opportunities. In this thesis, we investigate the problem of computing with unreliable resources in three emerging applications, reliable circuit design, scheduling parallel tasks, and crowd-based ranking. For each of this application, it is necessary to introduce redundancy to obtain reliable results, and we investigate how to apply redundancy efficiently so that the extra resource usage does not outweigh the gain in reliability. To tackle this, we propose analytical frameworks that aim to capture the essence of the systems of interest, and derive performance trade-offs accordingly. In particular, we show that by taking the statistical property of unreliability into account, we can introduce redundancy efficiently to increase fabrication yield, reduce computation latency and improve sorting accuracy.

Besides the three applications in this thesis, the problem of computing with unreliable resources occurs in many other settings that frameworks proposed in this thesis can be useful. For example, our model of digital circuit design with faulty components can be generalized to model a redundant system with redundancy sharing constraints, and this occur in other contexts such as logistics planning. For another example, our framework for scheduling parallel tasks can be used to analyze more applications that involve distributed computation, such as clinical trials and circuit timing analysis. In clinical trials, drugs are provided to subjects (animals or human beings) for effectiveness test, and the response time for each subject is stochastic. It is of great economic interest to understand the trade-off between the latency of a trial and its cost. In another instance, we can view a circuit as a special case of distributed computing, where each computing node is a logic unit, such as AND gate and OR gate. Then the delay of each gate is the execution time of each computing node, and analyze the overall latency of the circuit is the *timing closure* problem in statistical timing analysis [117].

In addition to including more applications, it is worthwhile to connect the problem of computing with unreliable resources to related research areas. For example, with unreliable resources, it may be impossible to obtain exact processing and we have to settle with approximate answers. Therefore, research areas such as *approximate computing* and *approximate signal processing* [118] may offer some meaningful connections.

In summary, computing with unreliable resources is a rich subject, and there are still many applications to explore and many questions to answer. We hope this thesis provides some first steps towards an overarching theory on building reliable and efficient large scale computing systems with unreliable resources!

A journey of a thousand miles begins with a single step.

Lao Tzu

Appendices

Appendix A

Derivations and proofs for Part I

■ A.1 Results regarding the performance metrics of ADC

Lemma A.1. *For an ADC with evenly spaced reproduction values, i.e.,* $c_i = (i-1/2)v_{\mathrm{LSB}} + v_{\mathrm{lo}}, 1 \le i \le n+1$*, then the maximum quantization error* $e_{\max}$*, the maximum difference between an input value* v_{in} *and its reproduction value, satisfies*

$$e_{\max} \le (\mathrm{INL} + v_{\mathrm{LSB}}/2)\,. \tag{A.1}$$

In particular, when $v_1 \ge v_{\mathrm{lo}} + v_{\mathrm{LSB}}$ *or* $v_n \le v_{\mathrm{lo}} + (n-1)v_{\mathrm{LSB}}$*,*

$$e_{\max} = (\mathrm{INL} + v_{\mathrm{LSB}}/2)\,. \tag{A.2}$$

Proof. Without loss of generality, assume $v_{\mathrm{lo}} = 0$, and then $c_i = (i-1/2)v_{\mathrm{LSB}}, 1 \le i \le n+1$. Note that

$$\begin{aligned} e_{\max} &= \max_{1\le i\le n+1} \max\left\{|v_i - c_i|\,, |c_i - v_{i-1}|\right\} \\ &= \max\left\{ \max_{1\le i\le n+1} |v_i - c_i|\,, \max_{1\le i\le n+1} |c_i - v_{i-1}| \right\}, \end{aligned}$$

where

$$\begin{aligned} \max_{1\le i\le n+1} |v_i - c_i| &= \max_{1\le i\le n+1} |v_i - i\cdot v_{\mathrm{LSB}} + 0.5v_{\mathrm{LSB}}|\,, \\ &\le \max_{1\le i\le n+1} |v_i - i\cdot v_{\mathrm{LSB}}| + 0.5v_{\mathrm{LSB}}, \\ &= \mathrm{INL} + 0.5v_{\mathrm{LSB}} \end{aligned}$$

and similarly,

$$\begin{aligned} \max_{1\le i\le n+1} |c_i - v_{i-1}| &\le \max_{1\le i\le n+1} |v_{i-1} - (i-1)v_{\mathrm{LSB}}| + 0.5v_{\mathrm{LSB}} \\ &= \mathrm{INL} + 0.5v_{\mathrm{LSB}}. \end{aligned}$$

Therefore,

$$e_{\max} \le \mathrm{INL} + 0.5v_{\mathrm{LSB}}.$$

To show (A.2), define

$$\begin{aligned} \mathrm{INL}_i &\triangleq |v_i - i \cdot v_{\mathrm{LSB}}|, \quad , 1 \le i \le n \\ i^* &\in \arg\max_{1 \le i \le n} \mathrm{INL}_i. \end{aligned}$$

then $\mathrm{INL} = \mathrm{INL}_{i^*}$. Note that $v_1 \ge v_{\mathrm{lo}} + v_{\mathrm{LSB}}$ or $v_n \le v_{\mathrm{lo}} + (n-1)v_{\mathrm{LSB}}$ means $e_{\max} \ge v_{\mathrm{LSB}}/2$, and hence

$$\begin{aligned} e_{\max} &= \max\left\{ \max_{1 \le i \le n} |v_i - c_i|, \max_{2 \le i \le n+1} |c_i - v_{i-1}| \right\} \\ &= \max\left\{ \max_{1 \le i \le n} |v_i - c_i|, \max_{1 \le i \le n} |c_{i+1} - v_i| \right\}. \end{aligned}$$

Also, for $1 \le i^* \le n$,

$$\begin{aligned} &\text{if } v_{i^*} - i^* \cdot v_{\mathrm{LSB}} \ge 0, \text{ then } |v_{i^*} - c_{i^*}| = \mathrm{INL} + 0.5 v_{\mathrm{LSB}}, \\ &\text{if } v_{i^*} - i^* \cdot v_{\mathrm{LSB}} < 0, \text{ then } |c_{i^*+1} - v_{i^*}| = \mathrm{INL} + 0.5 v_{\mathrm{LSB}}. \end{aligned}$$

Therefore,

$$e_{\max} \ge \max\left\{ |v_{i^*} - c_{i^*}|, |c_{i^*+1} - v_{i^*}| \right\} \ge \mathrm{INL} + 0.5 v_{\mathrm{LSB}},$$

which completes the proof for (A.2). □

A.2 Derivations for high resolution analysis

A.2.1 High resolution analysis of MSE

In this section we first show a result for the MSE for a quantizer with random uniformly distributed partition points in Lemma A.2, and its extension in Lemma A.3, then proceed to show the high resolution approximation result in (3.15) by showing the increase in MSE (comparing to (3.16)) is due to the random interval sizes resulting from the random partitioning, rather than the random number of partition points in an interval.

Lemma A.2 (Theorem 1 in [20]). *Given* $X \sim \mathit{Unif}([0, \Delta])$ *and* $W_i \overset{i.i.d.}{\sim} \mathit{Unif}([0, \Delta])$, $1 \le i \le n$, *then*

$$\mathbb{E}_{X, W^n}\left[d\left(X, W^n\right) \right] = \frac{\Delta^2}{2(n+2)(n+3)}$$

Proof. See the proof in [20]. □

Lemma A.3. *Given* $X \sim \mathit{Unif}([0, \Delta])$, *and for* $1 \le i \le n$,

$$\begin{aligned} W_i &\sim \mathit{Unif}([0, \Delta]) \quad w.p.\ p_i \\ W_i &\notin [0, \Delta] \quad w.p.\ 1 - p_i, \end{aligned}$$

and let $k_n = \sum_{i=1}^n p_i$*, then if for some* $\varepsilon > 0$*,*

$$\lim_{n\to\infty} \frac{k_n}{n^{1/2+\varepsilon}} = c > 0,$$

then

$$\lim_{n\to\infty} k_n^2 \, \mathbb{E}_{X,W^n}\left[d(X, W^n)\right] = \frac{\Delta^2}{2}.$$

Proof. Define $U_i \triangleq \mathbb{1}\left\{W_i \sim \mathsf{Unif}\left([0,\Delta]\right)\right\}$, then $U_i \sim \mathsf{Bern}\left(p_i\right)$. Let $K \triangleq \sum_{i=1}^n U_i$, we have

$$\mathbb{E}_{X,W^n}\left[d\left(X, W^n\right)\right] = \sum_{k=0}^{n} \mathbb{E}_{X,W^n}\left[d\left(X, W^n\right) \middle| K = k\right] \mathbb{P}\left[K = k\right].$$

Lemma A.2 indicates that

$$\mathbb{E}_{X,W^n}\left[d\left(X, W^n\right) \middle| K = k\right] = \frac{\Delta^2}{2(k+2)(k+3)}.$$

Noting K is the sum of n independent Bernoulli random variables, by Hoeffding's inequality,

$$\mathbb{P}\left[\left|K - \mathbb{E}\left[K\right]\right| > t\right] \leq 2\exp\left(-\frac{2t^2}{n}\right).$$

Let $t_n = n^{1/2+\varepsilon/2}$, then

$$\mathbb{P}\left[\left|K - \mathbb{E}\left[K\right]\right| > t_n\right] \leq 2\exp\left(-2n^{\varepsilon}\right).$$

Therefore, since $\lim_{n\to\infty} t_n/k_n = 0$,

$$\begin{aligned}
&\lim_{n\to\infty} k_n^2 \sum_{k\in[k_n - t_n, k_n + t_n]} \mathbb{E}_{X,W^n}\left[d\left(X, W^n\right) \middle| K = k\right] \mathbb{P}\left[K = k\right] \\
\leq &\lim_{n\to\infty} \frac{k_n^2 \Delta^2}{2(k_n - t_n + 2)(k_n - t_n + 3)} \left[1 - 2\exp\left(-2n^{\varepsilon}\right)\right] = \Delta^2/2, \\
&\lim_{n\to\infty} k_n^2 \sum_{k\notin[k_n - t_n, k_n + t_n]} \mathbb{E}_{X,W^n}\left[d\left(X, W^n\right) \middle| K = k\right] \mathbb{P}\left[K = k\right] \\
\leq &\lim_{n\to\infty} \Delta^2/12 \cdot \mathbb{P}\left[K \notin [k_n - t_n, k_n + t_n]\right] = 0
\end{aligned}$$

and hence

$$\lim_{n\to\infty} k_n^2 \, \mathbb{E}_{X,W^n}\left[d\left(X, W^n\right)\right] \leq \Delta^2/2.$$

Similarly, we can show

$$\lim_{n\to\infty} k_n^2 \, \mathbb{E}_{X,W^n}\left[d\left(X, W^n\right)\right] \geq \Delta^2/2$$

and complete the proof. □

Derivations for (3.15). We partition $\mathrm{Supp}(f_X)$ by m points $x_1, x_2, \ldots, x_m$ and let x_0 and x_{m+1} be the two ends points of $\mathrm{Supp}(f_X)$, which could be $-\infty$ and $+\infty$ when $\mathrm{Supp}(f_X)$ is unbounded. We assume m is large enough such that

1. each interval $\mathcal{R}_j \triangleq (x_{j-1}, x_j], 1 \leq j \leq m+1$ is small enough so that the densities (f_X, ϕ, f_{W_i}) can be seen as constant over $\mathcal{R}_j$;
2. the expected number of partition points that fall into each region $\mathcal{R}_j$ satisfies $\mathbb{E}_{W^n}\left[N\left(x_{j-1}, x_j; W^n\right)\right] = \Omega\left(n^{1/2}\right)$.

Then

$$\mathbb{E}_{X,W^n}\left[d\left(X, W^n\right)\right] = \sum_{j=1}^{m+1} \mathbb{E}_{X,W^n}\left[d\left(X, W^n\right) \middle| X \in \mathcal{R}_j\right] \mathbb{P}\left[X \in \mathcal{R}_j\right]. \tag{A.3}$$

For each interval $\mathcal{R}_j, 1 \leq j \leq m+1$, based on the first assumption above,

$$\mathbb{P}\left[W_i \in \mathcal{R}_j\right] = f_{W_i}\left(x_j\right)\left|\mathcal{R}_j\right|$$

and the conditional density given that $W_i \in \mathcal{R}_j$ is uniform over $\mathcal{R}_j$. Therefore, let $p_{ij} = f_{W_i}\left(x_j\right)\left|\mathcal{R}_j\right|$, then

$$\begin{array}{ll} W_i \sim \mathsf{Unif}\left([x_{j-1}, x_j]\right) & w.p.\ p_{ij} \\ W_i \notin [x_{j-1}, x_j] & w.p.\ 1 - p_{ij}, \end{array}$$

and by the second assumption and Lemma A.3,

$$\mathbb{E}_{X,W^n}\left[d\left(X, W^n\right) \middle| X \in \mathcal{R}_j\right] \simeq \frac{\left|\mathcal{R}_j\right|^2}{2n_j^2},$$

where $n_j \triangleq \sum_{i=1}^n p_{ij}$. By the definition of $f_{\bar{W}}$ in (3.14),

$$n_j = n f_{\bar{W}}\left(x_j\right)\left|\mathcal{R}_j\right|.$$

Therefore,

$$\begin{aligned}\mathbb{E}_{X,W^n}\left[d\left(X,W^n\right)\right] &= \sum_{j=1}^{m+1}\mathbb{E}_{X,W^n}\left[d\left(X,W^n\right)|\, X\in\mathcal{R}_j\right]\mathbb{P}\left[X\in\mathcal{R}_j\right]\\ &\simeq \sum_{j=1}^{m+1}\frac{|\mathcal{R}_j|^2}{2n_j^2}\mathbb{P}\left[X\in\mathcal{R}_j\right]\\ &\simeq \sum_{j=1}^{m+1}\frac{|\mathcal{R}_j|^2}{2\left(nf_{\bar{W}}\left(x_j\right)|\mathcal{R}_j|\right)^2}f_X\left(x_j\right)|\mathcal{R}_j|\\ &\simeq \sum_{j=1}^{m+1}\frac{1}{2\left(nf_{\bar{W}}\left(x_j\right)\right)^2}f_X\left(x_j\right)|\mathcal{R}_j|\\ &\simeq \frac{1}{2n^2}\int f_X(x)f_{\bar{W}}^{-2}(x)\,dx.\end{aligned}$$

□

■ A.2.2 High resolution analysis of maximum quantization cell size

In this section we first define the notion of *spacing*, which plays a pivotal role in maximum quantization cell size analysis.

Definition A.1 (Spacing). *Given a sequence of independent random variables* $W_1, W_2, \ldots, W_{n-1}$ *with bounded support* $[a,b]$, *the* spacings *induced by* W^n *are the differences between two consecutive order statistics of* W^n, *i.e.,*

$$S_k \triangleq W_{(k)} - W_{(k-1)}, \quad 1\le k\le n \tag{A.4}$$

where we define $W_{(0)}\triangleq a, W_{(n)}\triangleq b$.

Lemma A.4 (Maximum spacing [68]). *Let* $U_1, U_2, \ldots, U_{n-1} \overset{i.i.d.}{\sim} \mathsf{Unif}([0,1])$, *then the maximum of the spacing* $S_1, S_2, \ldots, S_n$ *induced by* U^{n-1}, $S_{n:n}$, *satisfies*

$$\mathbb{P}\left[S_{n:n} > s\right] = \sum_{k=1}^{k_s}(-1)^{k-1}\binom{n}{k}(1-ks)^{n-1},$$

where $k_s = \lceil 1/s\rceil - 1$.

Proof. See [68, (6.4.4)] and the derivations therein. □

Lemma A.5 (Asymptotic approximation of maximum spacing [119]). *Let* $U_1, \ldots, U_{n-1} \overset{i.i.d.}{\sim} \mathsf{Unif}([0,1])$, *then the maximum of the spacing* $S_1, S_2, \ldots, S_n$ *induced by* U^{n-1}, $S_{n:n}$, *satisfies*

$$\lim_{n\to\infty}\mathbb{P}\left[S_{n:n} < \frac{\log n + a}{n}\right] = \exp\left[-\exp(-a)\right], \quad -\infty < a < \infty.$$

Alternatively,

$$\lim_{n\to\infty} \mathbb{P}\left[S_{n:n} < s\right] = \exp\left[-n\exp(-ns)\right], \quad -\infty < s < \infty.$$

Proof. See [119, Theorem 8.2 and 8.3].

Using Lemma A.5, we derive the high resolution analysis of maximum quantization cell size.

