Performance Modeling and Design of Computer Systems

Computer systems design is full of conundrums:

- Given a choice between a single machine with speed s, or n machines each with speed s/n, which should we choose?
- If both the arrival rate and service rate double, will the mean response time stay the same?
- Should systems really aim to balance load, or is this a convenient myth?
- If a scheduling policy favors one set of jobs, does it necessarily hurt some other jobs, or are these "conservation laws" being misinterpreted?
- Do greedy, shortest-delay, routing strategies make sense in a server farm, or is what is good for the individual disastrous for the system as a whole?
- How do high job size variability and heavy-tailed workloads affect the choice of a scheduling policy?
- How should one trade off energy and delay in designing a computer system?
- If 12 servers are needed to meet delay guarantees when the arrival rate is 9 jobs/sec, will we need 12,000 servers when the arrival rate is 9,000 jobs/sec?

Tackling the questions that systems designers care about, this book brings queueing theory decisively back to computer science. The book is written with computer scientists and engineers in mind and is full of examples from computer systems, as well as manufacturing and operations research. Fun and readable, the book is highly approachable, even for undergraduates, while still being thoroughly rigorous and also covering a much wider span of topics than many queueing books.

Readers benefit from a lively mix of motivation and intuition, with illustrations, examples, and more than 300 exercises – all while acquiring the skills needed to model, analyze, and design large-scale systems with good performance and low cost. The exercises are an important feature, teaching research-level counterintuitive lessons in the design of computer systems. The goal is to train readers not only to customize existing analyses but also to invent their own.

Mor Harchol-Balter is an Associate Professor in the Computer Science Department at Carnegie Mellon University. She is a leader in the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, having served as technical program committee chair in 2007 and conference chair in 2013.

Performance Modeling and Design of Computer Systems

Queueing Theory in Action

Mor Harchol-Balter
Carnegie Mellon University, Pennsylvania

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town,
Singapore, São Paulo, Delhi, Mexico City

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org
Information on this title: www.cambridge.org/9781107027503

First published 2013

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data

Harchol-Balter, Mor, 1966–
Performance modeling and design of computer systems : queueing theory in
action / Mor Harchol-Balter.
pages cm
Includes bibliographical references and index.
ISBN 978-1-107-02750-3
1. Transaction systems (Computer systems) – Mathematical models. 2. Computer
systems – Design and construction – Mathematics. 3. Queueing theory.
4. Queueing networks (Data transmission) I. Title.
QA76.545.H37 2013
519.8′2–dc23 2012019844

ISBN 978-1-107-02750-3 Hardback

To my loving husband Andrew, my awesome son Danny,
and my parents, Irit and Micha

I have always been interested in finding better designs for computer systems, designs that improve performance without the purchase of additional resources. When I look back at the problems that I have solved and I look ahead to the problems I hope to solve, I realize that the problem formulations keep getting simpler and simpler, and my footing less secure. Every wisdom that I once believed, I have now come to question: If a scheduling policy helps one set of jobs, does it necessarily hurt some other jobs, or are these "conservation laws" being misinterpreted? Do greedy routing strategies make sense in server farms, or is what is good for the individual actually disastrous for the system as a whole? When comparing a single fast machine with n *slow machines, each of* $1/n$*th the speed, the single fast machine is typically much more expensive – but does that mean that it is necessarily better? Should distributed systems really aim to balance load, or is this a convenient myth? Cycle stealing, where machines can help each other when they are idle, sounds like a great idea, but can we quantify the actual benefit? How much is the performance of scheduling policies affected by variability in the arrival rate and service rate and by fluctuations in the load, and what can we do to combat variability? Inherent in these questions is the impact of real user behaviors and real-world workloads with heavy-tailed, highly variable service demands, as well as correlated arrival processes. Also intertwined in my work are the tensions between theoretical analysis and the realities of implementation, each motivating the other. In my search to discover new research techniques that allow me to answer these and other questions, I find that I am converging toward the fundamental core that defines all these problems, and that makes the counterintuitive more believable.*

