

Analyzing Computer System Performance with Perl::PDQ

Neil J. Gunther

Analyzing Computer System Performance with Perl::PDQ

Second Edition



Springer

Neil J. Gunther
Performance Dynamics Company
4061 East Castro Valley Boulevard
Castro Valley, CA 94552
USA
<http://www.perfdynamics.com/>

ISBN 978-3-642-22582-6 e-ISBN 978-3-642-22583-3
DOI 10.1007/978-3-642-22583-3
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011934511

ACM Codes: C.0, C.2.4, C.4, D.2.5, D.2.8, D.4.8, K.6.2

© Springer-Verlag Berlin Heidelberg 2005, 2011

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: KünkelLopka

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

This book is dedicated to the memory of my father
Walter August Gunther, *MIEAust*, *CPEng*
October 23, 1908 – November 16, 2010

Preface to Second Edition

What's New?

It seems fitting that this new edition appears as we pass the centenary of the original paper by Erlang [1909], where he first introduced the concept of a queue. This edition incorporates a considerable number of new features compiled since the publication of the first edition in 2005.

New Chapters and Partitioning

The amount of additional material means the book now comprises four parts instead of three. In particular, Part I contains new chapters that present a more complete discussion of the underlying concepts used throughout this book.

Improved Perl Formatting

All listings have a highlighted format to aid readability of PDQ codes.

Listing 1. Example of the new PDQ code format

```
#!/usr/bin/perl

use pdq;

pdq::Init("Example");
pdq::CreateNode($NodeName, $pdq::CEN, $pdq::FCFS);
pdq::CreateOpen($WorkName, $ArrivalRate);
pdq::SetDemand($NodeName, $WorkName, $ServiceDemand);
pdq::Solve($pdq::CANON);
pdq::Report();
```

Virtualization

A new Chapter 13: *Virtual Machine Analysis with PDQ*, in Part III, presents queueing models of *fair-share scheduling* that underpins all modern virtual machine implementations from hyperthreading to cloud computing.

PDQ on SourceForge

All PDQ development is now gated through SourceForge sourceforge.net/projects/pdq-qnm-pkg/ under the title *Pretty Damn Quick Queueing Model Package*. PDQ can also be downloaded from the author's distribution page at www.perfdynamics.com/Tools/PDQcode.html.

Why Queues Rule

A new Chapter 1: *Why Queues Rule Performance Analysis*, endeavors to explain why queueing models are so powerful for doing computer performance analysis. See Example 1.2 which presents a PDQ performance and capacity model of servers that are dedicated to filtering email spam.

PDQ Manual

Part IV comprises a set of appendices. Included there is the PDQ Manual which has been broken out from its previous inclusion in the chapter: *Pretty Damn Quick—A Slow Introduction*. Updates are available online at www.perfdynamics.com/Tools/PDQman.html.

CreateMultiNode Function

The latest release of the open source PDQ code now implements multi-server queueing nodes. See Appendix D.3.2 for details.

Brief History of Buffers

The potted history of queueing theory entitled *A Brief History of Buffers*, that was previously isolated as a separate Appendix, has been updated and now appears at the end of the new Chapter 1.

Performance Management Tools

The Appendix on performance management tools in the first edition has now been expanded in a new Chapter 2.

Scalability and Queueing

A new Section 4.11.12 in Chapter 4 shows how the author's *universal scalability law* (developed in the book *Guerrilla Capacity Planning* [Gunther 2007b]) is related to the queueing models presented in this book, viz., the *machine repairman* model [Gunther 2008].

Jackson's Theorem

Chapter 5 contains a new section explaining the importance of Jackson's theorem for circuits of queues. This concept is vital for constructing performance models of modern multi-tier applications, such as those employed at large-scale web sites.

Glossary Removed

The Glossary in the first edition became outdated and has been removed in order to accommodate the new chapter content without unduly increasing the size of the entire book.

Crowd-sourced Corrections

The corrigenda at www.perfdynamics.com/iBook/ppdgerrata.html is a testament to the power of the internet for enabling many eyes to spot typos and errors. Every effort has been made to include all the listed errata in this edition.