Derivations for (3.17). Given the cumulative distribution $F_{\bar{W}}(\cdot)$ and the range $[a,b]$, we define $w_0 = a$ and $w_m = b$, and partition the interval $[a,b]$ into m intervals by $m-1$ points $w_1, w_2, \ldots, w_{m-1}$. We denote the expected number of partition points in interval $[w_{i-1}, w_i]$ k_i, namely,

$$k_i \triangleq \mathbb{E}\left[N\left(w_{i-1}, w_i; W^n\right)\right].$$

We assume m is large enough such that $F_{\bar{W}}$ can be seen as constant over each interval $[w_{i-1}, w_i]$, and since $f_{\bar{W}}(w) > 0$, for any i, $k_i \to \infty$ as $n \to \infty$.

Define $S_{\mathrm{D}}\left(W^n; a, b\right)$ as the maximum spacing introduced by W^n over the interval $[a,b]$, then

$$\lim_{n\to\infty} \mathbb{P}\left[S_{\mathrm{D}}\left(W^n\right) \le s\right] \overset{(a)}{\simeq} \prod_{i=1}^{m} \mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [w_{i-1}, w_i]\right) \le s\right],$$

where in (a) we ignore the spacing cross two neighboring intervals as $n \gg m$. Then by the uniform assumption of $f_{\bar{W}}$ over $[w_{i-1}, w_i]$ and Lemma A.5, denoting the length of $[w_{i-1}, w_i]$ as $d_i = w_i - w_{i-1}$, we have

$$\lim_{n\to\infty} \mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [w_{i-1}, w_i]\right) \le s\right] \simeq \exp\left[-k_i \exp(-k_i s/d_i)\right],$$

where we approximate the number of partition points in $[w_{i-1}, w_i]$ by its expected value k_i, due to the concentration of measure phenomenon that we have shown in the derivation of (3.15) in Appendix A.2.1. Therefore,

$$\begin{aligned}\mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [a,b]\right) \le s\right] &\simeq \prod_{i=1}^{m} \mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [w_{i-1}, w_i]\right) \le s\right] \\ &= \exp\left[-\sum_{i=1}^{m} k_i \exp(-k_i s/d_i)\right],\end{aligned}$$

where we use the approximation that S_{D} in different intervals are independent. Noting that $k_i \simeq n f_{\bar{W}}(w_i)\, d_i$,

$$\begin{aligned}\sum_{i=1}^{m} k_i \exp(-k_i s/d_i) &= \sum_{i=1}^{m} n(w_i - w_{i-1}) f_{\bar{W}}(w_i)\, e^{-n s f_{\bar{W}}(w_i)} \\ &\simeq n \int_a^b f_{\bar{W}}(w)\, e^{-n s f_{\bar{W}}(w)}\, dw,\end{aligned}$$

and

$$\lim_{n\to\infty} \mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [a,b]\right) \le s\right] \simeq \exp\left\{-n \int_a^b f_{\bar{W}}(w)\, e^{-n s f_{\bar{W}}(w)}\, dw\right\}. \qquad \square$$

Derivations for (3.18). Let $s = t \log n/n$, then

$$\begin{aligned}\mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [a,b]\right) \le s\right] &\simeq \exp\left\{-n \int_a^b f_{\bar{W}}(w)\, e^{-t \log n f_{\bar{W}}(w)}\, dx\right\} \\ &= \exp\left\{-\int_a^b f_{\bar{W}}(w)\, e^{-t f_{\bar{W}}(w)} n^{1 - t f_{\bar{W}}(w)}\, dx\right\}.\end{aligned}$$

When $t > 1/f_{\min}^{[a,b]}$,

$$\lim_{n\to\infty} n^{1 - t f_{\bar{W}}(w)} \to 0$$

and thus $\mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [a,b]\right) \le s\right] \to 1$. Similarly, when $t < 1/f_{\max}^{[a,b]}$,

$$\lim_{n\to\infty} n^{1 - t f_{\bar{W}}(w)} \to \infty$$

and thus $\mathbb{P}\left[S_{\mathrm{D}}\left(W^n; [a,b]\right) \le s\right] \to 0$. $\square$

■ A.2.3 High resolution analysis of maximum quantization error

Derivation for (3.22). Let $\Delta = (b-a)/2$, then when $n \gg m$, $S_{\mathrm{D}} \le \Delta/m$ w.h.p.. Define $u_i \triangleq a + i \cdot (b-a)/m, 0 \le i \le m$, and $p_i(s) \triangleq F_{\bar{W}}(c_i + s) - F_{\bar{W}}(c_i - s)$, then for $0 < s < \Delta/m$,

$$\begin{aligned}\mathbb{P}\left[S_{\mathrm{I}} < \Delta/m + s\right] &\simeq \mathbb{P}\left[\forall u_i, \exists \tilde{V}_j \in [u_i - s, u_i + s], 1 \le i \le m-1\right] \\ &\simeq \prod_{i=1}^{m-1} \left[1 - (1 - p_i(s))^n\right] \\ &\simeq \prod_{i=1}^{m-1} \left[1 - e^{-n p_i(s)}\right] \\ &\simeq 1 - \sum_{i=1}^{m-1} e^{-n p_i(s)}\end{aligned}$$

$\square$

Derivation for (3.23). Note

$$2f_{\min}^{[a,b]}s \leq p_i(s) \leq 2f_{\max}^{[a,b]}s,$$

thus

$$1-(m-1)e^{-2nf_{\min}^{[a,b]}s} \leq 1-\sum_{i=0}^{m} e^{-np_i(s)} \leq 1-(m-1)e^{-2nf_{\max}^{[a,b]}s}.$$

Then for

$$s=\frac{\log m}{2nf_{\max}^{[a,b]}},,$$

$$\mathbb{P}\left[S_{\mathrm{I}} \leq \Delta/m+s\right] \leq 1-(m-1)e^{-2nf_{\max}^{[a,b]}s}=0$$

and for

$$s=\frac{\log m+t}{2nf_{\min}^{[a,b]}},,$$

$$\mathbb{P}\left[S_{\mathrm{I}} \leq \Delta/m+s\right] \leq 1-(m-1)e^{-2nf_{\min}^{[a,b]}s}=1-e^{-t}. \qquad \square$$

■ A.3 Proofs regarding Flash ADC design

■ A.3.1 Proof of Lemma 3.1

Proof for Lemma 3.1.

$$\begin{aligned}
&\frac{1}{n}\mathbb{E}\left[N\left(x,x+dx;\tilde{V}^n\right)\right]\\
=&\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\mathbb{1}\left\{\tilde{V}_i\in[x,x+dx]\right\}\right]\\
=&\frac{1}{n}\sum_{i=1}^{n}\mathbb{P}\left[V_i+Z_i\in[x,x+dx]\right]\\
=&\int\phi(z)\frac{1}{n}\sum_{i=1}^{n}\mathbb{P}\left[V_i\in[x-z,x-z+dx]\right]d\,z\\
=&\int\phi(z)\frac{1}{n}N\left(x-z,x-z+dx;V^n\right)d\,z\\
=&\int\phi(z)\tau(x-z)dxd\,z=(\tau*\phi)(x)dx. \qquad \square
\end{aligned}$$

■ A.3.2 Optimal partition point density analysis

In this section we first prove the optimal conditions in Theorem 3.2. Following that we specialize Theorem 3.2 to $\tau^*(\cdot) = \delta(\cdot)$ in Lemma A.6, and derive the corresponding conditions for Gaussian and uniform input distributions respectively in Lemmas A.7 and A.8.

Proof for Theorem 3.2. When the existence condition in (3.28) is satisfied, then (3.29) follows from the Panter and Dite formula [22].

In general, given the optimal τ^*, for any distribution h such that

$$\int h = 1, \tag{A.5}$$

$R((1-\epsilon)\tau^* + \epsilon h) \geq R(\tau^*)$. Therefore,

$$\lim_{\varepsilon \to 0} \frac{R((1-\epsilon)\tau^* + \epsilon h) - R(\tau^*)}{\varepsilon} \geq 0,$$

which leads to

$$\begin{aligned} \left\langle f_X, \frac{1}{(\tau^* * \phi)^2} \right\rangle &\geq \left\langle h * \phi, \frac{f_X}{(\tau^* * \phi)^3} \right\rangle \\ &= \left\langle h, \frac{f_X}{(\tau^* * \phi)^3} * \phi \right\rangle. \end{aligned}$$

Since the above holds for any h that satisfies (A.5), we have

$$\sup_x \left(\frac{f_X}{(\tau^* * \phi)^3} * \phi \right)(x) \leq \left\langle f_X, \frac{1}{(\tau^* * \phi)^2} \right\rangle. \tag{A.6}$$

□

Lemma A.6 (Condition for $\tau^*(x) = \delta(x)$). *Define*

$$g(x) \triangleq \left(\frac{f_X}{\phi^3} * \phi \right)(x), \tag{A.7}$$

then if for any $x \in \mathcal{A}$, $g'(x) \leq 0$, $\tau^*(x) = \delta(x)$.

Proof. Substitute $\tau^*(x) = \delta(x)$ in (A.6), we have

$$\sup_x \left(\frac{f_X}{\phi^3} * \phi \right)(x) \leq \left\langle f_X, \frac{1}{\phi^2} \right\rangle. \tag{A.8}$$

Since f_X is symmetric and smooth on $\mathcal{A}$, $g(x)$ is an even function on $\mathcal{A}$ and is smooth, therefore, $g'(0) = 0$. Since

$$\sup_x g(x) \geq g(0) = \left\langle f_X, \frac{1}{\phi^2} \right\rangle, \tag{A.9}$$

we know if for any $x \in \mathcal{A}$ $g'(x) \leq 0$ then $x = 0$ maximizes $g(x)$, thus (A.8) is satisfied and hence $\delta(x)$ is indeed the optimal solution. □

Remark A.1. *Calculation based on (A.7) shows*

$$g'(x) = 2\pi \int \left\{ f_X(t)(t-x) \exp\left[\frac{\left(3t^2 - (x-t)^2\right)}{2\sigma^2} \right] \right\} dt. \tag{A.10}$$

Below we show that for both Gaussian and uniform input distributions, the optimal quantization density $\tau^*(x) = \delta(x)$ when the noise standard deviation σ is above some threshold.

Lemma A.7. *When $X \sim \mathsf{N}\left(0, \sigma_X^2\right)$, $\tau^*(x) = \delta(x)$ if and only if $\sigma^2 \geq 3\sigma_X^2$.*

Proof. When $\sigma^2 \geq 3\sigma_X^2$, substitute $f_X(t) \propto \exp\left[-t^2/(2\sigma_X^2)\right]$ into (A.10), we have

$$\begin{aligned} g'(x) &\propto \int (t-x) \exp\left[-\frac{\sigma^2 - 2\sigma_X^2}{2\sigma_X^2\sigma^2} \left(t - \frac{x\sigma_X^2}{\sigma^2 - 2\sigma_X^2} \right)^2 \right] d\,x \\ &= \frac{x\sigma_X^2}{\sigma^2 - 2\sigma_X^2} - x = \frac{x(3\sigma_X^2 - \sigma^2)}{\sigma^2 - 2\sigma_X^2} \leq 0. \end{aligned}$$

When $\sigma^2 < 3\sigma_X^2$, $\tau^*(x) \neq \delta(x)$ by (3.29) in Theorem 3.2. □

Lemma A.8. *When $X \sim \mathit{Unif}([-1,1])$ $\tau^*(x) = \delta(x)$ if and only if $\sigma \geq \sigma_0 \approx 0.7228$.*

Proof. From (A.10) we have for $\mathsf{Unif}([-1,1])$,

$$g'(x) \propto \int_{-1}^{1} (t-x) \exp\left(\frac{t + x/2}{\sigma^2} \right)^2 d\,t.$$

Numerically solution indicates if $\sigma \geq \sigma_0 \approx 0.7228$, $g'(x) \leq 0$ for any x, and if $\sigma < \sigma_0$, $g''(0) > 0$, and (3.27) is violated when $\tau(x) = \delta(x)$. □

■ A.4 Proofs for the deterministic error correction setting

■ A.4.1 Proofs for the general purpose setting

Proof of Lemma 4.6. For each functional element s_i with the set of possible elements $\mathcal{A}_i(\mathcal{S})$, it needs to connect to t copies for each type of element $x \in \mathcal{A}_i(\mathcal{S})$. Therefore,

$$|\mathcal{E}| \geq \sum_{i=1}^{k} t n_i = kt\bar{n},$$

and thus $E = |\mathcal{E}|/k \geq t\bar{n}$. In addition, define $n_x = \sum_{i=1}^{k} \mathbb{1}\{x \in \mathcal{A}_i(\mathcal{S})\}$ and let m_x be the number of redundant elements with type x, then $m_x d \geq n_x t$, and hence

$$\begin{aligned} m &= \sum_{x \in \mathcal{X}} m_x \\ &\geq t/d \sum_{x \in \mathcal{X}} n_x \\ &= t/dk\bar{n}, \end{aligned}$$

and $\rho = m/k \geq t\bar{n}/d$. □

■ A.4.2 Proofs for the application-specific setting

To prove (a) in Lemma 4.9, we first show the following lemma.

Lemma A.9. *For the application-specific setting, a redundant circuit $\mathcal{C}$ is t-correcting only if for any subsequence $s_{\mathcal{I}}$ of s^k,*

$$|\mathcal{N}(s_{\mathcal{I}})| \geq t\,|\mathcal{A}(s_{\mathcal{I}})|\,,$$

where $\mathcal{N}(s_{\mathcal{I}}) = \cup_{i\in\mathcal{I}}\mathcal{N}(s_i)$.

Proof of (a) in Lemma 4.9. Since $E = t$ and the circuit to be t-correcting, each functional element s_i connects to exactly t redundant elements. If there exist $i \neq j$ such that $\mathcal{N}(s_i) \cap \mathcal{N}(s_j) \neq \emptyset$, then $\mathcal{N}(s_i) \cup \mathcal{N}(s_j) < 2t$, by Lemma A.9 this circuit is not t-error correcting when $s_i \neq s_j$. Therefore, $\mathcal{N}(s_i) \cap \mathcal{N}(s_j) = \emptyset$ for any $i \neq j$, and hence $\rho = t$. □

■ A.5 Proofs for the probabilistic error correction setting

■ A.5.1 Proofs for Lemma 4.11

We use the standard random graph processes $\mathcal{G}(m, k, p = p(k))$, which is a bipartite graph with m nodes on the left (vertex set $\mathcal{U}$) and k nodes on the right (vertex set $\mathcal{V}$), and generate edges with probability p. We assume $m = \rho k$, where $\rho \geq 1$.

We show a general result regarding matching in random bipartite graph in Lemma A.11, which relies on Lemma A.10.

Lemma A.10. *For a random bipartite graph $\mathcal{G}(m, k, p)$, if*

$$mp_k = \log k + w(k),$$

where $w(k) \to \infty$ as $k \to \infty$, then

$$\mathbb{P}\left[\deg(s_i) > 0\ \forall\ s_i \in \mathcal{V}\right] \to 1.$$

Proof. Note that the distribution for the degree of s_i $\deg(s_i) \overset{i.i.d.}{\sim} \mathsf{Binomial}\,(m, p_k)$, and hence the probability of existing at least one zero degree node in $\mathcal{V}$ is

$$\begin{aligned}\mathbb{P}\left[\min_{s_i\in\mathcal{V}} \deg(s_i) = 0\right] &\leq \sum_{s_i\in\mathcal{V}} \mathbb{P}\left[\deg(s_i) = 0\right] \\ &= k(1-p_k)^m \\ &\leq \exp\left\{-\left[mp_k - \log k\right]\right\}.\end{aligned}$$

□

Lemma A.11 (Matching in random bipartite graph). *For a random bipartite graph $\mathcal{G}(m, k, p)$, if*

$$mp_k = \log k + w(k),$$

where $w(k) \to \infty$ *as* $k \to \infty$*, then*

$$\mathbb{P}\left[G \textit{ does not contain a matching with size } k\right] \to 0$$

as $k \to \infty$*.*

Proof. This results follows from Lemma A.10 and a generalized version of [120, Theorem 7.11], where we generalize the proof from perfect matching to a matching that saturates $\mathcal{V}$. The generalization can be done by modifying [120, Lemma 7.9,Lemma 7.12] accordingly.□

Proof of Lemma 4.11. Construct a sequence of random bipartite graph ensembles $\mathcal{G}(m, k, p)$ with

$$m = \rho k \tag{A.11}$$

$$p_k = \frac{\log k + w(k)}{m(1-\varepsilon)}, \tag{A.12}$$

where $\rho = \varepsilon/(1-\varepsilon)$ and $w(k) \to \infty$ as $k \to \infty$. We want to show that there exists a sequence of graphs in this sequence of ensembles that achieves $(\rho, E) = (\varepsilon/(1-\varepsilon), 1/(1-\varepsilon))$.