Cambridge University Press
978-1-107-02750-3 - Performance Modeling and Design of Computer Systems: Queueing Theory in Action
Mor Harchol-Balter
Frontmatter
More information

Contents

Cambridge University Press
978-1-107-02750-3 - Performance Modeling and Design of Computer Systems: Queueing Theory in Action
Mor Harchol-Balter
Frontmatter
More information

Preface

The ad hoc World of Computer System Design

The design of computer systems is often viewed very much as an art rather than a science. Decisions about which scheduling policy to use, how many servers to run, what speed to operate each server at, and the like are often based on *intuitions* rather than mathematically derived formulas. Specific policies built into kernels are often riddled with secret "voodoo constants,"[1] which have no explanation but seem to "work well" under some benchmarked workloads. Computer systems students are often told to *first* build the system and *then* make changes to the policies to improve system performance, rather than first creating a formal model and design of the system on paper to ensure the system meets performance goals.

Even when trying to evaluate the performance of an *existing* computer system, students are encouraged to *simulate* the system and spend many days running their simulation under different workloads waiting to see what happens. Given that the search space of possible workloads and input parameters is often huge, vast numbers of simulations are needed to properly cover the space. Despite this fact, mathematical models of the system are rarely created, and we rarely characterize workloads stochastically. There is no formal analysis of the parameter space under which the computer system is likely to perform well versus that under which it is likely to perform poorly. It is no wonder that computer systems students are left feeling that the whole process of system evaluation and design is very ad hoc. As an example, consider the trial-and-error approach to updating resource scheduling in the many versions of the Linux kernel.

Analytical Modeling for Computer Systems

But it does not have to be this way! These same systems designers could mathematically model the system, stochastically characterize the workloads and performance goals, and then analytically derive the performance of the system as a function of workload and input parameters. The fields of *analytical modeling* and *stochastic processes* have existed for close to a century, and they can be used to save systems designers huge numbers of hours in trial and error while improving performance. Analytical modeling can also be used in conjunction with simulation to help guide the simulation, reducing the number of cases that need to be explored.

[1] The term "voodoo constants" was coined by Prof. John Ousterhout during his lectures at the University of California, Berkeley.

Unfortunately, of the hundreds of books written on stochastic processes, almost none deal with computer systems. The examples in those books and the material covered are oriented toward operations research areas such as manufacturing systems, or *human* operators answering calls in a call center, or some assembly-line system with different priority jobs.

In many ways the analysis used in designing manufacturing systems is not all that different from computer systems. There are many parallels between a human operator and a computer server: There are faster human operators and slower ones (just as computer servers); the human servers sometimes get sick (just as computer servers sometimes break down); when not needed, human operators can be sent home to save money (just as computer servers can be turned off to save power); there is a startup overhead to bringing back a human operator (just as there is a warmup cost to turning on a computer server); and the list goes on.

However, there are also many differences between manufacturing systems and computer systems. To start, computer systems workloads have been shown to have extremely high variability in job sizes (service requirements), with squared coefficients of variation upward of 100. This is very different from the low-variability service times characteristic of job sizes in manufacturing workloads. This difference in variability can result in performance differences of orders of magnitude. Second, computer workloads are typically preemptible, and time-sharing (Processor-Sharing) of the CPU is extremely common. By contrast, most manufacturing workloads are non-preemptive (first-come-first-serve service order is the most common). Thus most books on stochastic processes and queueing omit chapters on Processor-Sharing or more advanced preemptive policies like Shortest-Remaining-Processing-Time, which are very much at the heart of computer systems. Processor-Sharing is particularly relevant when analyzing server farms, which, in the case of computer systems, are typically composed of Processor-Sharing servers, not First-Come-First-Served ones. It is also relevant in any computing application involving bandwidth being shared between users, which typically happens in a processor-sharing style, not first-come-first-serve order. Performance metrics may also be different for computer systems as compared with manufacturing systems (e.g., power usage, an important metric for computer systems, is not mentioned in stochastic processes books). Closed-loop architectures, in which new jobs are not created until existing jobs complete, and where the performance goal is to maximize throughput, are very common in computer systems, but are often left out of queueing books. Finally, the particular types of interactions that occur in disks, networking protocols, databases, memory controllers, and other computer systems are very different from what has been analyzed in traditional queueing books.