Acknowledgments

Phil Feller masterfully applied SWIG (www.swig.org/) to the PDQ function C library in order to programmatically convert it to Perl. Stefan Parvu championed the use of PDQ in the field and provided important feedback for Section 4.11.12. The performance group at VMware Inc., contributed to some very useful discussions that helped to shape Chap. 13.

Once again, I am indebted to the alumni of Performance Dynamics Company classes, and other diligent readers, who contributed errata for the first edition at www.perfdynamics.com/iBook/ppdgerrata.html. In alphabetical order they are: P. Altevogt, D. Anburaj, W. Baeck, T. Becker, E. Borasky L. Braswell, D. Hagler, E. Juan, S. Kannan, M. Marino, P. Puglia, J. Purinton, T. Sych, I. Tegebo, D. Walter, T. Wilson. In particular, P. Cañadilla did a truly outstanding job, as his record tally attests. If it there is such a thing as a *copy-editor gene*, I believe he has it.

Finally, I am grateful to Ralf Gerstner, my editor, for his patience while I searched for fragmented opportunities to update the manuscript during some difficult periods over the past two years.

Melbourne, Australia
December, 2010

N.J.G.

```
perl -le '@q=("120145162154","120104121");
$s="115141171040171157165162040@q040bs040142145";
$q[0]=s/e/ea/;$q[0]=lcfirst($q[0]);@q=reverse(@q);$s.=" @q \bs!";print $s'
```

Preface to First Edition

Motivation

This book arose out of an attempt to meet two key objectives. The first was to communicate the theory and practice of performance analysis to those who need it most, viz. IT professionals, system administrators, software developers, and performance test engineers. Many of the currently available books on computer performance analysis fall into one of three distinct camps:

1. Books that discuss tuning the performance of a particular platform, e.g., Linux, Solaris, Windows. These books explain how you can turn individual software “knobs” with the hope that this will tune your platform.
2. Books that emphasize formal queueing theory under the rubric of performance modeling. These books are written by mathematicians for mathematicians and therefore are burdened with too much Greek for the average IT professional to suffer through.
3. Books that employ queueing theory without the Greek but the performance models are unrealistic because they are essentially academic toys.

Each of these categories has pedagogic merit, but the focus tends to be on detailed particulars that are not easily generalized to a different context. These days, IT professionals are required to be versed in more than one platform or technology. It seemed to me that the best way to approach the performance analysis of such a panoply is to adopt a *system* perspective. The system view also provides an economy of thought. Understanding gained on one system can often be applied to another. Successful performance analysis on one platform often translates successfully to another, with little extra effort. Expressed in today’s vernacular—*learn once, apply often*.

Second, I wanted to present system performance principles in the context of a software tool, *Pretty Damn Quick* (PDQ), that can be applied quickly to address performance issues as they arise in today’s hectic business environment. In order to meet the pressures of ever-shortening time horizons, performance analysis has to be done in *zero time*. Project managers cannot

and will not allow their schedules to be stretched by what they perceive as inflationary performance analysis. A performance analysis tool based on a scripting language helps to meet these severe time constraints by avoiding the need to wrestle with compilers and debuggers.

Why Perl?

Defending the choice of a programming language is always a losing proposition, but in a recent poll on `slashdot.org`, Perl (Practical Extraction and Reporting Language,) was ranked third after Bourne shell and Ruby in terms of ease of use for accomplishing a defined set of tasks with a scripting language. Python, Tcl, and Awk, came in fifth, seventh, and eighth respectively, while Java (interpreted but not a scripting language) came in last. Neither *Mathematica* nor PHP were polled. On a more serious note, John Ousterhout (father of Tcl), has written an essay (`home.pacbell.net/ouster/scripting.html`) on the general virtues of scripting languages for prototyping. Where he says *prototyping*, I would substitute the word *modeling*.