Let $\mathbf{e} \in \{0,1\}^{k+m}$ be the random error pattern vector of a given circuit, where 1 is used to indicate failures and 0 otherwise, and let $m'(\mathbf{e})$ be the number of failed redundant elements and $k'(\mathbf{e})$ be the number of failed functional elements indicated by $\mathbf{e}$ respectively, then for any $\lambda > 0$, define

$$\begin{aligned} E_1(\lambda) &= \left\{\mathbf{e} : m' \le \rho k(\varepsilon+\lambda), k' \le k(\varepsilon+\lambda)\right\}, \\ E_2(\lambda) &= \{0,1\}^{k+m} \setminus E_1(\lambda). \end{aligned}$$

Using G to denote a graph drawn from the random bipartite graph ensemble $\mathcal{G}(m, k; p)$, then

$$\begin{aligned} \overline{P_f} &= \sum_G P_{\mathcal{G}}(G) \sum_{\mathbf{e}} P_e(\mathbf{e}) P_f(G, \mathbf{e}) \\ &= \sum_{\mathbf{e}} P_e(\mathbf{e}) \sum_G P_{\mathcal{G}}(G) P_f(G, \mathbf{e}) \\ &= \sum_{\mathbf{e} \in E_1(\lambda)} P_e(\mathbf{e}) P_f(\mathbf{e}) + \sum_{\sum_{\mathbf{e}\in E_2(\lambda)}} P_e(\mathbf{e}) P_f(\mathbf{e}). \end{aligned}$$

where $P_f(\mathbf{e}) \triangleq \sum_G P_{\mathcal{G}}(G) P_f(G, \mathbf{e})$.

Note that as $k \to \infty$, there exists $\lambda_k \to 0$ such that

$$\sum_{\mathbf{e} \in E_2(\lambda_k)} P_e(\mathbf{e}) \to 0. \tag{A.13}$$

In addition, for $\rho > \varepsilon/(1-\varepsilon)$, we have for any $e \in E_1(\lambda_k)$, when k is sufficiently large, the number of working redundant elements is no less than the number of failed functional elements. Then by Lemma A.11, for any $\mathbf{e} \in E_1(\lambda_k)$, which essentially results in a

$\mathcal{G}(m', k', p)$ subgraph,

$$P_f(\mathbf{e}) \to 0. \tag{A.14}$$

Therefore, (A.13) and (A.14) lead to

$$\overline{P_f} \to 0.$$

Hence for any $\delta > 0$, there exits K such that for any $k \geq K$,

$$\overline{P_f} \leq \delta/2. \tag{A.15}$$

One the other hand, define

$$\begin{aligned}
\mathcal{G}_1(\lambda) &\triangleq \{G \in \mathcal{G} : \mathcal{E}(G) \leq km(p_k + \lambda)\}, \\
\mathcal{G}_2(\lambda) &\triangleq \{G \in \mathcal{G} : \mathcal{E}(G) > km(p_k + \lambda)\}.
\end{aligned}$$

Note that

$$\begin{aligned}
\overline{P_f} &= \sum_{G} P_{\mathcal{G}}(G) P_f(G) \\
&= \sum_{G \in \mathcal{G}_1} P_{\mathcal{G}}(G) P_f(G) + \sum_{G \in \mathcal{G}_2} P_{\mathcal{G}}(G) P_f(G),
\end{aligned}$$

where $P_f(G) \triangleq \sum_{\mathbf{e}} P_e(\mathbf{e}) P_f(G, \mathbf{e})$, and as $k \to \infty$, there exists $\lambda_k \to 0$ such that

$$\begin{aligned}
\sum_{G \in \mathcal{G}_1(\lambda_k)} P_{\mathcal{G}}(G) &\to 1 \\
\sum_{G \in \mathcal{G}_2(\lambda_k)} P_{\mathcal{G}}(G) &\to 0.
\end{aligned}$$

If for any $G \in \mathcal{G}_1(\lambda_k)$, $P_f(G) \geq \delta$, then

$$\begin{aligned}
\overline{P_f} &= \sum_{G} P_{\mathcal{G}}(G) P_f(G) \\
&\geq \sum_{G \in \mathcal{G}_1(\lambda_k)} P_{\mathcal{G}}(G) \delta \\
&\geq \delta/2,
\end{aligned}$$

which contradicts (A.15). Therefore, there exists a sequence of graph $\{G \in \mathcal{G}_1\}$ such that $P_f(G) \to 0$, which leads to

$$E(\mathcal{G}) = \frac{1}{1 - \varepsilon}.$$

□

■ A.5.2 Proof of Lemma 4.12

Theorem A.12 (Chernoff bound [121]). *Let X be a Binomial random variable* Binomial (n, p)*, then*

$$\mathbb{P}[X > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}, \tag{A.16}$$

$$\mathbb{P}[X < (1-\delta)\mu] < \exp(-\mu\delta^2/2), \tag{A.17}$$

where $\mu = \mathbb{E}[X] = np$.

Proof of Lemma 4.12. Let $t = -\log\varepsilon$, then the lemma follows from Theorem A.12 and union bound. □

■ A.5.3 Proof of Lemma 4.13

Proof of Lemma 4.13. Let $c_k = 1/\log k$, and let

$$t = (1+c_k)\frac{\log k}{\log(1/\varepsilon)},$$

then for the redundant circuit $\mathcal{C}_{\mathrm{dr}}$,

$$\begin{aligned} P_{\mathrm{e}} &= \mathbb{P}[\text{exists } s_i \text{ that cannot be corrected}] \\ &\le k\varepsilon^{t+1} = k\varepsilon k^{-1-c_k} = k^{-c_k}\varepsilon \to 0. \end{aligned}$$

as $k \to \infty$. Therefore, as $k \to \infty$, $\rho = t \to \infty$, and

$$\begin{aligned} E &= \lim_{k\to\infty} \frac{kt}{k\log k} \\ &= \lim_{k\to\infty} \frac{1+c_k}{\log(1/\varepsilon)} = \frac{1}{\log(1/\varepsilon)}. \end{aligned}$$

□

■ A.5.4 Proof of Lemma 4.14

Proof of Lemma 4.13. Assume a redundant circuit has $E < -1/\log(\varepsilon)$, then let $E = -(1-\delta)1/log\varepsilon, \delta > 0$. Then when k large enough, there exists at least $\delta/2k$ functional elements with degree less than $t = -\log k/\log\varepsilon$. Therefore,

$$\mathbb{P}[\text{One of these functional elements cannot be corrected}] \ge 1 - (1-\varepsilon^{t+1})^{\delta k + o(k)}.$$

Noting

$$\varepsilon^{t+1} > \varepsilon/k,$$

$$\begin{aligned}\lim_{k\to\infty} 1-(1-\varepsilon^{t+1})^{\delta k/2} &\geq \lim_{k\to\infty} 1-(1-\varepsilon/k)^{\delta k/2} \\ &= \lim_{k\to\infty} 1-\left[(1-\varepsilon/k)^{k/\varepsilon}\right]^{\delta\varepsilon/2} \\ &= 1-e^{\delta\varepsilon/2} > 0.\end{aligned}$$

Therefore, this redundant circuit is not ε-reliable. □

Appendix B

Results and proofs for Part II

B.1 Some results in order statistics

In this section we provide some results in order statistics they are useful for our performance analysis.

We calculate the expected value of various order statistics (maxima, minima, k-th) of two common distributions, Exponential and Pareto, in Lemmas B.1 and B.2 and Lemmas B.3 and B.4 respectively.

Also, in Lemma B.5, we provide the values of the expected value for the extreme value distributions defined in Theorem 6.5.

Exponential distribution The minimum of Exponential random variables is again an Exponential random variable. The maximum of Exponential random variables follows the Gumbel distribution in (6.7).

Lemma B.1 (Minimum and Maximum of Exponential random variables). *Given* $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F_X \sim \mathsf{Exp}(\lambda)$, *i.e.,*

$$F_X(x) \triangleq 1 - e^{-\lambda x}, x \geq 0,$$

then

$$\begin{aligned} X_{1:n} &\sim \mathsf{Exp}(n\lambda), \\ \lambda X_{n:n} - \ln n &\sim \mathsf{Gumbel}. \end{aligned}$$

Lemma B.2. *Given an Exponential distribution* $F_X \sim \mathsf{Exp}(\lambda)$, *for* $1 \leq k \leq n$,

$$\mathbb{E}[X_{k:n}] = (H_n - H_{n-k})/\lambda, \tag{B.1}$$

where $H_n = \sum_{k=1}^{n} 1/k$ *is the* n*-th harmonic number.*

Pareto distribution The minimum of Pareto random variables is again an Pareto random variable. The maximum of Pareto random variables follows the Fréchet distribution in (6.8).

Lemma B.3 (Minimum and Maximum of Pareto random variables). *Given* $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F_X \sim \mathsf{Pareto}(\alpha, x_m)$, *i.e.,*

$$F_X(x) \triangleq \begin{cases} 1 - (x_m/x)^{\alpha} & x \geq x_m \\ 0 & x < x_m \end{cases},$$

then

$$X_{1:n} \sim \textit{Pareto}\left(n\alpha, x_m\right),$$
$$\frac{X_{n:n}}{x_m n^{1/\alpha}} \sim \textit{Fréchet}\left(\alpha\right).$$

Lemma B.4 (Expected value of order statistics [122]). *Given a Pareto distribution* $F_X \sim \textit{Pareto}\left(x_m, \alpha\right)$*, for* $1 \leq k \leq n$*,*

$$\mathbb{E}\left[X_{k:n}\right] = \frac{\Gamma\left(n-k+1-1/\alpha\right) \cdot n!}{\Gamma\left(n+1-1/\alpha\right) \cdot (n-k)!} x_m, \tag{B.2}$$

where $\Gamma\left(\cdot\right)$ *is the Gamma function (cf. (6.10)).*

Lemma B.5 (Expected value of extreme value distributions).

$$\mathbb{E}\left[\Lambda\right] = \gamma_{\mathrm{EM}},$$
$$\mathbb{E}\left[\Phi_\alpha\right] = \begin{cases} \Gamma\left(1-1/\alpha\right) & \alpha > 1 \\ +\infty & \textit{otherwise} \end{cases},$$
$$\mathbb{E}\left[\Psi_\alpha\right] = -\Gamma\left(1+1/\alpha\right),$$

where γ_{EM} *is the Euler-Mascheroni constant and* $\Gamma(\cdot)$ *is the Gamma function, i.e.,*

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

■ B.2 Proofs regarding the single-fork policy

■ B.2.1 Proof for latency and costs

Proof for Lemma 6.1. Since we apply the same action to all remaining tasks at time $T^{(1)}$, Y_j are i.i.d.. Also,

$$\begin{aligned} \mathbb{P}\left[Y_j > y\right] &= \mathbb{P}\left[\min\left\{\Delta_{X_j,T^*}, {X'}_{1:r}\right\} > y\right] \\ &= \mathbb{P}\left[\Delta_{X_j,T^*} > y\right] \mathbb{P}\left[{X'}_{1:r} > y\right] \\ &= \bar{F}_{\Delta_{X,T^*}}\left(y\right) \left(\bar{F}_X\left(y\right)\right)^r. \end{aligned}$$

□

Proof for Theorem 6.3. The total cost of $\pi_{\mathrm{SF}}\left(p, r, l; n\right)$ can be analyzed by the cost in the first stage and second stage.

For cloud computing,

$$\begin{aligned} C_{\text{cloud}}\left(\pi_{\text{SF}}\right) &= C^{(1)}\left(\pi_{\text{SF}}\right) + C^{(2)}\left(\pi_{\text{SF}}\right) \\ C^{(1)}\left(\pi_{\text{SF}}\right) &= \sum_{i=1}^{\bar{p}n} X_{i:n} + npT^{(1)}\left(\pi_{\text{SF}}\right) \\ C^{(2)}\left(\pi_{\text{SF}}\right) &= (r+1)\sum_{j=1}^{pn} Y_{j:pn} \\ &= (r+1)\sum_{j=1}^{pn} Y_j \end{aligned}$$

For crowd sourcing,

$$C_{\text{crowd}}\left(\pi_{\text{SF}}\right) = n + npr$$ □

■ B.2.2 Proofs for Theorem 6.10

Proof of Lemma 6.9. Denote $x^* = l \cdot x_{\bar{p}}$. First, note that

$$\begin{aligned} \bar{F}_Y(y) &= \bar{F}_{\Delta_{X,x^*}}(y)\left(\bar{F}_X(y)\right)^r. \\ &= \frac{\bar{F}_X(x^*+y)}{\bar{F}_X(x^*)}\left(\bar{F}_X(y)\right)^r. \end{aligned}$$

If $F_X \in \text{DA}(\Lambda)$, then

$$\lim_{x\to\omega(F_X)^-} \frac{\bar{F}_X(x+t\eta(x))}{\bar{F}_X(x)} = e^{-t},$$

and thus

$$\begin{aligned} \lim_{y\to\omega(F_Y)^-} \frac{\bar{F}_Y(y+u\eta(y))}{\bar{F}_Y(y)} &= \lim_{y\to\omega(F_Y)^-} \frac{\bar{F}_X(x^*+y+u\eta(y))\left(\bar{F}_X(y+u\eta(y))\right)^r}{\bar{F}_X(x^*+y)\left(\bar{F}_X(y)\right)^r} \\ &= \lim_{y\to\omega(F_Y)^-} \frac{\bar{F}_X(x^*+y+u\eta(y))}{\bar{F}_X(x^*+y)}\left(\frac{\bar{F}_X(y+u\eta(y))}{\bar{F}_X(y)}\right)^r \\ &= \begin{cases} e^{-u(r+1)} & \omega(F_X) = \infty \text{ or } x^* = 0 \\ e^{-u} & \omega(F_X) < \infty \text{ and } x^* \neq 0 \end{cases} \end{aligned} \tag{B.3}$$

Therefore, $F_Y \in \text{DA}(\Lambda)$.

If $F_X \in \text{DA}(\Phi_\alpha)$, then for any $t > 0$,

$$\lim_{x\to\infty} \frac{\bar{F}(tx)}{\bar{F}(x)} = t^{-\alpha}, \quad \alpha > 0,$$

and thus for any $u > 0$,

$$\begin{aligned}\lim_{y\to\infty}\frac{\bar{F}_Y(uy)}{\bar{F}_Y(y)} &= \lim_{y\to\infty}\frac{\bar{F}_X(x^*+uy)\left(\bar{F}_X(uy)\right)^r}{\bar{F}_X(x^*+y)\left(\bar{F}_X(y)\right)^r}\\ &= \lim_{y\to\infty}\frac{\bar{F}_X(x^*+uy)}{\bar{F}_X(x^*+y)}\left(\frac{\bar{F}_X(yu)}{\bar{F}_X(y)}\right)^r\\ &= u^{-(r+1)\alpha}, \quad u>0\end{aligned} \tag{B.4}$$

Therefore, $F_Y \in \mathrm{DA}\left(\Phi_{(r+1)\alpha}\right)$.

If $F_X \in \mathrm{DA}\left(\Psi_\alpha\right)$, then for any $t > 0$,

$$\lim_{x\to 0^+}\frac{\bar{F}_X(\omega(F_X)-tx)}{\bar{F}_X(\omega(F_X)-x)} = t^\alpha, \quad \alpha>0,$$

and note that $\omega(F_Y) = \omega(F_X) - x^*$,

$$\begin{aligned}\lim_{y\to 0}\frac{\bar{F}_Y(\omega(F_Y)-uy)}{\bar{F}_Y(\omega(F_Y)-y)} &= \lim_{y\to 0}\frac{\bar{F}_X(\omega(F_Y)-uy+x^*)\left(\bar{F}_X(\omega(F_Y)-uy)\right)^r}{\bar{F}_X(\omega(F_Y)-y+x^*)\left(\bar{F}_X(\omega(F_Y)-y)\right)^r}\\ &= \lim_{y\to 0}\frac{\bar{F}_X(\omega(F_X)-uy)}{\bar{F}_X(\omega(F_X)-y)}\left(\frac{\bar{F}_X(\omega(F_X)-x^*-uy)}{\bar{F}_X(\omega(F_Y)-x^*-y)}\right)^r\\ &= \begin{cases}u^{(r+1)\alpha} & x^*=0\\ u^\alpha & x^*>0\end{cases}.\end{aligned} \tag{B.5}$$

Therefore, $F_Y \in \mathrm{DA}\left(\Psi_{((1-l)r+1)\alpha}\right)$. □

Proofs of Theorem 6.10. Lemma 6.9 indicates F_Y is in the same domain of attraction as F_X. Then (6.6) indicates

$$\mathbb{E}[Y_{n:n}] = \tilde{a}_n\mathbb{E}[G_\gamma] + \tilde{b}_n,$$

and finally the proof follows from Lemma B.5 and Theorem 6.5. □

■ B.3 Calculations regarding the single-fork policy

■ B.3.1 Calculations for Pareto execution time distribution

Scaling coefficient for the case of no relaunching

In the case of no relaunching, $Y = \min\{\mathsf{Pareto}(\alpha, x_{\bar{p}}) - x_{\bar{p}}, \mathsf{Pareto}(r\alpha, x_m)\}$, which leads to

$$\bar{F}_Y(y) = \begin{cases}\left(1+\dfrac{y}{x_{\bar{p}}}\right)^{-\alpha} & y < x_m\\ \left[\left(1+\dfrac{y}{x_{\bar{p}}}\right)\left(\dfrac{y}{x_m}\right)^r\right]^{-\alpha} & y \geq x_m\end{cases}.$$

Then by Corollary 6.7, as $n \to \infty$, $\tilde{a}_n = \bar{F}_Y^{-1}(1/n)$, thus

$$\left[\left(1+\frac{\tilde{a}_n}{x_{\bar{p}}}\right)\left(\frac{\tilde{a}_n}{x_m}\right)^r\right] = n^{1/\alpha},$$

which leads to (6.12).