The Goal of This Book

Many times I have walked into a fellow computer scientist's office and was pleased to find a queueing book on his shelf. Unfortunately, when questioned, my colleague was quick to answer that he never uses the book because "The world doesn't look like an M/M/1 queue, and I can't understand anything past that chapter." The problem is that the queueing theory books are not "friendly" to computer scientists. The applications are not computer-oriented, and the assumptions used are often unrealistic for computer systems. Furthermore, these books are abstruse and often impenetrable by anyone who has not studied graduate-level mathematics. In some sense this is hard to avoid: If one wants to do more than provide readers with formulas to "plug into," then one has to *teach* them to derive their own formulas, and this requires learning a good deal of math. Fortunately, as one of my favorite authors, Sheldon Ross, has shown, it *is* possible to teach a lot of stochastic analysis in a fun and simple way that does not require first taking classes in measure theory and real analysis.

My motive in writing this book is to improve the design of computer systems by introducing computer scientists to the powerful world of queueing-theoretic modeling and analysis. Personally, I have found queueing-theoretic analysis to be extremely valuable in much of my research including: designing routing protocols for networks, designing better scheduling algorithms for web servers and database management systems, disk scheduling, memory-bank allocation, supercomputing resource scheduling, and power management and capacity provisioning in data centers. Content-wise, I have two goals for the book. First, I want to provide enough applications from computer systems to make the book relevant and interesting to computer scientists. Toward this end, almost half the chapters of the book are "application" chapters. Second, I want to make the book mathematically rich enough to give readers the ability to actually *develop new queueing analysis*, not just apply existing analysis. As computer systems and their workloads continue to evolve and become more complex, it is unrealistic to assume that they can be modeled with known queueing frameworks and analyses. As a designer of computer systems myself, I am constantly finding that I have to invent new queueing concepts to model aspects of computer systems.

How This Book Came to Be

In 1998, as a postdoc at MIT, I developed and taught a new computer science class, which I called "Performance Analysis and Design of Computer Systems." The class had the following description:

> In designing computer systems one is usually constrained by certain performance goals (e.g., low response time or high throughput or low energy). On the other hand, one often has many choices: One fast disk, or two slow ones? What speed CPU will suffice? Should we invest our money in more buffer space or a faster processor? How should jobs be scheduled by the processor? Does it pay to migrate active jobs? Which routing policy will work best? Should one balance load among servers? How can we best combat high-variability workloads? Often answers to these questions are counterintuitive. Ideally, one would like to have answers to these questions before investing the time and money to build a system. This class will introduce students to analytic stochastic modeling, which allows system designers to answer questions such as those above.

Since then, I have further developed the class via 10 more iterations taught within the School of Computer Science at Carnegie Mellon, where I taught versions of the class to both PhD students and advanced undergraduates in the areas of computer science, engineering, mathematics, and operations research. In 2002, the Operations Management department within the Tepper School of Business at Carnegie Mellon made the class a qualifier requirement for all operations management students.

As other faculty, including my own former PhD students, adopted my lecture notes in teaching their own classes, I was frequently asked to turn the notes into a book. This is "version 1" of that book.

Outline of the Book

This book is written in a question/answer style, which mimics the Socratic style that I use in teaching. I believe that a class "lecture" should ideally be a long sequence of bite-sized questions, which students can easily provide answers to and which lead students to the right intuitions. In reading this book, it is extremely important to try to answer each question *without* looking at the answer that follows the question. The questions are written to remind the reader to "think" rather than just "read," and to remind the teacher to ask questions rather than just state facts.