I chose Perl because it fitted the requirement of a rapid prototyping language for computer performance analysis. The original implementation of PDQ was in C (and still is as far as the library functions are concerned). To paraphrase a leading UNIX developer, one of the disadvantages of the C language is that you can spend a lot of time in the debugger when you stab yourself with a misreferenced pointer. Perl has a C-like syntax but is much more forgiving at runtime. Moreover, Perl has arguably become the most ubiquitous of the newer-generation scripting languages, including MacPerl on MacOS (prior to MacOS X). One reason for Perl's ubiquity is that it is designed for extracting text and data from files. Why not for extracting performance data? It therefore seemed like a good choice to offer a Perl version of PDQ as an enhancement to the existing toolset of system administrators. By a happy coincidence, several students, who were also system administrators, began asking me if PDQ could be made available in Perl. So, here it is. Bonne programmation!

How should PDQ be used? In my view, the proper analysis of computer performance data requires a conceptual framework within which the information hidden in those data can be revealed. That is the role of PDQ. It provides a framework of expectations in which to assess data. If you do performance analysis without such a framework (as is all too common), how can you know when you are wrong? When your conclusion does not reconcile with the data, you must stop and find where the inconsistency lies. It is much easier to detect inconsistencies when you have certain expectations. Setting some expectations (even wrong ones) is far better than not setting any.

I sometimes liken the role of PDQ to that of a subway map. A subway map has two key properties. It is an *abstract* representation of the real situation in that the distances between train stations are not in geographical

proportion, and it is *simple* because it is uncluttered by unimportant real-world physical details. The natural urge is to create a PDQ “map” adorned with an abundance of physical detail because that would seem to constitute a more faithful representation of the computer system being analyzed. In spite of this urge, you should strive instead to make your PDQ models as simple and abstract as a subway map. Adding complexity does not guarantee accuracy. Unfortunately, there is no simple recipe for constructing PDQ maps. Einstein reputedly said that things should be as simple as possible, but no simpler. That should certainly be the goal for applying PDQ, but like drawing any good map there are aspects that remain more in the realm of art than science. Those aspects are best demonstrated by example, and that is the purpose of Part II of this book.

Book Structure

Very simply, this book falls into two parts, so that the typical rats-nest diagram of chapter dependencies is rendered unnecessary.

Part I explains the fundamental metrics used in computer performance analysis. Chapter 3 discusses the zeroth metric, time, that is common to all performance analysis. This chapter is recommended reading for those new to computer performance analysis but may be skipped in a first reading by those more familiar with performance analysis concepts. The queueing concepts encoded in PDQ tool are presented in Chaps. 4, 5, and 7, so these chapters may also be read sequentially.

For those familiar with UNIX platforms, a good place to start might be Chap. 6 where the connection between queues (buffers) and the *load average* metric is dissected at the kernel level. Linux provides the particular context because the source code is publicly available to be dissected—on the Web, no less! The generalization to other operating systems should be obvious. Similarly, another starting point for those with a UNIX orientation could be Section 1.7 *A Short History of Buffers* (pun intended) which summarizes the historical interplay between queueing theory and computer performance analysis, commencing with the ancestors of UNIX viz. CTSS and Multics.

Irrespective of the order you choose to read them, none of the chapters in Part I requires a knowledge of formal probability theory or stochastic methods. Thus, we avoid the torrent of Greek that otherwise makes very powerful queueing concepts incomprehensible to those readers who would actually benefit from them most.

Part II covers a wide variety of examples demonstrating how to apply PDQ. These include the performance analysis of multicomputer architectures in Chap. 9, analyzing benchmark results in Chap. 10, client/server scalability in Chap. 11, and Web-based applications in Chap. 12. These chapters can be read in any order. Dependencies on other chapters are cross-referenced in the text.

Chapter 8 (Pretty Damn Quick (PDQ)—A Slow Introduction) contains the PDQ *driver's manual* and, because it is a reference manual, can be read independently of the other chapters. It also contains many examples that were otherwise postponed from Chaps. 4–7.

Appendix D contains the steps for installing Perl PDQ together with a complete list of the Perl programs used in this book. The more elementary of these programs are specially identified for those unfamiliar with writing Perl scripts.

Classroom Usage

This book grew out of class material presented at both academic institutions and corporate training facilities. In that sense, the material is pitched at the graduate or mature student level and could be covered in one or two semesters.