Calculation of $\mathbb{E}[Y]$ for the case of no relaunching

$$\mathbb{E}[Y] = \mathbb{E}[Y|\, Y < x_m]\,\mathbb{P}[Y < x_m] + \mathbb{E}[Y|\, Y \geq x_m]\,\mathbb{P}[Y \geq x_m]. \tag{B.6}$$

Let $Z \sim \mathsf{Pareto}(x_{\bar{p}}, \alpha) - x_{\bar{p}}$, then $Z|Z \geq x_m \sim \mathsf{Pareto}(x_{\bar{p}} + x_m, \alpha)$, and thus

$$\begin{aligned}\bar{F}_{Y|Y\geq x_m}(y) &= \left(\frac{x_{\bar{p}}+x_m}{y}\right)^{\alpha}\left(\frac{x_m}{y}\right)^{r\alpha} \\ &= \left(\frac{[(x_{\bar{p}}+x_m)x_m^r]^{1/(r+1)}}{y}\right)^{(r+1)\alpha},\end{aligned}$$

which corresponds to $\mathsf{Pareto}\left([(x_{\bar{p}}+x_m)x_m^r]^{1/(r+1)}, (r+1)\alpha\right)$. Therefore

$$\mathbb{E}[Y|\, Y \geq x_m] = \frac{(r+1)\alpha}{(r+1)\alpha - 1}[(x_{\bar{p}}+x_m)x_m^r]^{1/(r+1)}.$$

Regarding $\mathbb{E}[Y|\, Y < x_m]$, we have

$$F_{Y|Y<x_m}(y) = \frac{1}{\int_0^{x_m} F_Y(y)\, dy}\left[1 - \left(\frac{x_{\bar{p}}}{y+x_{\bar{p}}}\right)^{\alpha}\right],$$

and we can evaluate $\mathbb{E}[Y|\, Y < x_m] = \int_0^{x_m}\left[1 - F_{Y|Y<x_m}(y)\right] dy$ numerically.

Calculation for the cloud computing cost

For the nodes that finish before replication, the total running time is

$$\begin{aligned}\sum_{i=1}^{\bar{p}n}\mathbb{E}[X_{i:n}] &\approx \sum_{i=1}^{\bar{p}n} F_X^{-1}(i/n) \\ &= x_m\sum_{i=1}^{\bar{p}n}(1-i/n)^{-1/\alpha} \\ &\approx x_m n\int_0^{\bar{p}}(1-x)^{-1/\alpha}dx \\ &= nx_m\frac{\alpha}{\alpha-1}\left[1-p^{(\alpha-1)/\alpha}\right].\end{aligned}$$

Therefore, by Theorem 6.3,

$$\mathbb{E}\left[C_{\text{cloud}}\left(\pi_{\text{SF}}\right)\right] = nx_m\frac{\alpha}{\alpha-1}\left[1-p^{(\alpha-1)/\alpha}\right] + np\mathbb{E}\left[X_{\bar{p}n:n}\right] + (r+1)(pn)\mathbb{E}\left[Y\right],$$

where $\mathbb{E}\left[X_{\bar{p}n:n}\right]$ given by Lemma B.4, $\mathbb{E}\left[Y\right]$ for no relaunching is discussed in Appendix B.3.1 and $\mathbb{E}\left[Y\right]$ with relaunching satisfies

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

■ B.3.2 Calculations for Shifted Exponential execution time distribution

Calculation for the case of no relaunching

In the case of no relaunching, $Y = \min\left\{\mathsf{Exp}\left(\lambda\right), x_m + \mathsf{Exp}\left(r\lambda\right)\right\}$, and thus

$$\bar{F}_Y\left(y\right) = \begin{cases} e^{-\lambda y} & 0 < y < x_m \\ e^{\lambda r x_m}e^{-\lambda(r+1)y} & y \geq x_m \end{cases}. \tag{B.7}$$

Calculation of $\mathbb{E}\left[Y\right]$

$$\mathbb{E}\left[Y\right] = \mathbb{E}\left[Y \middle| Y < x_m\right]\mathbb{P}\left[Y < x_m\right] + \mathbb{E}\left[Y \middle| Y \geq x_m\right]\mathbb{P}\left[Y \geq x_m\right]. \tag{B.8}$$

Let $Z \sim \mathsf{Exp}\left(\lambda\right)$, then

$$\mathbb{E}\left[Y \middle| Y < x_m\right]\mathbb{P}\left[Y < x_m\right] = \mathbb{E}\left[Z \middle| Z < x_m\right]\mathbb{P}\left[Z < x_m\right] \tag{B.9}$$

$$\mathbb{E}\left[Y \middle| Y \geq x_m\right]\mathbb{P}\left[Y \geq x_m\right] = \mathbb{E}\left[\mathsf{Exp}\left((r+1)\lambda\right)\right]\left(e^{-\lambda r x_m}\right) = \frac{1}{(r+1)\lambda}\left(e^{-\lambda r x_m}\right). \tag{B.10}$$

In addition,

$$\begin{aligned} 1/\lambda = \mathbb{E}\left[Z\right] &= \mathbb{E}\left[Z \middle| z < x_m\right]\mathbb{P}\left[Z < x_m\right] + \mathbb{E}\left[Z \middle| z \geq x_m\right]\mathbb{P}\left[Z \geq x_m\right] \\ &= \mathbb{E}\left[Z \middle| z < x_m\right]\mathbb{P}\left[Z < x_m\right] + e^{-\lambda x_m}/\lambda. \end{aligned} \tag{B.11}$$

Then $\mathbb{E}\left[Y\right]$ in (6.17) follows from (B.8) to (B.11).

Calculation of $\mathbb{E}\left[Y_{pn:pn}\right]$ Based on (B.7), for

$$\eta(y) = \frac{1}{(r+1)\lambda},$$

we have

$$\lim_{y\to\omega(F_Y)}\frac{\bar{F}_Y\left(y+u\eta(y)\right)}{\bar{F}_Y\left(y\right)} = e^{-u}, \tag{B.12}$$

and

$$\tilde{a}_n = 1/\left[\lambda(1+r)\right],$$
$$\tilde{b}_n = \bar{F}_Y^{-1}(1/n) = \frac{\ln(pn) + \lambda r x_0}{\lambda(r+1)}.$$

Then $\mathbb{E}[Y_{pn:pn}]$ in (6.17) follows from Theorem 6.10.

Argument for Observation 6.18

Noting $H_n \approx \ln n + \gamma_{\text{EM}}$, then the total running time for nodes that finish before replication is

$$\begin{aligned}
\sum_{i=1}^{\bar{p}n} \mathbb{E}[X_{i:n}] &= nx_m + \sum_{i=0}^{\bar{p}n} \frac{1}{\lambda}(H_n - H_{n-i}) \\
&\approx nx_m + \frac{1}{\lambda}\sum_{i=0}^{\bar{p}n} \ln\frac{n}{n-i} \\
&\approx nx_m + \frac{1}{\lambda}\left[\bar{p}n\ln n - \ln\frac{n!}{(pn)!}\right] \\
&\approx nx_m + \frac{1}{\lambda}\left[\bar{p}n + pn\ln p\right].
\end{aligned}$$

Therefore,

$$\begin{aligned}
\lambda\mathbb{E}[C_{\text{cloud}}] &= \lambda pnx_m - pn\ln p + \lambda nx_m + [\bar{p}n + pn\ln p] + (r+1)pn\lambda\mathbb{E}[Y] \\
&= \lambda nx_m + n + pn\left(\lambda x_m + (r+1)\lambda\cdot\mathbb{E}[Y] - 1\right) \\
&= \begin{cases} \lambda nx_m + n + pn\left[\lambda x_m + r\left(1 - e^{-\lambda x_m}\right)\right] & l = 1 \\ \lambda nx_m + n + pn\lambda(r+2)x_m & l = 0 \end{cases}.
\end{aligned}$$

Derivations for Observation 6.19

Define $\beta = \lambda x_m$, then

$$\begin{aligned}
& & \lambda\mathbb{E}[C_{\text{cloud}}(l=1)] &> \lambda\mathbb{E}[C_{\text{cloud}}(l=0)] \\
\Leftrightarrow\quad & & n + pn\left[\lambda x_m + r\left(1 - e^{-\lambda x_m}\right)\right] &> n + pn\lambda(r+2)x_m \\
\Leftrightarrow\quad & & \beta r + \beta - r + re^{-\beta} &< 0.
\end{aligned}$$

And note that the function $g(\beta) = \beta r + \beta - r + re^{-\beta}$ is monotonically increasing as $g'(\beta) = r + 1 - re^{-\beta} > 0$ for any $r \in \mathbb{Z}^+$.

■ B.4 Proofs for single task scheduling

In this section we present the detailed analysis for the problem of optimal scheduling for a single task.

■ B.4.1 Proofs for Theorem 7.1 and Theorem 7.2

In this section we show that the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off curve is always piecewise linear, with the vertices of the piecewise linear curve corresponding to starting time vector that satisfies certain properties.

We first define the possible finishing time

$$w_{i,j} \triangleq t_i + \alpha_j, 1 \le i \le m, 1 \le j \le l$$

and the set of all possible finishing times

$$\mathcal{W} \triangleq \{w_{i,j}, 1 \le i \le m, 1 \le j \le l\}.$$

Let $k = |\mathcal{W}|$, we denote the sorted version of $\mathcal{W}$ as $\mathbf{w} = [w_{\sigma_\mathbf{t}(1)}, w_{\sigma_\mathbf{t}(2)}, \ldots, w_{\sigma_\mathbf{t}(k)}]$ such that

$$w_{\sigma_\mathbf{t}(1)} \le w_{\sigma_\mathbf{t}(2)} \le \ldots \le w_{\sigma_\mathbf{t}(k)},$$

where $\sigma_\mathbf{t}(k)$ maps the rank of the finishing time k to a tuple (i_k, j_k).

Note that

$$T = \min_{1 \le i \le m} X_i + t_i,$$

and $T \in \mathcal{W}$, we define the event

$$\mathcal{A}_{k_1,k_2} \triangleq \left\{ \overset{*}{\min}_{1 \le i \le m, 1 \le j \le l} \{t_i + X_j\} = t_{k_1} + \alpha_{k_2} \right\},$$

where $\min^*$ indicates we always choose the smallest (k_1, k_2) (by lexicographic order in k_1 and k_2) so that all the events $\{\mathcal{A}_{k_1,k_2}, 1 \le k_1 \le m, 1 \le k_2 \le l\}$ are disjoint.

Therefore,

$$\begin{aligned}
\mathbb{E}[T] &= \sum_{k_1,k_2} \mathbb{E}[T|\mathcal{A}_{k_1,k_2}] \mathbb{P}[\mathcal{A}_{k_1,k_2}] \\
&= \sum_{k_1,k_2} (t_{k_1} + \alpha_{k_2}) \mathbb{P}[\mathcal{A}_{k_1,k_2}], \qquad \text{(B.13)} \\
\mathbb{E}[C] &= \sum_{j=1}^{m} \mathbb{E}[C_j] \\
&= \sum_{j=1}^{m} \sum_{k_1,k_2} |t_{k_1} + \alpha_{k_2} - t_j|^+ \mathbb{P}[\mathcal{A}_{k_1,k_2}]. \qquad \text{(B.14)}
\end{aligned}$$

To analyze (B.13) and (B.14), we first show that the relative ordering of elements in $\mathcal{W}$ determines $\mathbb{P}[\mathcal{A}_{k_1,k_2}]$.

Lemma B.6. $\mathbb{P}[\mathcal{A}_{k_1,k_2}]$ *is independent of* $\{\alpha_j, 1 \le l \le l\}$ *given the relative ordering of elements in* $\mathcal{W}$*, i.e.,*

$$\mathbb{P}[\mathcal{A}_{k_1,k_2} \,|\, \sigma_{\mathbf{t}}] = f(\sigma_{\mathbf{t}}, k_1, k_2, p_1, \ldots, p_l), \tag{B.15}$$

where f *is some function.*

Proof. $\mathcal{A}_{k_1,k_2}$ indicates node k_1 is the first node that finishes execution, and it finishes execution after running for α_{k_2}. Define $k \triangleq \sigma_{\mathbf{t}}^{-1}(k_1, k_2)$, i.e.,

$$t_{k_1} + \alpha_{k_2} = w_{\sigma_{\mathbf{t}}(k)},$$

then

$$\begin{aligned}\mathbb{P}[\mathcal{A}_{k_1,k_2} \,|\, \sigma_{\mathbf{t}}] &= \mathbb{P}\left[\bigcap_{j\neq k_1} \{t_j + X_j\} > w_{k_1,k_2} \,\middle|\, \sigma_{\mathbf{t}}\right] \\ &= \prod_{j\neq k_1} \mathbb{P}[t_j + X_j > w_{k_1,k_2} \,|\, \sigma_{\mathbf{t}}]\,. \end{aligned} \tag{B.16}$$

Define

$$\mathcal{P}_j \triangleq \{p : \sigma_{\mathbf{t}}(i) = (j,p), i > k\}\,,$$

which is uniquely determined by $\sigma_{\mathbf{t}}$ and k, then for any $i \neq k_1$,

$$q_j \triangleq \mathbb{P}[t_j + X_j > w_{k_1,k_2} \,|\, \sigma_{\mathbf{t}}] = \sum_{p\in\mathcal{P}_j} p, \tag{B.17}$$

which is a function of k, σ_t and $\mathbf{p} = [p_1, p_2, \ldots, p_m]$.

Combing (B.16) and (B.17), we have (B.15). □

Proof of Theorem 7.1. We prove that there exists finitely many subspaces of $[0, \alpha_l]^m$ such that in each subspace, $\mathbb{E}[T]$ and $\mathbb{E}[C]$ is a linear function in $\mathbf{t}$, and thus they are piecewise linear in $\mathbf{t}$ on $[0, \alpha_l]^m$.

Define $\mathcal{B}_1(\sigma) \triangleq \{\mathbf{t} : \sigma_{\mathbf{t}} = \sigma\}$, and $(k_1, k_2) = \sigma(k)$, then the set

$$\mathcal{B}_1(\sigma) = \{\mathbf{t} : t_{1_1} + \alpha_{1_2} \le t_{2_2} + \alpha_{2_2} \le \ldots \le t_{k_1} + \alpha_{k_2}\} \tag{B.18}$$

is defined by $k-1$ inequalities, and each of this inequality partition the space $[0, \alpha_l]^n$ into two subspaces. Therefore, $\mathcal{B}_1(\sigma)$ is the intersection of $k-1$ connected subspaces, resulting itself being a subspace of $[0, \alpha_l]^m$. And it is obvious that there are only finitely many such subspaces. Therefore, by Lemma B.6 and (B.13), in each subspace $\mathcal{B}_1(\sigma)$, $\mathbb{E}[T]$ is a linear function in $\mathbf{t}$.

Regarding $\mathbb{E}[C]$, we define

$$\begin{aligned}&\mathcal{B}_2(\mathbf{b} = [b_{i,j}]_{1\le i\le m, 1\le j\le k}, \sigma) \subset \mathcal{B}_1(\sigma) \\ \triangleq &\left\{\mathbf{t} : \sigma_{\mathbf{t}} = \sigma,\ \mathbb{1}\{t_{k_1} + \alpha_{k_2} - t_i > 0\} \in \{0,1\}, (k_1,k_2) = \sigma^{-1}(j), 1 \le j \le k\right\},\end{aligned} \tag{B.19}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function. Similar to the argument above, given σ and $\mathbf{b}$, $\mathcal{B}_2(\mathbf{b},\sigma)$ corresponds to a subspace of $[0,\alpha_l]^m$ and there are only finitely many such subspaces. By Lemma B.6 and (B.14), in each subspace $\mathcal{B}_2(\mathbf{b},\sigma)$, $\mathbb{E}[C]$ is a linear function in $\mathbf{t}$.