There are exercises at the end of each chapter. The exercises are an integral part of the book and should not be skipped. Many exercises are used to illustrate the application of the theory to problems in computer systems design, typically with the purpose of illuminating a key insight. All exercises are related to the material covered in the chapter, with early exercises being straightforward applications of the material and later exercises exploring extensions of the material involving greater difficulty.

The book is divided into seven parts, which mostly build on each other.

Part I introduces queueing theory and provides motivating examples from computer systems design that can be answered using basic queueing analysis. Basic queueing terminology is introduced including closed and open queueing models and performance metrics.

Part II is a probability refresher. To make this book self-contained, we have included in these chapters all the probability that will be needed throughout the rest of the book. This includes a summary of common discrete and continuous random variables, their moments, and conditional expectations and probabilities. Also included is some material on generating random variables for simulation. Finally we end with a discussion of sample paths, convergence of sequences of random variables, and time averages versus ensemble averages.

Part III is about operational laws, or "back of the envelope" analysis. These are very simple laws that hold for all well-behaved queueing systems. In particular, they do not require that any assumptions be made about the arrival process or workload (like Poisson arrivals or Exponential service times). These laws allow us to quickly reason at a high level (averages only) about system behavior and make design decisions regarding what modifications will have the biggest performance impact. Applications to high-level computer system design are provided throughout.

Part IV is about Markov chains and their application toward stochastic analysis of computer systems. Markov chains allow a much more detailed analysis of systems by representing the full space of possible states that the system can be in. Whereas the operational laws in Part III often allow us to answer questions about the overall mean number of jobs in a system, Markov chains allow us to derive the probability of exactly i jobs being queued at server j of a multi-server system. Part IV includes both discrete-time and continuous-time Markov chains. Applications include Google's PageRank algorithm, the Aloha (Ethernet) networking protocol, and an analysis of dropping probabilities in finite-buffer routers.

Part V develops the Markov chain theory introduced in Part IV to allow the analysis of more complex networks, including server farms. We analyze networks of queues with complex routing rules, where jobs can be associated with a "class" that determines their route through the network (these are known as BCMP networks). Part V also derives theorems on capacity provisioning of server farms, such as the "square-root staffing rule," which determines the minimum number of servers needed to provide certain delay guarantees.

The fact that Parts IV and V are based on Markov chains necessitates that certain "Markovian" (memoryless) assumptions are made in the analysis. In particular, it is assumed that the service requirements (sizes) of jobs follow an Exponential distribution and that the times between job arrivals are also Exponentially distributed. Many applications are reasonably well modeled via these Exponential assumptions, allowing us to use Markov analysis to get good insights into system performance. However, in some cases, it is important to capture the high-variability job size distributions or correlations present in the empirical workloads.

Part VI introduces techniques that allow us to replace these Exponential distributions with high-variability distributions. Phase-type distributions are introduced, which allow us to model virtually any general distribution by a *mixture of Exponentials*, leveraging our understanding of Exponential distributions and Markov chains from Parts IV and V. Matrix-analytic techniques are then developed to analyze systems with phase-type workloads in both the arrival process and service process. The M/G/1 queue is introduced, and notions such as the Inspection Paradox are discussed. Real-world workloads are described including heavy-tailed distributions. Transform techniques are also introduced that facilitate working with general distributions. Finally, even the service order at the queues is generalized from simple first-come-first-served service order to time-sharing (Processor-Sharing) service order, which is more common in computer systems. Applications abound: Resource allocation (task assignment) in server farms with high-variability job sizes is studied extensively, both for server farms with non-preemptive workloads and for web server farms with time-sharing servers. Power management policies for single servers and for data centers are also studied.