Each chapter has a set of exercises at the end. These exercises are intended to amplify key points raised in the chapter, but instructors could also complement them with questions of their own. I anticipate compiling more exercises and making them available on my Web site (www.perfdynamics.com). Solutions to selected exercises can be found in Appendix E.

Key points that should be retained by both students and practitioners are contained in a box like this one.

Prerequisites and Limitations

This is a book about performance analysis, not performance tuning. The world is already full of books explaining how to tune this or that application on this or that platform. Whereas performance tuning is about particulars, the power of performance analysis comes from discerning general principals. General principals are often best detected at the system level. The payoff is that a generalizable analysis technique learned once will find application in solving a wide variety of future performance problems.

Good analysis requires clarity of thought, and clear thinking benefits from the structure of formalism. The formalism used throughout this book is queueing theory or what might be more accurately termed *queueing theory lite*. By that I mean the elements of queueing theory are presented in a minimalist style without the need for penetrating many of the complexities of mathematical queueing theory, but without loss of correctness. That said, a knowledge of mathematics at the level of high-school algebra is assumed throughout the text (it is hard to imagine doing any kind of meaningful performance analysis without it), and those readers exposed to introductory probability and calculus will find most of the concepts transparent.

Queueing theory algorithms are encoded into PDQ. This frees the performance analyst to focus on the application of queueing concepts to the problem at hand. Inevitably, there is a price for this freedom. The algorithms contain certain assumptions that facilitate the solution of queueing models. One of these is the *Poisson* assumption. In probability theory, the Poisson distribution is associated with events which are statistically *random* (like the clicks of a Geiger counter). PDQ assumes that arrivals into a queue and departures from the service center are random. How well this assumption holds up against behavior of a real computer system will impact the accuracy of your analysis.

In many cases, it holds up well enough that the assumption does not need to be scrutinized. More often, the accuracy of your measurements is the more important issue. All measurements have errors. Do you know the magnitude of the errors in your performance data? See Sect. 2.8 in Chapter 2 (was Appendix D). In those cases where there is doubt about the Poisson assumption, Sect. 2.9 of Chapter 2 (was Appendix D) provides a test together with a Perl script to analyze your data for randomness. One such case is packet queueing.

Internet packets, for example, are known to seriously violate the Poisson assumption [See Park and Willinger 2000]. So PDQ cannot be expected to give accurate performance predictions in that case, but as long as the performance analysis is conducted at the transaction or connection level (as we do in Chap. 12), PDQ is applicable. For packet level analysis, alternative performance tools such simulators (see e.g., NS-2 <http://www.isi.edu/nsnam/ns/>) are a better choice. One has to take care, however, not to be lulled into a false sense of security with simulators. A simulation is assumed to be more accurate because it allows you to construct a faithful representation of the real computer system by accounting for every component—sometimes including the proverbial kitchen sink. The unstated fallacy is that complexity equals completeness. An example of the unfortunate consequences that can ensue from ignoring this point is noted in Sect. 3.7.

Even in the era of simulation tools, you still need an independent framework to validate the results. PDQ can fulfill that role. Otherwise, your simulation stands in danger of being just another pseudo-random number generator. That PDQ can act like an independent framework in which to assess your data (be it from measurement or simulation) is perhaps its most important role. In that sense, the very act of modeling can be viewed as an organizing principle in its own right. *A fortiori*, the insights gained by merely initiating the construction of a PDQ model may be more important than the results it produces.

Acknowledgments

Firstly, I would like to thank the alumni of my computer performance analysis classes, including *Practical Performance Methods* given at Stanford University (1997–2001), UCLA Extension Course 819.328 *Scalable Server Performance*

and *Capacity Planning*, the many training classes given at major corporations, and the current series *Guerrilla Capacity Planning* sponsored by Performance Dynamics. Much of their feedback has found its way into this book. My Stanford classes replaced those originally given by Ed Lazowska, Ken Sevcik, and John Zahorjan. I finally attended their 1993 Stanford class, several years after reading their classic text [Lazowska et al. 1984]. Their approach inspired mine.