Therefore, both $\mathbb{E}[T]$ and $\mathbb{E}[C]$ are piecewise linear functions of $\mathbf{t}$ in $[0,\alpha_l]^m$. □

Proof of Theorem 7.2. By Theorem 7.1 and the fact that J_λ is a linear combination of $\mathbb{E}[T]$ and $\mathbb{E}[C]$, the optimal $\mathbf{t}^*$ that minimizes $J_\lambda(\mathbf{t})$ is at the boundaries of two or more subspaces defined in (B.19).

Then by (B.18) and (B.19), it is not hard to see that for some j_1, j_2, j_3, j_4 and l_1, l_2, l_3, we have

$$t^*_{j_1} - t^*_{j_2} = \alpha_{l_1} - \alpha_{l_2}$$
$$t^*_{j_3} - t^*_{j_4} = \alpha_{l_3}.$$

Then it is not hard to see that given m, $\mathbf{t} = [t_1, t_2, \ldots, t_m]$, and without loss of generality, let $t_1 = 0$,

$$t^*_i \in \mathcal{V}_m,$$

where $\mathcal{V}_m$ is defined in (7.8), i.e.,

$$\mathcal{V}_m \triangleq \left\{ v : v = \sum_{j=1}^{l} \alpha_j w_j, 0 \le v \le \alpha_l, \sum_{j=1}^{l} |w_j| \le m, w_j \in \mathbb{Z} \right\}.$$

Note that an element in $\mathcal{V}_m$ is uniquely determined by $\mathbf{w} = [w_1, \ldots, w_l]$, and the number of possible $\mathbf{w}$ is

$$2^l \binom{m+l-1}{l-1}.$$

Therefore,

$$|\mathcal{V}_m| \le 2^l \binom{m+l-1}{l-1} \le [2(m+l-1)]^l,$$

which is finite given finite m and l. □

■ B.4.2 Proofs related to corner points

Proof of Theorem 7.4. Let $U_{i+1} = [u_1, u_2, \ldots, u_{k_i}]$ be the sorted version of $\mathcal{U}_{i+1}$, then $\mathbb{E}[T(\mathbf{t}')]$ and $\mathbb{E}[C(\mathbf{t}')]$ are linear in t_{i+1} over the each interval $[u_j, u_{j+1}], 1 \le j \le k_i - 1$. Therefore, the optimal $t_{i+1} \in \mathcal{U}_{i+1}$. □

■ B.4.3 Proof of Lemma 7.5

Proof of Lemma 7.5. Consider a set of m nodes on which we run the task according to the policy $\pi = [t_1, t_2, \cdots, t_m]$. Without loss of generality, we assume $t_1 = 0$. If the starting time of a node is $\alpha_l > t_j \ge \alpha_l - \alpha_1$, the earliest time it can finish execution of the task is $t + \alpha_1$. This time is greater than α_l, the latest time at which the first node started at time $t_1 = 0$ finishes the task. Thus, starting the node at time t_j only adds to the

cloud computing cost $\mathbb{E}[C]$, without reducing the latency $\mathbb{E}[T]$. Hence, any starting time $t_j \geq \alpha_l - \alpha_1$ should be replaced by α_l, which corresponds to not using that node at all.□

■ B.4.4 Proof of Theorem 7.6

Theorem 7.6 follows directly from the following lemma.

Lemma B.7. *Given P_X is a bimodal distribution and we have at most two nodes, the expected latency and total cloud computing cost satisfies that if $t_2 + \alpha_1 < \alpha_2$,*

$$\mathbb{E}[T] = \alpha_1(p_2 - p_1)p_1 + \alpha_2 p_2^2 + t_2 p_1 p_2,$$
$$\mathbb{E}[C] = \begin{cases} 2\mathbb{E}[T] - t_2(p_1^2 + p_2^2) & \text{if } t_2 < \alpha_1 \\ 2\mathbb{E}[T] - \alpha_1 p_1 - t_2 p_2 & \text{if } t_2 \geq \alpha_1 \end{cases};$$

otherwise if $t_2 + \alpha_1 \geq \alpha_2$,

$$\mathbb{E}[T] = \alpha_1 p_1 + \alpha_2 p_2$$
$$\mathbb{E}[T] = 2\mathbb{E}[T] - \alpha_1 p_1 - t_2 p_2$$

Proof. By (7.5) and (7.6) and calculation. □

■ B.4.5 Proof of Theorem 7.7

Proof of Theorem 7.7. (a) Follows from Lemma 7.5.

(b) If $\frac{\alpha_1}{\alpha_2} > \frac{1}{2}$ then by Lemma 7.5 we know that if $[0, \alpha_1]$ is suboptimal. Now suppose $\frac{\alpha_1}{\alpha_2} > \frac{1}{2}$. We know that $\pi_1 = [0, 0]$ and $\pi_2 = [0, \alpha_2]$ are the two extreme ends of the $(\mathbb{E}[C], \mathbb{E}[T])$ trade-off. If the line joining points $(\mathbb{E}[C(\pi_1)], \mathbb{E}[T(\pi_1)])$ and $(\mathbb{E}[C(\pi_2)], \mathbb{E}[T(\pi_2)])$ lies below $(\mathbb{E}[C(\pi)], \mathbb{E}[T(\pi)])$, then $\pi = [0, \alpha_1]$ will be suboptimal. Comparing the slopes of the lines gives the condition $\frac{\alpha_1}{\alpha_2} > \frac{p_1}{1+p_1}$.

(c) Policy $\pi_2 = [0, \alpha_2]$ is suboptimal if it is dominated by either $\pi = [0, \alpha_1]$ or $\pi_1 = [0, 0]$. Both π and π_1 give lower expected execution time $\mathbb{E}[T]$ than $[0, \alpha_2]$. So if one of them has expected cloud computing cost $\mathbb{E}[C]$ lower than $[0, \alpha_2]$, then it follows that $[0, \alpha_2]$ is dominated by that strategy. But the $\mathbb{E}[C]$ with starting time vector $[0, 0]$ is always greater than that of $[0, \alpha_1]$. Thus, checking if the expected cloud user cost $\mathbb{E}[C]$ with $[0, \alpha_1]$ is smaller than that for $[0, \alpha_2]$, gives the condition $\frac{\alpha_1}{\alpha_2} < \frac{2p_1 - 1}{4p_1 - 1}$ for suboptimality of $[0, \alpha_2]$.

(d) , (e), (f) For cost function $J = \lambda\mathbb{E}[T] + (1 - \lambda)\mathbb{E}[C]$, the constant cost contour is a line with slope $-\frac{1-\lambda}{\lambda}$. As we increase J, the contour line shifts upward until it hits the $(\mathbb{E}[C], \mathbb{E}[T])$ trade-off. The point where it meets the $(\mathbb{E}[C], \mathbb{E}[T])$ trade-off corresponds to the optimal policy. In $\mathcal{R}_1$, policy $\pi_2 = [0, \alpha_2]$ is optimal if the slope of the line joining $(\mathbb{E}[C(\pi_1)], \mathbb{E}[T(\pi_1)])$ and $(\mathbb{E}[C(\pi_2)], \mathbb{E}[T(\pi_2)])$ is less than or equal to $-\frac{1-\lambda}{\lambda}$. We can simplify and show that the slope of the line is $-\tau_1$. The result follows from this. Similarly, the slope of the line joining $[0, 0]$, $[0, \alpha_1]$ is $-\tau_2$, and that of the line joining $[0, \alpha_1$ and $[0, \alpha_2]$ is $-\tau_3$. Comparing the slope of the contour, $-\frac{1-\lambda}{\lambda}$ with these slopes gives the conditions of optimality for each of the policies.

■ B.4.6 Proof of Theorem 7.8

Proof of Theorem 7.8. By Theorem 7.4, for a policy that replicate once, the optimal launch time t satisfies $t = \alpha_1$. Hence

$$\mathbf{t}_d = [t_1 = 0, \ldots, t_{m-d} = 0, t_{m-d+1} = \alpha_1, \ldots, t_m = \alpha_1],$$

which implies

$$\begin{aligned}\mathbb{P}[T > \alpha_1] &= p_2^{m-d},\\ \mathbb{P}[T > 2\alpha_1] &= p_2^m.\end{aligned}$$

Therefore,

$$\begin{aligned}\mathbb{P}[T = \alpha_1] &= 1 - p_2^{m-d},\\ \mathbb{P}[T = 2\alpha_1] &= p_2^{m-d} - p_2^m,\\ \mathbb{P}[T = \alpha_2] &= p_2^m,\end{aligned}$$

and

$$\begin{aligned}\mathbb{E}[T(d)] &\triangleq \mathbb{E}[T(\mathbf{t}_d)]\\ &= \alpha_1 \cdot \left[1 + p_2^{m-d} - 2p_2^m\right] + \alpha_2 \cdot p_2^m\\ \mathbb{E}[C(d)] &\triangleq \mathbb{E}[C(\mathbf{t}_d)]\\ &= (m-d)\mathbb{E}[T(d)] + d\left(\mathbb{E}[T(d)] - \alpha_1\right)\\ &= m\mathbb{E}[T(d)] - \alpha_1 d\\ J_\lambda(d) &\triangleq J_\lambda(\mathbf{t}_d)\\ &= (\lambda + m\bar{\lambda})\mathbb{E}[T(d)] - \bar{\lambda}\alpha_1 d,\end{aligned}$$

where $\bar{\lambda} \triangleq 1 - \lambda$. Therefore,

$$\begin{aligned}J_\lambda(d+1) - J_\lambda(d) &= (\lambda + m\bar{\lambda})(\mathbb{E}[T(d+1)] - \mathbb{E}[T(d)]) - \bar{\lambda}\alpha_1\\ &= (\lambda + m\bar{\lambda})\alpha_1\left(p_2^{m-d-1} - p_2^{m-d}\right) - \bar{\lambda}\alpha_1\\ &= (\lambda + m\bar{\lambda})\alpha_1 p_2^{m-d-1}p_1 - \bar{\lambda}\alpha_1.\end{aligned}$$

Solve

$$J_\lambda(d+1) - J_\lambda(d) > 0,$$

we have

$$p_2^{m-d-1}p_1 > \frac{\bar{\lambda}}{(\lambda + m\bar{\lambda})},$$

where the left hand side increases as d increases from 0 to $n-1$. Therefore, when

$$d > a \triangleq m - 1 - \frac{\log\bar{\lambda} - \log\left((\lambda + m\bar{\lambda})p_1\right)}{\log(p_2)}, \tag{B.20}$$

$$J_\lambda(d+1) > J_\lambda(d),$$

and otherwise

$$J_\lambda(d+1) \leq J_\lambda(d).$$

Note that when

$$p_2^{m-1} p_1 > \frac{\bar{\lambda}}{(\lambda + m\bar{\lambda})},$$

$a < 0$, and when

$$\frac{\bar{\lambda}}{(\lambda + m\bar{\lambda})} > p_1,$$

$a > m-1$. Therefore,

$$d = \begin{cases} 0 & \text{when } \bar{\lambda}/(\lambda + m\bar{\lambda}) < p_2^{m-1} p_1 \\ m-1 & \text{when } \bar{\lambda}/(\lambda + m\bar{\lambda}) > p_1 \\ \lceil a \rceil - 1, \text{ or } \lceil a \rceil & \text{otherwise} \end{cases}. \qquad \square$$

■ B.5 Proofs for multi-task scheduling

■ B.5.1 Proof of Theorem 7.9

Proof. We prove the statement by showing an example that a scheduling policy that takes the interaction of task latencies into account (joint policy) is better than a scheduling each task independently (separate policy).

Suppose we have two tasks and 4 computing nodes. The service time distribution of each node is bimodal, taking values α_1 and $\alpha_2 > \alpha_1$ with probability p_1 and $1 - p_1$ respectively. Assume $2\alpha_1 < \alpha_2$.

Separate Policy

Consider a policy π_s where we choose the optimal scheduling policy separately for each task. We can follow the analysis of the bimodal two-node case in Appendix B.4.5 as a guideline to choose the optimal policy for each task.

Suppose the policy $[0, \alpha_2]$ is optimal for a given cost function. For this to be true, the parameters α_1, α_2 and p_1 need to satisfy,

$$\frac{\alpha_1}{\alpha_2} > \frac{2p_1 - 1}{4p_1 - 1} \tag{B.21}$$

If we run each task on two nodes using the policy $[0, \alpha_2]$, the expected total latency and cloud computing cost are,

$$\begin{aligned} \mathbb{E}\left[T\left(\pi_s\right)\right] &= p_1^2 \alpha_1 + (1 - p_1^2)\alpha_2, \\ \mathbb{E}\left[C_{\text{cloud}}\left(\pi_s\right)\right] &= 2p_1^2 \alpha_1 + 2p_1(1 - p_1)(\alpha_1 + \alpha_2) + 2(1 - p_1^2)\alpha_2. \end{aligned}$$

Joint Policy

Consider a joint policy π_d where we start with each task according to policy $[0, \alpha_2]$. If

task 1 (task 2) is served by its node at time α_1, we start the execution the task 2 (task 1) on an additional node at time α_1.

Using this joint policy the performance metrics are given by

$$\mathbb{E}[T(\pi_d)] = p_1^2\alpha_1 + 2p_1^2(1-p_1)(2\alpha_1) + (1-p_1)^2(2p_1+1)\alpha_2,$$
$$\mathbb{E}[C(\pi_d)] = p_1^2(2\alpha_1) + 2p_1^2(1-p_1)(3\alpha_1) + (1-p_1)^2(2p_1+1)(2\alpha_2).$$

We can show that for $2\alpha_1 < \alpha_2$, $\mathbb{E}[T(\pi_d)] < \mathbb{E}[T(\pi_s)]$. Now let us find the condition for $\mathbb{E}[C(\pi_d)] < \mathbb{E}[C(\pi_s)]$.

$$\begin{aligned} & \mathbb{E}[C(\pi_d)] < \mathbb{E}[C(\pi_s)] \\ \Rightarrow \qquad & \frac{\alpha_1}{\alpha_2} < \frac{2p_1-1}{3p_1-1}. \end{aligned}$$

Thus, the joint policy gives strictly lower cost $J_\lambda = \lambda\mathbb{E}[C] + (1-\lambda)\mathbb{E}[T]$ than the separate policy for any λ if

$$\frac{2p_1-1}{4p_1-1} < \frac{\alpha_1}{\alpha_2} < \frac{2p_1-1}{3p_1-1}. \tag{B.22}$$

□

Appendix C

Results and proofs for Part III

C.1 Geometry of permutation spaces

In this section we provide results on the geometry of the permutation space that are useful in deriving the rate-distortion bounds.

We first define D-balls centered at $\sigma \in \mathcal{S}_n$ with radius D under distance $d(\cdot,\cdot)$ and their maximum sizes:

$$B_d(\sigma, D) \triangleq \{\pi : d(\pi, \sigma) \le D\}, \tag{C.1}$$

$$N_d(D) \triangleq \max_{\sigma \in \mathcal{S}_n} |B_d(\sigma, D)|. \tag{C.2}$$

Let $B_\tau(\sigma, D)$, $B_{\ell_1}(\sigma, D)$ and $B_{\mathbf{x},\ell_1}(\sigma, D)$ be the balls that correspond to the Kendall tau distance, ℓ_1 distance of the permutations, and ℓ_1 distance of the inversion vectors, and $N_\tau(D)$, $N_{\ell_1}(D)$, and $N_{\mathbf{x},\ell_1}(D)$ be their maximum sizes respectively.

Note that (9.11) implies $B_\tau(\sigma, D) \subset B_{\mathbf{x},\ell_1}(\sigma, D)$ and thus $N_\tau(D) \le N_{\mathbf{x},\ell_1}(D)$. Below we establish upper bounds for $N_{\mathbf{x},\ell_1}(D)$ and $N_\tau(D)$, which are useful for establishing converse results later.

Lemma C.1. *For $0 \le D \le n$,*

$$N_\tau(D) \le \binom{n+D-1}{D}. \tag{C.3}$$

Proof. Let the number of permutations in $\mathcal{S}_n$ with at most k inversions be $T_n(d) \triangleq \sum_{k=0}^{d} K_n(k)$, where $K_n(k)$ is defined in (9.1). Since $\mathcal{X}(\mathcal{S}_n, d_\tau)$ is a regular metric space,

$$N_\tau(D) = T_n(D),$$

which is noted in several references such as [75]. An expression for $K_n(D)$ (and thus $T_n(D)$) for $D \le n$ appears in [75] (see [123] also). The following bound is weaker but sufficient in our context.

By induction, or [124], $T_n(D) = K_{n+1}(D)$ when $D \le n$. Then noting that for $k < n$, $K_n(k) = K_n(k-1) + K_{n-1}(k)$ [75, Section 5.1.1] and for any $n \ge 2$,

$$K_n(0) = 1, \quad K_n(1) = n-1, \quad K_n(2) = \binom{n}{2} - 1,$$

by induction, we can show that when $1 \le k < n$,

$$K_n(k) \le \binom{n+k-2}{k}. \tag{C.4}$$

□

The product structure of $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$ leads to a simpler analysis of the upper bound of $N_{\mathbf{x},\ell_1}(D)$.