Part VII, the final part of the book, is devoted to scheduling. Smart scheduling is extremely important in computer systems, because it can dramatically improve system performance without requiring the purchase of any new hardware. Scheduling is at the heart of operating systems, bandwidth allocation in networks, disks, databases, memory hierarchies, and the like. Much of the research being done in the computer systems area today involves the design and adoption of new scheduling policies. Scheduling can be counterintuitive, however, and the analysis of even basic scheduling policies is far from simple. Scheduling policies are typically evaluated via simulation. In introducing the reader to analytical techniques for evaluating scheduling policies, our hope is that more such policies might be evaluated via analysis.

We expect readers to mostly work through the chapters in order, with the following exceptions: First, any chapter or section marked with a star (*) can be skipped without disturbing the flow. Second, the chapter on transforms, Chapter 25, is purposely moved to the end, so that most of the book does not depend on knowing transform analysis. However, because learning transform analysis takes some time, we recommend that any teacher who plans to cover transforms introduce the topic a little at a time, starting early in the course. To facilitate this, we have included a large number of exercises at the end of Chapter 25 that do not require material in later chapters and can be assigned earlier in the course to give students practice manipulating transforms.

Finally, we urge readers to please check the following websites for new errors/software:

http://www.cs.cmu.edu/∼harchol/PerformanceModeling/errata.html

http://www.cs.cmu.edu/∼harchol/PerformanceModeling/software.html

Please send any additional errors to harchol@cs.cmu.edu.

Acknowledgments

Writing a book, I quickly realized, is very different from writing a research paper, even a very long one. Book writing actually bears much more similarity to teaching a class. That is why I would like to start by thanking the three people who most influenced my teaching. Manuel Blum, my PhD advisor, taught me the art of creating a lecture out of a series of bite-sized questions. Dick Karp taught me that you can cover an almost infinite amount of material in just one lecture if you spend enough time in advance simplifying that material into its cleanest form. Sheldon Ross inspired me by the depth of his knowledge in stochastic processes (a knowledge so deep that he never once looked at his notes while teaching) and by the sheer clarity and elegance of both his lectures and his many beautifully written books.

I would also like to thank Carnegie Mellon University, and the School of Computer Science at Carnegie Mellon, which has at its core the theme of interdisciplinary research, particularly the mixing of theoretical and applied research. CMU has been the perfect environment for me to develop the analytical techniques in this book, all in the context of solving hard applied problems in computer systems design. CMU has also provided me with a never-ending stream of gifted students, who have inspired many of the exercises and discussions in this book. Much of this book came from the research of my own PhD students, including Sherwin Doroudi, Anshul Gandhi, Varun Gupta, Yoongu Kim, David McWherter, Takayuki Osogami, Bianca Schroeder, Adam Wierman, and Timothy Zhu. In addition, Mark Crovella, Mike Kozuch, and particularly Alan Scheller-Wolf, all longtime collaborators of mine, have inspired much of my thinking via their uncanny intuitions and insights.

A great many people have proofread parts of this book or tested out the book and provided me with useful feedback. These include Sem Borst, Doug Down, Erhun Ozkan, Katsunobu Sasanuma, Alan Scheller-Wolf, Thrasyvoulos Spyropoulos, Jarod Wang, and Zachary Young. I would also like to thank my editors, Diana Gillooly and Lauren Cowles from Cambridge University Press, who were very quick to answer my endless questions, and who greatly improved the presentation of this book. Finally, I am very grateful to Miso Kim, my illustrator, a PhD student at the Carnegie Mellon School of Design, who spent hundreds of hours designing all the fun figures in the book.

On a more personal note, I would like to thank my mother, Irit Harchol, for making my priorities her priorities, allowing me to maximize my achievements. I did not know what this meant until I had a child of my own. Lastly, I would like to thank my husband, Andrew Young. He won me over by reading all my online lecture notes and doing every homework problem – this was his way of asking me for a first date. His ability to understand it all without attending any lectures made me believe that my lecture notes might actually "work" as a book. His willingness to sit by my side every night for many months gave me the motivation to make it happen.