Peter Harding deserves all the credit for porting my C implementation of PDQ to Perl. Several people said they would do it (including myself), but only Peter delivered.

Ken Christensen, Robert Lane, David Latterner, and Pedro Vazquez reviewed the entire manuscript and made many excellent suggestions that improved the final content. Jim Brady and Steve Jenkin commented on Appendix A and Chap. 6, respectively. Ken Christensen also kindly provided me with a copy of Erlang's first paper. An anonymous reviewer helped tidy up some of the queue-theoretic discussion in Chaps. 4 and 5. Myron Hlynka and Peter Taylor put my mind at rest concerning the recent controversial claim that Jackson's 50-year-old theorem (Chap. 5) was invalid.

Giordano Beretta rendered his expert scientific knowledge of image processing as well as a monumental number of hours of computer labor to improve the quality of the illustrations. His artistic flair reveals itself in Fig. 4.1. Andrew Trevorrow deserves a lot of thanks, not only for porting and maintaining the OzTeX implementation of L^AT_EX 2_ε on MacOS, but for being very responsive to email questions. The choice of OzTeX was key to being able to produce camera-ready copy in less than a year. Mirko Fluher kindly provided remote access to his Linux system in Melbourne, Australia.

It is a genuine pleasure to acknowledge the cooperation and patience of my editor Ralf Gerstner, as well as the excellent technical support of Frank Holzwarth and Jacqueline Lenz at Springer-Verlag in Heidelberg. Tracey Wilbourn meticulously copyedited the penultimate manuscript and Michael Reinfarth of LE-TeX GbR in Leipzig handled the final production of the book.

Aline and Topin Dawson provided support and balance during the otherwise intense solitary hours spent composing this book. My father tolerated several postponed trans-Pacific visits during the course of this project. Only someone 95 years young has that kind of patience.

I would also like to take this opportunity to thank the many diligent readers who contributed to the errata for *Practical Performance Analyst* [Gunter 2000a]. In alphabetical order they are: M. Allen, A. Bondi, D. Chan, K. Christensen, A. Cockcroft, L. Dantzler, V. Davis, M. Earp, W.A. Gunther, I.S. Hobbs, P. Kraus, R. Lane, T. Lange, P. Lauterbach, C. Millsap, D. Molero, J. A. Nolasco-Flores, W. Pelz and students, H. Schwetman, P. Sinclair, D. Tran, B. Vestermark, and Y. Yan. I trust the errata for this book will be much shorter.

And finally to you, dear reader, thank you for purchasing this book and reading this far. Don't stop now!

Warranty Disclaimer

No warranties are made, express or implied, that the information in this book and the associated computer programs are error free, or are consistent with any particular standard of merchantability, or that they will meet your requirements for any particular application. They should not be relied upon for solving a problem the incorrect solution of which could result in injury to a person or loss of property. The author disclaims all liability for direct or consequential damages resulting from the use of this book.

Palomares Hills, California
July, 2004

N.J.G.

Contents

Preface to Second Edition	vii
--	-----

Preface to First Edition	xi
---------------------------------------	----

Part I Preliminary Concepts

1	Why Queues Rule Performance Analysis	3
1.1	Introduction	3
1.2	Buffers Are Queues	3
1.3	Modeling Efficiencies	4
1.4	Bandwidth and Latency Are Related	8
1.5	Stretch Factor	14
1.6	How Long Should My Queue Be?	15
1.7	A Brief History of Buffers	18
2	Measurement Tools and Techniques	23
2.1	Steady as She Goes	23
2.2	Performance Counters and Objects	26
2.3	Java Bytecode Instrumentation	27
2.4	Generic Performance Tools	27
2.5	Displaying Performance Metrics	28
2.6	Storing Performance Metrics	30
2.7	Performance Prediction Tools	30
2.8	How Accurate Are Your Data?	31
2.9	Are Your Data Poissonian?	31
2.10	Performance Measurement Standards	34
3	Time—The Zeroth Performance Metric	37
3.1	Introduction	37
3.2	What Is Time?	38