Lemma C.2. *For* $0 \le D \le n(n-1)/2$,

$$N_{\mathbf{x},\ell_1}(D) \le 2^{\min\{n,D\}} \binom{n+D}{D}. \tag{C.5}$$

Proof. For any $\sigma \in \mathcal{S}_n$, let $\mathbf{x} = \mathbf{x}_\sigma \in \mathcal{G}_n$, then

$$|B_{\mathbf{x},\ell_1}(D)| = \sum_{r=0}^{D} |\{\mathbf{y} \in \mathcal{G}_n : d_{\ell_1}(\mathbf{x},\mathbf{y}) = r\}|.$$

Let $\mathbf{d} \triangleq |\mathbf{x} - \mathbf{y}|$, and $Q(n,r)$ be the number of integer solutions of the equation $z_1 + z_2 + \ldots + z_n = r$ with $z_i \ge 0, 0 \le i \le n$, then it is well known [125, Section 1.2] that

$$Q(n,r) = \binom{n+r-1}{r},$$

and it is not hard to see that the number of such $\mathbf{d} = [d_1, d_2, \ldots, d_{n-1}]$ that satisfies $\sum_{i=1}^{n-1} d_i = r$ is upper bounded by $Q(n-1,r)$. Given $\mathbf{x}$ and $\mathbf{d}$, at most $m \triangleq \min\{D, n\}$ elements in $\{y_i, 0 \le i \le n\}$ correspond to $y_i = x_i \pm d_i$. Therefore, for any $\mathbf{x}$, $|\{\mathbf{y} \in \mathcal{G}_n : d_{\ell_1}(\mathbf{x},\mathbf{y}) = r\}| \le 2^m Q(n,r)$ and hence

$$|B_{\ell_1}(\mathbf{x}, D)| \le \sum_{r=0}^{D} 2^m Q(n,r) = 2^m \binom{n+D}{D}. \qquad \square$$

Below we upper bound $\log N_\tau(D)$ and $\log N_{\mathbf{x},\ell_1}(D)$ for small, moderate and large D regimes in Lemmas C.3 to C.5 respectively.

Lemma C.3 (Small distortion regime). *When* $D = an^\delta, 0 < \delta \le 1$ *and* $a > 0$ *is a constant,*

$$\log N_\tau(D) \le \begin{cases} a(1-\delta)n^\delta \log n + O\left(n^\delta\right), & 0 < \delta < 1 \\ n\left[\log \frac{(1+a)^{1+a}}{a^a}\right] + o(n), & \delta = 1 \end{cases}, \tag{C.6}$$

$$\log N_{\mathbf{x},\ell_1}(D) \le \begin{cases} a(1-\delta)n^\delta \log n + O\left(n^\delta\right), & 0 < \delta < 1 \\ n\left[2 + \log \frac{(1+a)^{1+a}}{a^a}\right] + o(n), & \delta = 1 \end{cases}. \tag{C.7}$$

Proof. To upper bound $N_\tau(D)$, when $0 < \delta < 1$, we apply Stirling's approximation to (C.3) to have

$$\log \binom{n+D-1}{D} = n \log \frac{n-1+D}{n-1} + D \log \frac{n-1+D}{D} + O(\log n).$$

Substituting $D = an^\delta$, we obtain (C.6). When $\delta = 1$, the result follows from (9) in [126, Section 4]. The upper bound on $N_{\mathbf{x},\ell_1}(D)$ can be obtained similarly via (C.5). □

Lemma C.4 (Moderate distortion regime). *Given $D = \Theta\left(n^{1+\delta}\right)$, $0 < \delta \leq 1$, then*

$$\log N_\tau(D) \leq \log N_{\mathbf{x},\ell_1}(D) \leq \delta n \log n + O(n). \tag{C.8}$$

Proof. Apply Stirling's approximation to (C.5) and substitute $D = \Theta\left(n^{1+\delta}\right)$. □

Remark C.1. *It is possible to obtain tighter lower and upper bounds for $\log N_\tau(D)$ and $\log N_{\mathbf{x},\ell_1}(D)$ based on results in [123].*

Lemma C.5 (Large distortion regime). *Given $D = bn(n-1) \in \mathbb{Z}^+$, then*

$$\log N_\tau(D) \leq \log N_{\mathbf{x},\ell_1}(D) \leq n \log(2ben) + O(\log n). \tag{C.9}$$

Proof. Substitute $D = bn(n-1)$ into (C.5). □

■ C.2 Proofs on the relationships among distortion measures

Lemma C.6.

$$\max_{\sigma \in \mathcal{S}_n, \sigma' \in \mathcal{S}_n} d_{\ell_1}(\sigma, \sigma') = \lfloor n^2/2 \rfloor.$$

Proof. Note that $d_{\ell_1}(\sigma, \sigma')$ is maximized when $\sigma = [1, 2, \ldots, n]$ and $\sigma = [n, n-1, \ldots, 1]$. Therefore, when n is even,

$$\max_{\sigma \in \mathcal{S}_n, \sigma' \in \mathcal{S}_n} d_{\ell_1}(\sigma, \sigma') = \begin{cases} \sum_{k=1}^{n/2} (2k-1) = n^2/2 & n \text{ is even}, \\ \sum_{k=1}^{(n-1)/2} 2k = (n^2-1)/2. & n \text{ is odd}. \end{cases}$$ □

■ C.2.1 Proof of Theorem 9.1

Lemma C.7. *For any $\pi \in \mathcal{S}_n$, let σ be a permutation chosen uniformly from $\mathcal{S}_n$, and $X_{\ell_1} \triangleq d_{\ell_1}(\pi, \sigma)$, then*

$$\mathbb{E}[X_{\ell_1}] = \frac{n^2-1}{3} \quad \mathrm{Var}[X_{\ell_1}] = \frac{2n^3}{45} + O\left(n^2\right). \tag{C.10}$$

Proof.

$$\begin{aligned}
\mathbb{E}\left[X_{\ell_1}\right] &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}|i-j| = \frac{2}{n}\sum_{i=1}^{n}\sum_{j=1}^{i}|i-j| \\
&= \frac{2}{n}\sum_{i=1}^{n}\sum_{j'=0}^{i-1} j' = \frac{1}{n}\sum_{i=1}^{n}(i^2-i) \\
&= \frac{1}{n}\left(\sum_{i=1}^{n} i^2 - \sum_{i=1}^{n} i\right) \\
&= \frac{1}{n}\left(\frac{2n^3+3n^2+n}{6} - \frac{n^2+n}{2}\right) \\
&= \frac{n^2-1}{3}.
\end{aligned}$$

And $\mathrm{Var}\left[X_{\ell_1}\right]$ can be derived similarly [112, Table 1]. □

Proof for Theorem 9.1. For any $c > 0$, $cn \cdot d_{\ell_\infty}(\pi,\sigma) \le cn(n-1)$, and for any $c_1 < 1/3$, Lemma C.7 and Chebyshev inequality indicate $\mathbb{P}\left[d_{\ell_1}(\pi,\sigma) < c_1 n(n-1)\right] = O(1/n)$. Therefore,

$$\begin{aligned}
&\mathbb{P}\left[d_{\ell_1}(\pi,\sigma) \ge c_1 n \cdot d_{\ell_\infty}(\pi,\sigma)\right] \\
\ge\ &\mathbb{P}\left[d_{\ell_1}(\pi,\sigma) \ge c_1 n(n-1)\right] \\
=\ &1 - \mathbb{P}\left[d_{\ell_1}(\pi,\sigma) < c_1 n(n-1)\right] \\
=\ &1 - O\left(1/n\right).
\end{aligned}$$

□

■ C.2.2 Proof of Theorem 9.3

Lemma C.8. *For any two permutations π, σ in $\mathcal{S}_n$ such that $d_{\mathbf{x},\ell_1}(\pi,\sigma) = 1$, $d_\tau(\pi,\sigma) \le n-1$.*

Proof. Let $\mathbf{x}_\pi = [a_2, a_3, \ldots, a_n]$ and $\mathbf{x}_\sigma = [b_2, b_3, \ldots, b_n]$, then without loss of generality, we have for a certain $2 \le k \le n$,

$$a_i = \begin{cases} b_i & i \ne k \\ b_i + 1 & i = k. \end{cases}$$

Let π' and σ' be permutations in $\mathcal{S}_{n-1}$ with element k removed from π and σ correspondingly, then $\mathbf{x}_{\pi'} = \mathbf{x}_{\sigma'}$, and hence $\pi' = \sigma'$. Therefore, the Kendall tau distance between σ and π is determined only by the location of element k in σ and π, which is at most $n-1$.□

Proof of Theorem 9.3. It is known that (see, e.g., [127, Lemma 4])

$$d_{\ell_1}(\mathbf{x}_{\pi_1}, \mathbf{x}_{\pi_2}) \le d_\tau(\pi_1, \pi_2).$$

Furthermore, the proof of [127, Lemma 4] indicates that for any two permutation π_1 and π_2 with $k = d_{\mathbf{x},\ell_1}(\pi_1, \pi_2)$, let $\sigma_0 \triangleq \pi_1$ and $\sigma_k \triangleq \pi_2$, then there exists a sequence of permutations $\sigma_1, \sigma_2, \ldots, \sigma_{k-1}$ such that $d_{\mathbf{x},\ell_1}(\sigma_i, \sigma_{i+1}) = 1, 0 \le i \le k-1$. Then

$$\begin{aligned} d_\tau(\pi_1, \pi_2) &= \sum_{i=0}^{k-1} d_\tau(\sigma_i, \sigma_{i-1}) \\ &\overset{(a)}{\le} \sum_{i=0}^{k-1} (n-1) = (n-1) d_{\mathbf{x},\ell_1}(\pi_1, \pi_2), \end{aligned}$$

where (a) is due to Lemma C.8. □

■ C.2.3 Proof of Theorem 9.4

To prove Theorem 9.4, we analyze the mean and variance of the Kendall tau distance and inversion-ℓ_1 distance between a permutation in $\mathcal{S}_n$ and a randomly selected permutation, in Lemma C.9 and Lemma C.10 respectively.

Lemma C.9. *For any $\pi \in \mathcal{S}_n$, let σ be a permutation chosen uniformly from $\mathcal{S}_n$, and $X_\tau \triangleq d_\tau(\pi, \sigma)$, then*

$$\mathbb{E}[X_\tau] = \frac{n(n-1)}{4}, \tag{C.11}$$

$$\mathrm{Var}[X_\tau] = \frac{n(2n+5)(n-1)}{72}. \tag{C.12}$$

Proof. Let σ' be another permutation chosen independently and uniformly from $\mathcal{S}_n$, then we have both $\pi\sigma^{-1}$ and $\sigma'\sigma^{-1}$ are uniformly distributed over $\mathcal{S}_n$.

Note that Kendall tau distance is right-invariant [111], then $d_\tau(\pi, \sigma) = d_\tau(\pi\sigma^{-1}, \mathrm{Id})$ and $d_\tau(\sigma', \sigma) = d_\tau(\sigma'\sigma^{-1}, \mathrm{Id})$ are identically distributed, and hence the result follows [112, Table 1] and [75, Section 5.1.1]. □

Lemma C.10. *For any $\pi \in \mathcal{S}_n$, let σ be a permutation chosen uniformly from $\mathcal{S}_n$, and $X_{\mathbf{x},\ell_1} \triangleq d_{\mathbf{x},\ell_1}(\pi, \sigma)$, then*

$$\mathbb{E}[X_{\mathbf{x},\ell_1}] > \frac{n(n-1)}{8},$$

$$\mathrm{Var}[X_{\mathbf{x},\ell_1}] < \frac{(n+1)(n+2)(2n+3)}{6}.$$

Proof. It is not hard to see that when σ is a permutation chosen uniformly from $\mathcal{S}_n$, $\mathbf{x}_\sigma(i)$ is uniformly distributed in $[0:i]$, $1 \le i \le n-1$. Therefore, $X_{\mathbf{x},\ell_1} = \sum_{i=1}^{n-1} |a_i - U_i|$, where $U_i \sim \mathsf{Unif}([0:i])$ and $a_i \triangleq \mathbf{x}_\pi(i)$. Let $V_i = |a_i - U_i|$, $m_1 = \min\{i - a_i, a_i\}$ and

$m_2 = \max\{i - a_i, a_i\}$, then

$$\mathbb{P}[V_i = d] = \begin{cases} 1/(i+1) & d = 0 \\ 2/(i+1) & 1 \le d \le m_1 \\ 1/(i+1) & m_1 + 1 \le d \le m_2 \\ 0 & \text{otherwise.} \end{cases}$$

Hence,

$$\begin{aligned} \mathbb{E}[V_i] &= \sum_{d=1}^{m_1} d\frac{2}{i+1} + \sum_{d=m_1+1}^{m_2} d\frac{1}{i+1} \\ &= \frac{2(1+m_1)m_1 + (m_2+m_1+1)(m_2-m_1)}{2(i+1)} \\ &= \frac{1}{2(i+1)}(m_1^2 + m_2^2 + i) \\ &\ge \frac{1}{2(i+1)}\left(\frac{(m_1+m_2)^2}{2} + i\right) = \frac{i(i+2)}{4(i+1)} > \frac{i}{4}, \\ \operatorname{Var}[V_i] &\le \mathbb{E}\left[V_i^2\right] \le \frac{2}{i+1}\sum_{d=0}^{i} d^2 \le (i+1)^2. \end{aligned}$$

Then,

$$\begin{aligned} \mathbb{E}[X_{\mathbf{x},\ell_1}] &= \sum_{i=1}^{n-1} \mathbb{E}[V_i] > \frac{n(n-1)}{8}, \\ \operatorname{Var}[X_{\mathbf{x},\ell_1}] &= \sum_{i=1}^{n-1} \operatorname{Var}[V_i] < \frac{(n+1)(n+2)(2n+3)}{6}. \end{aligned}$$

□

With Lemma C.9 and Lemma C.10, now we show that the event that a scaled version of the Kendall tau distance is larger than the inversion-ℓ_1 distance is unlikely.

Proof for Theorem 9.4. Let $c_2 = 1/3$, let $t = n^2/7$, then noting

$$\begin{aligned} t &= \mathbb{E}[c \cdot X_\tau] + \left|\Theta\left(\sqrt{n}\right)\right| \operatorname{Std}[X_\tau] \\ &= \mathbb{E}[X_{\mathbf{x},\ell_1}] - \left|\Theta\left(\sqrt{n}\right)\right| \operatorname{Std}[X_{\mathbf{x},\ell_1}], \end{aligned}$$

by Chebyshev inequality,

$$\begin{aligned} \mathbb{P}[c \cdot X_\tau > X_{\mathbf{x},\ell_1}] &\le \mathbb{P}[c \cdot X_\tau > t] + \mathbb{P}[X_{\mathbf{x},\ell_1} < t] \\ &\le O(1/n) + O(1/n) = O(1/n). \end{aligned}$$

The general case of $c_2 < 1/2$ can be proved similarly. □

■ C.3 Proofs on the rate distortion functions

■ C.3.1 Proof of Theorem 9.5

Proof. Statement 1 follows from (9.8).

Statement 2 and 3 follow from Theorem 9.2. For statement 2, let the encoding mapping for the (n, D_n) source code in $\mathcal{X}(\mathcal{S}_n, d_{\ell_1})$ be f_n and the encoding mapping in $\mathcal{X}(\mathcal{S}_n, d_\tau)$ be g_n, then

$$g_n(\pi) = \left[f_n(\pi^{-1})\right]^{-1}$$

is a (n, D_n) source code in $\mathcal{X}(\mathcal{S}_n, d_\tau)$. The proof for Statement 3 is similar.

Statement 4 follow directly from (9.11).

For Statement 5, define

$$\mathcal{B}_n(\pi) \triangleq \{\sigma : c_1 \cdot n \cdot d_{\ell_\infty}(\sigma, \pi) \leq d_{\ell_1}(\sigma, \pi)\},$$

then Theorem 9.1 indicates that

$$|\mathcal{B}_n(\pi)| = (1 - O(1/n))n!.$$

Let $\bar{\mathcal{C}}'_n$ be the (n, D_n) source code for $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$, π_σ be the codeword for σ in $\mathcal{C}'_n$, then by Theorem 9.1,

$$\begin{aligned}
\mathbb{E}\left[d_{\ell_\infty}(\pi_\sigma, \sigma)\right] &= \frac{1}{n!}\sum_{\sigma\in\mathcal{S}_n} d_{\ell_\infty}(\sigma, \pi_\sigma) \\
&= \frac{1}{n!}\left[\sum_{\sigma\in\mathcal{B}_n(\pi_\sigma)} d_{\ell_\infty}(\sigma, \pi_\sigma) + \sum_{\sigma\in\mathcal{S}_n\setminus\mathcal{B}_n(\pi_\sigma)} d_{\ell_\infty}(\sigma, \pi_\sigma)\right] \\
&\leq \frac{1}{n!}\left[\sum_{\sigma\in\mathcal{B}_n(\pi_\sigma)} d_{\ell_1}(\sigma, \pi_\sigma) + \sum_{\sigma\in\mathcal{S}_n\setminus\mathcal{B}_n(\pi_\sigma)} n\right] \\
&\leq D_n/(nc_1) + O(1/n)\, n = D_n/(nc_1) + O(1).
\end{aligned}$$

For Statement 6, similar to the proof of statement 5, define

$$\mathcal{A}_n(\pi) \triangleq \{\sigma : c_2 \cdot d_\tau(\sigma, \pi) \leq d_{\mathbf{x},\ell_1}(\sigma, \pi)\}$$

then Theorem 9.4 indicates that $|\mathcal{A}_n(\pi)| = (1 - O(1/n))n!$. Let $\bar{\mathcal{C}}'_n$ be the (n, D_n) source code for $\mathcal{X}(\mathcal{S}_n, d_{\mathbf{x},\ell_1})$ and σ be a permutation chosen uniformly from $\mathcal{S}_n$, then let π_σ be

the codeword for σ in $\bar{\mathcal{C}}'_n$,

$$\begin{aligned}
&\mathbb{E}\left[d_\tau\left(\pi_\sigma, \sigma\right)\right] \\
&= \frac{1}{n!} \sum_{\sigma \in \mathcal{S}_n} d_\tau\left(\sigma, \pi_\sigma\right) \\
&= \frac{1}{n!} \left[\sum_{\sigma \in \mathcal{A}_n(\pi_\sigma)} d_\tau\left(\sigma, \pi_\sigma\right) + \sum_{\sigma \in \mathcal{S}_n \setminus \mathcal{A}_n(\pi_\sigma)} d_\tau\left(\sigma, \pi_\sigma\right) \right] \\
&\leq \frac{1}{n!} \left[\sum_{\sigma \in \mathcal{A}_n(\pi_\sigma)} d_{\mathbf{x},\ell_1}\left(\sigma, \pi_\sigma\right)/c_2 + \sum_{\sigma \in \mathcal{S}_n \setminus \mathcal{A}_n(\pi_\sigma)} n^2/2 \right] \\
&\leq D_n/c_2 + O\left(1/n\right) n^2 = D_n/c_2 + O\left(n\right). \qquad \square
\end{aligned}$$

■ C.3.2 Proof of Theorem 9.6

We prove Theorem 9.6 by achievability and converse.

Achievability

The achievability for all permutation spaces of interest under both *worst-case distortion* and *average-case distortion* are established via the explicit code constructions in Section 9.5.

Converse

For the converse, we show by contradiction that under average-case distortion, if the rate is less than $1-\delta$, then the average distortion is larger than D_n. Therefore, $\bar{R} \geq 1-\delta$, and hence $\hat{R} \geq \bar{R} \geq 1-\delta$.

When $\delta = 1$, $\bar{R} = \hat{R} = 0$. When $0 \leq \delta < 1$, for any $0 < \varepsilon < 1-\delta$ and any codebook $\bar{\mathcal{C}}_n$ with size such that

$$\log\left|\bar{\mathcal{C}}_n\right| = (1-\delta-\varepsilon) n \log n + O\left(n\right), \tag{C.13}$$

from (9.6), when $D_n = \Theta\left(n^{1+\delta}\right)$ or $D_n = O\left(n\right)$,

$$N_{\ell_1}\left(2D_n\right)\left|\bar{\mathcal{C}}_n\right| \leq N_\tau\left(2D_n\right)\left|\bar{\mathcal{C}}_n\right| \leq N_{\mathbf{x},\ell_1}\left(2D_n\right)\left|\bar{\mathcal{C}}_n\right| \overset{(a)}{\leq} n!/2;$$

when $D_n = \Theta\left(n^{\delta}\right)$ or $D_n = O\left(1\right)$,

$$N_{\ell_\infty}\left(2D_n\right)\left|\bar{\mathcal{C}}_n\right| \leq N_{\mathbf{x},\ell_1}\left(2D_n n\right)\left|\bar{\mathcal{C}}_n\right| \leq n!/2$$

when n sufficiently large, where (a) follows from (C.8).

Therefore, given $\bar{\mathcal{C}}_n$, there exists at least $n!/2$ permutations in $\mathcal{S}_n$ that has distortion larger than $2D_n$, and hence the average distortion w.r.t. uniform distribution over $\mathcal{S}_n$ is larger than D_n.

Therefore, for any codebook with size indicated in (C.13), we have average distortion larger than D_n. Therefore, any (n, D_n) code must satisfy $\hat{R} \geq \bar{R} \geq 1-\delta$.

■ C.4 Proofs on Mallows Model

■ C.4.1 Proof of Lemma 9.13

Proof. When $q = 1$ the Mallows model reduces to the uniform distribution on the permutation space. When $q \neq 1$, let $X^n = [X_1, X_2, \ldots, X_n]$ be the inversion vector, and denote a geometric random variable by G and a geometric random variable truncated at k by G_k. Define

$$E_k = \begin{cases} 0 & G \leq k \\ 1 & \text{o.w.} \end{cases},$$

then $\mathbb{P}[E_k = 0] = Q_k = 1 - q^{k+1}$. Note

$$\begin{aligned} H(G_k, E) &= H(G|E_k) + H(E_k) \\ &= H(E_k|G) + H(G) \\ &= H(G) \end{aligned}$$

and

$$\begin{aligned} H(G|E_k) &= H(G|E_k = 0) Q_k \\ &\quad + H(G|E_k = 1)(1 - Q_k) \\ &= H(G_k) Q_k + H(G)(1 - Q_k), \end{aligned}$$

we have

$$H(G_k) = H_b(q)/(1-q) - H_b(Q_k)/Q_k.$$

Then

$$\begin{aligned} H(\mathcal{M}(q)) &= \sum_{k=0}^{n-1} H(G_k) \\ &= \frac{nH_b(q)}{1-q} - \sum_{k=1}^{n} \frac{H_b\left(q^k\right)}{1-q^k}. \end{aligned}$$

It can be shown via algebraic manipulations that

$$\sum_{k=1}^{n} H_b\left(q^k\right) \leq \frac{2q - q^2}{(1-q)^2} = \Theta(1),$$

therefore

$$H(\mathcal{M}(q)) = \frac{nH_b(q)}{1-q} - \Theta(1). \qquad \square$$

■ C.4.2 Proof of Lemma 9.14

We first show an upper bound $K_n(k)$ (cf. (9.1) for definition), the number of permutations with k inversion in $\mathcal{S}_n$.

Lemma C.11 (Bounds on $K_n(k)$). *For $k = cn$,*

$$K_n(k) \leq \frac{1}{\sqrt{2\pi nc/(1+c)}} 2^{n(1+c)H_b(1/(1+c))}.$$

Proof. By definition, $K_n(k)$ equals to the number of non-negative integer solutions of the equation $z_1 + z_2 + \ldots + z_{n-1} = k$ with $0 \leq z_i \geq i, 1 \leq i \leq n-1$. Then similar to the derivations in the proof of Lemma C.2,

$$K_n(k) < Q(n-1,k) = \binom{n+k-2}{k}.$$

Finally, applying the bound [128]

$$\binom{n}{pn} \leq \frac{2^{nH_b(p)}}{\sqrt{2\pi np(1-p)}}$$

completes the proof. □

Proof of Lemma 9.14. Note

$$d_\tau(\sigma, \mathrm{Id}) = d_{\mathbf{x},\ell_1}(\sigma, \mathbf{0}).$$

Therefore,

$$\sum_{\sigma \in \mathcal{S}_n, d_\tau(\sigma,\mathrm{Id}) \geq r_0} \mathbb{P}[\sigma] = \frac{1}{Z_q} \sum_{r=r_0}^{\binom{n}{2}} q^r K_n(r).$$

And Lemma C.11 indicates for any $r = cn$,

$$q^r K_n(r) \leq \frac{2^{n\left[(1+c)H_b\left(\frac{1}{1+c}\right) - c\log_2 \frac{1}{q}\right]}}{\sqrt{2\pi nc/(1+c)}}.$$

Define

$$E(c,q) \triangleq \left[(1+c)H_b\left(\frac{1}{1+c}\right) - c\log_2 \frac{1}{q}\right],$$

then for any $\varepsilon > 0$, there exits c_0 such that for any $c \geq c_0(q)$, $E(c, q) < -\varepsilon$. Therefore, let $r_0 \geq c_0 n$,

$$\begin{aligned}\sum_{\sigma \in \mathcal{S}_n, d_\tau(\sigma, \mathrm{Id}) \geq r_0} \mathbb{P}[\sigma] &\leq \frac{1}{\sqrt{2\pi n c/(1+c)}} \frac{1}{Z_q} \sum_{r=r_0}^{\binom{n}{2}} 2^{-n\varepsilon} \\ &\to 0\end{aligned}$$

as $n \to \infty$. □

List of Notations

$\mathrm{DA}(\cdot)$ domain of attraction 81

$\omega(F)$ upper end point of a cumulative distribution function F 71

$\mathcal{S}_n$ symmetric group of n elements, i.e., the set of all possible permutations for elements $1, 2, \ldots, n$ 108

Id identity permutation 108

$\mathbf{x}_\sigma$ inversion vector 116

$d_{\mathbf{x},\ell_1}(\cdot,\cdot)$ inversion-ℓ_1 distance 116

$\lesssim$ less than after scaling 118

Index

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948. (Cited on page 17.)

[2] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38, no. 8, p. 4, Apr. 1965. (Cited on page 23.)

[3] ITRS, "2010 technology roadmap for semiconductors update," The International Technology Roadmap for Semiconductors, Tech. Rep., 2010. [Online]. Available: http://www.itrs.net/Links/2010ITRS/Home2010.htm (Cited on page 23.)

[4] L. Latif, "AMD claims 20nm transition signals the end of moore's law—economic viability comes into question," Online at http://www.theinquirer.net/inquirer/news/2258444/amd-claims-20nm-transition-signals-the-end-of-moores-law, Apr. 2013, accessed: 2013-05-21. [Online]. Available: http://www.theinquirer.net/inquirer/news/2258444/amd-claims-20nm-transition-signals-the-end-of-moores-law (Cited on page 23.)

[5] K. Kuhn, M. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. Ma, A. Maheshwari, and S. Mudanai, "Process technology variation," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2197–2208, 2011. (Cited on page 23.)

[6] H. Lundin, "Characterization and correction of Analog-to-Digital converters," dissertation, KTH, 2005. (Cited on page 25.)

[7] M. Flynn, C. Donovan, and L. Sattler, "Digital calibration incorporating redundancy of flash ADCs," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 50, no. 5, pp. 205–213, 2003. (Cited on pages 25, 30, and 45.)

[8] D. Daly and A. Chandrakasan, "A 6-bit, 0.2 v to 0.9 v highly digital flash ADC with comparator redundancy," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 11, pp. 3030–3038, 2009. (Cited on pages 25, 30, 31, 45, and 51.)

[9] S. Weaver, B. Hershberg, P. Kurahashi, D. Knierim, and U. Moon, "Stochastic flash Analog-to-Digital conversion," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 11, pp. 2825–2833, 2010. (Cited on pages 25, 30, 41, 42, 45, and 47.)

[10] G. Keskin, J. Proesel, J. Plouchart, and L. Pileggi, "Exploiting combinatorial redundancy for offset calibration in flash ADCs," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 8, pp. 1904–1918, 2011. (Cited on pages 25 and 30.)

[11] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Boston: Kluwer Academic Publishers, 1992. (Cited on pages 25 and 32.)

[12] "IEEE standard for terminology and test methods for Analog-To-Digital converters," *IEEE Std 1241-2000*, pp. 1–98, 2001. (Cited on pages 27 and 28.)

[13] M. J. Demler, *High-Speed Analog-to-Digital Conversion*, 1st ed. Academic Press, Jun. 1991. (Cited on page 29.)

[14] D. Johns and K. Martin, *Analog integrated circuit design*. New York, NY: Wiley, 1996. (Cited on page 29.)

[15] F. Kaess, R. Kanan, B. Hochet, and M. Declercq, "New encoding scheme for high-speed flash ADC's," in *Proceedings of 1997 IEEE International Symposium on Circuits and Systems, 1997. ISCAS '97*, vol. 1, 1997, pp. 5–8 vol.1. (Cited on page 29.)

[16] Texas Instruments Incorporated, "Datasheet for TLC5540: 8-bit high-speed analog-to-digital converter," Online at http://www.ti.com/product/tlc5540, Apr. 2004, accessed: 2014-04-21. [Online]. Available: http://www.ti.com/lit/gpn/tlc5540 (Cited on page 29.)

[17] I. Analog Devices, "Datasheet for AD9057: 8-bit, 40/60/80 MSPS A/D converter," Online at http://www.analog.com/en/analog-to-digital-converters/high-speed-ad-converters/ad9057/products/product.html, May 2003, accessed: 2014-04-21. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD9057.pdf (Cited on page 29.)

[18] P. Kinget, "Device mismatch and tradeoffs in the design of analog circuits," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 6, pp. 1212–1224, 2005. (Cited on pages 29 and 49.)

[19] P. Nuzzo, F. D. Bernardinis, P. Terreni, and G. V. der Plas, "Noise analysis of regenerative comparators for reconfigurable ADC architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 6, pp. 1441–1454, 2008. (Cited on pages 29 and 49.)

[20] V. Goyal, "Scalar quantization with random thresholds," *IEEE Signal Processing Letters*, vol. 18, no. 9, pp. 525–528, 2011. (Cited on pages 33 and 142.)

[21] W. R. Bennett, "Spectra of quantized signals," *Bell System Technical Journal*, vol. 27, no. 3, pp. 446–472, 1948. (Cited on pages 33 and 35.)

[22] P. F. Panter and W. Dite, "Quantization distortion in Pulse-Count modulation with nonuniform spacing of levels," *Proceedings of the IRE*, vol. 39, no. 1, pp. 44–48, 1951. (Cited on pages 33, 40, and 149.)

[23] R. Gray and A. Jr. Gray, "Asymptotically optimal quantizers (Corresp.)," *IEEE Trans. Inf. Theory*, vol. 23, no. 1, pp. 143–144, 1977. (Cited on page 33.)

[24] J. G. Smith, "The information capacity of amplitude- and variance-constrained sclar gaussian channels," *Information and Control*, vol. 18, no. 3, pp. 203–219, 1971. (Cited on page 42.)

[25] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. Vol. 34, pp. 43–99, 1956. (Cited on page 53.)

[26] R. Dobrushin and S. Ortyukov, "Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements," *Problemy Peredachi Informatsii*, vol. 13, no. 3, pp. 56–76, 1977. (Cited on page 53.)

[27] ——, "Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements," *Problemy Peredachi Informatsii*, vol. 13, no. 1, pp. 82–89, 1977. (Cited on page 53.)

[28] N. Pippenger, G. D. Stamoulis, and J. N. Tsitsiklis, "On a lower bound for the redundancy of reliable networks with noisy gates," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 639–643, 1991. (Cited on page 53.)

[29] I. Koren and A. Singh, "Fault tolerance in VLSI circuits," *Computer*, vol. 23, no. 7, pp. 73–83, 1990. (Cited on page 53.)

[30] C. Huang, C. Wu, J. Li, and C. Wu, "Built-in redundancy analysis for memory yield improvement," *IEEE Transactions on Reliability*, vol. 52, no. 4, pp. 386–399, 2003. (Cited on page 53.)

[31] N. Verma and A. Chandrakasan, "A 256 kb 65 nm 8T subthreshold SRAM employing Sense-Amplifier redundancy," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 141–149, 2008. (Cited on page 53.)

[32] T. Huang, "High-yield performance-efficient redundancy analysis for 2D memory," *SCIENCE CHINA Information Sciences*, vol. 54, no. 8, pp. 1663–1676, 2011. (Cited on page 53.)

[33] M. C. T. Chao, C. Y. Chin, C. W. Lin *et al.*, "Mathematical yield estimation for two-dimensional-redundancy memory arrays," in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, 2010, pp. 235–240. (Cited on page 53.)

[34] S. Lu, Z. Wang, Y. Tsai, and J. Chen, "Efficient Built-In Self-Repair techniques for multiple repairable embedded RAMs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 4, pp. 620–629, 2012. (Cited on page 53.)

[35] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, Mar. 2004. (Cited on page 63.)

[36] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008. (Cited on pages 69, 70, 78, and 94.)

[37] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA: USENIX, 2010, pp. 1–15. (Cited on page 70.)

[38] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43. (Cited on page 70.)

[39] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008. (Cited on page 70.)

[40] Wikipedia, "Embarrassingly parallel — Wikipedia, the free encyclopedia," Feb. 2013. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Embarrassingly_parallel&oldid=529101726 (Cited on page 70.)

[41] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011. (Cited on page 70.)

[42] W. Neiswanger, C. Wang, and E. Xing, "Asymptotically exact, embarrassingly parallel MCMC," *arXiv:1311.4780 [cs, stat]*, Nov. 2013. [Online]. Available: http://arxiv.org/abs/1311.4780 (Cited on page 70.)

[43] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013. (Cited on pages 70 and 84.)

[44] Amazon mechanical turk. http://www.mturk.com. Accessed: 2014-03-01. [Online]. Available: http://www.mturk.com (Cited on page 70.)

[45] oDesk.com. http://www.odesk.com/. Accessed: 2014-03-01. [Online]. Available: http://www.odesk.com/ (Cited on page 70.)

[46] G. D. Ghare and S. T. Leutenegger, "Improving speedup and response times by replicating parallel programs on a SNOW," in *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, Jan. 2005, pp. 264–287. (Cited on page 70.)

[47] W. Cirne, F. Brasileiro, D. Paranhos, Luís 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. (Cited on page 70.)

[48] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. Berkeley, CA: USENIX, 2008, pp. 29–42. (Cited on pages 70 and 71.)

[49] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA: USENIX, 2010, pp. 1–16. (Cited on pages 70 and 71.)

[50] J. Dean, "Achieving rapid response times in large online services," Online at http://research.google.com/people/jeff/latency.html, Mar. 2012. [Online]. Available: http://research.google.com/people/jeff/latency.html (Cited on pages 70 and 71.)

[51] 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*, ser. NSDI'13. Berkeley, CA: USENIX, 2013, pp. 185–198. (Cited on pages 70 and 71.)

[52] G. Weiss and M. Pinedo, "Scheduling tasks with exponential service times on non-identical processors to minimize various cost functions," *Journal of Applied Probability*, pp. 187–202, 1980. (Cited on page 71.)

[53] K. D. Glazebrook and J. C. Gittins, "On single-machine scheduling with precedence relations and linear or discounted costs," *Operations Research*, vol. 29, no. 1, pp. 161–173, 1981. (Cited on page 71.)

[54] G. Weiss, "Multiserver stochastic scheduling," *Deterministic and Stochastic Scheduling, ed. M. A. Dempster and JK Lenstra*, pp. 157–179, 1982. (Cited on page 71.)

[55] M. Pinedo and G. Weiss, "Scheduling jobs with exponentially distributed processing times and intree precedence constraints on two parallel machines," *Operations Research*, vol. 33, no. 6, pp. 1381–1388, 1985. (Cited on page 71.)

[56] T. Kämpke, "Optimal scheduling of jobs with exponential service times on identical parallel processors," *Operations Research*, vol. 37, no. 1, pp. 126–133, 1989. (Cited on page 71.)

[57] C. S. Chang, R. Nelson, and M. Pinedo, "Scheduling two classes of exponential jobs on parallel processors: Structural results and worst-case analysis," *Advances in applied probability*, pp. 925–944, 1991. (Cited on page 71.)

[58] S. R. Lawrence and E. C. Sewell, "Heuristic, optimal, static, and dynamic schedules when processing times are uncertain," *Journal of Operations Management*, vol. 15, no. 1, pp. 71–82, 1997. (Cited on page 71.)

[59] M. Uetz, *Algorithms for deterministic and stochastic scheduling*. Cuvillier, 2001. (Cited on page 71.)

[60] N. Megow, M. Uetz, and T. Vredeveld, "Models and algorithms for stochastic online scheduling," *Mathematics of Operations Research*, vol. 31, no. 3, pp. 513–525, 2006. (Cited on page 71.)

[61] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, Jan. 2012. (Cited on page 71.)

[62] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Proc. Annu. Allerton Conf. on Commu. Control. & Comput.*, Oct. 2012, pp. 326–333. (Cited on page 71.)

[63] ——, "On the Delay-Storage trade-off in content download from coded distributed storage systems," *arXiv:1305.3945 [cs, math]*, May 2013. [Online]. Available: http://arxiv.org/abs/1305.3945 (Cited on page 71.)

[64] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," *arXiv:1306.3707 [cs]*, Jun. 2013. [Online]. Available: http://arxiv.org/abs/1306.3707 (Cited on page 71.)

[65] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *arXiv:1311.2851 [cs]*, Nov. 2013. [Online]. Available: http://arxiv.org/abs/1311.2851 (Cited on page 71.)

[66] Google cluster data trace document (version 2). http://code.google.com/p/googleclusterdata/wiki/TraceVersion2. Accessed: 2014-03-01. [Online]. Available: http://code.google.com/p/googleclusterdata/wiki/TraceVersion2 (Cited on pages 72 and 75.)

[67] R. Reiss, *Approximate Distributions of Order Statistics: With Applications to Nonparametric Statistics*, 1st ed. Springer, Mar. 1989. (Cited on page 76.)

[68] H. A. David and H. N. Nagaraja, *Order statistics*. Hoboken, N.J.: John Wiley, 2003. (Cited on pages 76, 80, 83, and 145.)

[69] M. Asadi, N. Ebrahimi, G. G. Hamedani, and E. S. Soofi, "Information measures for pareto distributions and order statistics," in *Advances in Distribution Theory, Order Statistics, and Inference*, ser. Statistics for Industry and Technology, N. Balakrishnan, J. M. Sarabia, and E. Castillo, Eds. Birkhuser Boston, Jan. 2006, pp. 207–223. (Cited on page 76.)

[70] N. Balakrishnan and A. C. Cohen, *Order Statistics & Inference: Estimation Methods*. Boston: Academic Press, Jan. 1991. (Cited on page 76.)

[71] A. H. L. de, Ferreira, *Extreme value theory an introduction*. New York: Springer, 2006. (Cited on pages 80 and 81.)

[72] C. Reiss, A. Tumanov, G. R. 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.

[Online]. Available: http://www.pdl.cs.cmu.edu/PDL-FTP/CloudComputing/ISTC-CC-TR-12-101.pdf (Cited on pages 84 and 85.)

[73] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95*, 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-95.pdf (Cited on page 94.)

[74] L. L. Thurstone, "A law of comparative judgment," *Psychological Review*, vol. 34, no. 4, pp. 273–286, 1927. (Cited on page 109.)

[75] D. E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Addison-Wesley Professional, 1998. (Cited on pages 109, 110, 116, 124, 171, and 175.)

[76] J. Cardinal, S. Fiorini, G. Joret, Raphaël M. Jungers, and J. I. Munro, "An efficient algorithm for partial order production," *SIAM Journal on Computing*, vol. 39, no. 7, pp. 2927–2940, 2010. (Cited on page 110.)

[77] A. Schönhage, "The production of partial orders," *Astérisque*, vol. 38, no. 39, pp. 229–246, 1976. (Cited on page 110.)

[78] M. Aigner, "Producing posets," *Discrete Mathematics*, vol. 35, no. 1-3, pp. 1–15, 1981. (Cited on page 110.)

[79] J. Kahn and J. Kim, "Entropy and sorting," *Journal of Computer and System Sciences*, vol. 51, no. 3, pp. 390–399, 1995. (Cited on page 110.)

[80] J. Cardinal, S. Fiorini, G. Joret, Raphaël M. Jungers, and J. I. Munro, "Sorting under partial information (without the ellipsoid algorithm)," in *Proceedings of the 42nd ACM symposium on Theory of computing*, ser. Proc. ACM Symp. Theory Comp. (STOC). New York, NY, USA: ACM, 2010, pp. 359–368. (Cited on page 110.)

[81] J. M. Chambers, "Algorithm 410: Partial sorting," *Commun. ACM*, vol. 14, no. 5, pp. 357–358, 1971. (Cited on page 110.)

[82] M. Fredman, "How good is the information theory bound in sorting?" *Theoretical Computer Science*, vol. 1, no. 4, pp. 355–361, 1976. (Cited on pages 110, 111, 114, and 134.)

[83] K. Kaligosi, K. Mehlhorn, J. I. Munro, and P. Sanders, "Towards optimal multiple selection," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, Luís Caires, G. F. Italiano, Luís Monteiro, C. Palamidessi, and M. Yung, Eds. Springer Berlin Heidelberg, Jan. 2005, no. 3580, pp. 103–114. (Cited on pages 110, 114, and 134.)

[84] J. Cardinal, S. Fiorini, G. Joret, Raphaël M. Jungers, and J. I. Munro, "An efficient algorithm for partial order production," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. Proc. ACM Symp. Theory Comp. (STOC). New York, NY, USA: ACM, 2009, pp. 93–100. (Cited on page 110.)

[85] C. Martnez and H. Prodinger, "Moves and displacements of particular elements in quicksort," *Theoretical Computer Science*, vol. 410, no. 21-23, pp. 2279–2284, 2009. (Cited on page 110.)

[86] A. Panholzer, "Analysis of multiple quickselect variants," *Theoretical Computer Science*, vol. 302, no. 1-3, pp. 45–91, 2003. (Cited on page 110.)

[87] B. Chazelle, "The soft heap: an approximate priority queue with optimal error rate," *J. ACM*, vol. 47, no. 6, pp. 1012–1027, 2000. (Cited on page 110.)

[88] H. Kaplan and U. Zwick, "A simpler implementation and analysis of chazelle's soft heaps," in *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '09. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009, pp. 477–485. (Cited on page 110.)

[89] U. Feige, P. Raghavan, D. Peleg, and E. Upfal, "Computing with noisy information," *SIAM J. Comput.*, vol. 23, no. 5, pp. 1001–1018, 1994. (Cited on pages 110 and 134.)

[90] M. Braverman and E. Mossel, "Sorting from noisy information," *arXiv:0910.1191*, Oct. 2009. [Online]. Available: http://arxiv.org/abs/0910.1191 (Cited on pages 111 and 134.)

[91] R. Klein, R. Penninger, C. Sohler, and D. P. Woodruff, "Tolerant algorithms," in *Algorithms - ESA 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6942, pp. 736–747. (Cited on pages 111 and 134.)

[92] C. L. Mallows, "Non-null ranking models. i," *Biometrika*, vol. 44, no. 1-2, pp. 114–130, 1957. (Cited on pages 111, 114, and 130.)

[93] T. Lu and C. Boutilier, "Learning mallows models with pairwise preferences," *ICML-11, Bellevue, WA*, 2011. (Cited on pages 111, 114, 131, 132, and 135.)

[94] N. Linial, "The Information-Theoretic bound is good for merging," *SIAM Journal on Computing*, vol. 13, p. 795, 1984. (Cited on page 111.)

[95] J. Kahn and M. Saks, "Every poset has a good comparison," in *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, ser. Proc. ACM Symp. Theory Comp. (STOC). New York, NY, USA: ACM, 1984, pp. 299–301. (Cited on page 111.)

[96] János Komls, "A strange pigeon-hole principle," *Order*, vol. 7, no. 2, pp. 107–113, 1990. (Cited on page 111.)

[97] S. Lin, "Rank aggregation methods," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 5, pp. 555–570, 2010. (Cited on page 111.)

[98] N. Ailon, M. Charikar, and A. Newman, "Aggregating inconsistent information: Ranking and clustering," *J. ACM*, vol. 55, no. 5, pp. 23:1–23:27, 2008. (Cited on page 111.)

[99] A. Ammar and D. Shah, "Efficient rank aggregation using partial data," in *Proc. SIGMETRICS*, 2012. (Cited on page 111.)

[100] K. G. Jamieson and R. D. Nowak, "Active ranking using pairwise comparisons," *Arxiv preprint arXiv:1109.3701*, 2011. (Cited on page 112.)

[101] T. Liu, "Learning to rank for information retrieval," *Found. Trends Inf. Retr.*, vol. 3, no. 3, pp. 225–331, 2009. (Cited on page 112.)

[102] S. Agarwal, "Learning to rank on graphs," *Machine Learning*, vol. 81, no. 3, pp. 333–357, 2010. (Cited on page 112.)

[103] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," in *JMLR: Workshop and Conference Proceedings*, vol. 14, 2011, pp. 1–24. (Cited on page 112.)

[104] N. Ailon, "An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity," *arXiv:1011.0108*, Oct. 2010. [Online]. Available: http://arxiv.org/abs/1011.0108 (Cited on page 112.)

[105] D. P. Helmbold and M. K. Warmuth, "Learning permutations with exponential weights," *J. Mach. Learn. Res.*, vol. 10, pp. 1705–1736, Dec. 2009. (Cited on page 112.)

[106] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952. (Cited on page 114.)

[107] R. D. Luce, *Individual Choice Behavior: A Theoretical Analysis*. Wiley, 1959. (Cited on page 114.)

[108] R. L. Plackett, "The analysis of permutations," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 24, no. 2, pp. 193–202, 1975. (Cited on page 114.)

[109] W. Cheng and E. Hüllermeier, "A new Instance-Based label ranking approach using the mallows model," in *Advances in Neural Networks - ISNN 2009*, ser. Lecture Notes in Computer Science, W. Yu, H. He, and N. Zhang, Eds. Springer Berlin Heidelberg, Jan. 2009, no. 5551, pp. 707–716. (Cited on page 114.)

[110] A. Klementiev, D. Roth, and K. Small, "Unsupervised rank aggregation with distance-based models," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 472–479. (Cited on page 114.)

[111] M. Deza and T. Huang, "Metrics on permutations, a survey," *Journal of Combinatorics, Information and System Sciences*, vol. 23, pp. 173–185, 1998. (Cited on pages 114, 131, and 175.)

[112] P. Diaconis and R. L. Graham, "Spearman's footrule as a measure of disarray," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 2, pp. 262–268, 1977. (Cited on pages 119, 174, and 175.)

[113] D. Wang, A. Mazumdar, and G. Wornell, "A rate-distortion theory for permutation spaces," in *Proc. IEEE Int. Symp. Inform. Th. (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 2562–2566. (Cited on page 123.)

[114] F. Farnoud, M. Schwartz, and J. Bruck, "Rate-Distortion for ranking with incomplete information," *arXiv:1401.3093 [cs, math]*, Jan. 2014. [Online]. Available: http://arxiv.org/abs/1401.3093 (Cited on page 123.)

[115] P. Diaconis and A. Ram, *Analysis of systematic scan Metropolis algorithms using Iwahori-Hecke algebra techniques*. Department of Statistics, Stanford University, 2000. (Cited on pages 131 and 135.)

[116] J. Doignon, A. Peke, and M. Regenwetter, "The repeated insertion model for rankings: Missing link between two subset choice models," *Psychometrika*, vol. 69, no. 1, pp. 33–54, 2004. (Cited on pages 131, 132, and 134.)

[117] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, S. Narayan, D. Beece, J. Piaget, N. Venkateswaran, and J. Hemmett, "First-Order incremental Block-Based statistical timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2170–2180, 2006. (Cited on page 137.)

[118] S. Nawab, A. Oppenheim, A. Chandrakasan, J. Winograd, and J. Ludwig, "Approximate signal processing," *The Journal of VLSI Signal Processing*, vol. 15, no. 1, pp. 177–200, 1997. (Cited on page 137.)

[119] D. A. Darling, "On a class of problems related to the random division of an interval," *The Annals of Mathematical Statistics*, vol. 24, no. 2, pp. 239–253, 1953. (Cited on pages 145 and 146.)

[120] B. Ballobás, *Random Graphs*, 2nd ed. Cambridge University Press, Oct. 2001. (Cited on page 152.)

[121] R. Motwani and P. Raghavan, "Randomized algorithms," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 33–37, 1996. (Cited on page 154.)

[122] H. J. Malik, "Exact moments of order statistics from the pareto distribution," *Scandinavian Actuarial Journal*, vol. 1966, no. 3-4, pp. 144–157, 1966. (Cited on page 158.)

[123] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3158–3165, 2010. (Cited on pages 171 and 173.)

[124] R. Shreevatsa, "The On-Line Encyclopedia of Integer Sequences," 2013. [Online]. Available: http://oeis.org/A161169 (Cited on page 171.)

[125] R. P. Stanley, *Enumerative Combinatorics, Vol. 1.* Cambridge University Press, Apr. 1997. (Cited on page 172.)

[126] G. Louchard, H. Prodinger, and P. O. Wits, "The number of inversions in permutations: a saddle point approach," *Journal of Integer Sequences*, vol. 6, no. 2, pp. 1–19, 2003. (Cited on page 173.)

[127] A. Mazumdar, A. Barg, and G. Zemor, "Constructions of rank modulation codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 2, pp. 1018–1029, 2013. (Cited on pages 174 and 175.)

[128] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein, *Covering Codes.* Elsevier, Apr. 1997. (Cited on page 180.)