3.2.1	Physical Time	39
3.2.2	Synchronization and Causality	39
3.2.3	Discrete and Continuous Time	40
3.2.4	Time Scales	40
3.3	What Is a Clock?	42
3.3.1	Physical Clocks	42
3.3.2	Distributed Physical Clocks	43
3.3.3	Distributed Processing	43
3.3.4	Binary Precedence	44
3.3.5	Logical Clocks	44
3.3.6	Clock Ticks	46
3.3.7	Virtual Clocks	47
3.4	Representations of Time	48
3.4.1	In the Beginning	48
3.4.2	Making a Date With Perl	48
3.4.3	High-Resolution Timing	50
3.4.4	Benchmark Timers	52
3.4.5	Crossing Time Zones	53
3.5	Time Distributions	56
3.5.1	Gamma Distribution	56
3.5.2	Exponential Distribution	57
3.5.3	Poisson Distribution	59
3.5.4	Server Response Time Distribution	60
3.5.5	Network Response Time Distribution	62
3.6	Timing Chains and Bottlenecks	63
3.6.1	Bottlenecks and Queues	64
3.6.2	Distributed Instrumentation	65
3.6.3	Disk Timing Chains	65
3.6.4	Life and Times of an NFS Operation	66
3.7	Failing Big Time	68
3.7.1	Hardware Availability	68
3.7.2	Tyranny of the Nines	69
3.7.3	Hardware Reliability	69
3.7.4	Mean Time Between Failures	71
3.7.5	Distributed Hardware	72
3.7.6	Components in Series	72
3.7.7	Components in Parallel	73
3.7.8	Software Reliability	73
3.8	Metastable Lifetimes	75
3.8.1	Microscopic Metastability	75
3.8.2	Macroscopic Metastability	78
3.8.3	Metastability in Networks	78
3.8.4	Quantum-like Phase Transitions	80
3.9	Review	80
	Exercises	81

Part II Basic Queueing Theory for PDQ

4	Getting the Jump on Queueing	85
4.1	Introduction	85
4.2	What Is a Queue?	86
4.3	The Grocery Store—Checking It Out	87
4.3.1	Queueing Analysis View	87
4.3.2	Perceptions and Deceptions	88
4.3.3	The Post Office—Snail Mail	89
4.4	Fundamental Metric Relationships	89
4.4.1	Performance Measures	90
4.4.2	Arrival Rate	92
4.4.3	System Throughput	93
4.4.4	Nodal Throughput	94
4.4.5	Relative Throughput	94
4.4.6	Service Time	96
4.4.7	Service Demand	96
4.4.8	Utilization	97
4.4.9	Residence Time	97
4.5	Little's Law Means a Lot	98
4.5.1	A Little Intuition	99
4.5.2	A Visual Proof	100
4.5.3	Little's Microscopic Law	104
4.5.4	Little's Macroscopic Law	105
4.6	Unlimited Request (Open) Queues	106
4.6.1	Single Server Queue	106
4.6.2	Measured Service Demand	107
4.6.3	Queueing Delays	107
4.6.4	Twin Queueing Facility	112
4.6.5	Parallel Queues	113
4.6.6	Dual Server Queue—Heuristic Analysis	115
4.7	Multiserver Queue	119
4.7.1	Erlang's C Formula	120
4.7.2	Accuracy of the Heuristic Formula	122
4.7.3	Erlang's B Formula	122
4.7.4	Erlang Algorithms in Perl	124
4.7.5	Dual Server Queue—Exact Analysis	127
4.8	Limited Request (Closed) Queues	128
4.8.1	Closed Queueing Facility	128
4.8.2	Interactive Response Time Law	129
4.8.3	Repairman Algorithm in Perl	131
4.8.4	Response Time Characteristic	131
4.8.5	Throughput Characteristic	133
4.8.6	Finite Response Times	135

4.8.7	Approximating Closed Queues	136
4.9	Shorthand for Queues	140
4.9.1	Queue Schematics	140
4.9.2	Kendall Notation	141
4.10	Comparative Performance	142
4.10.1	Multiserver Versus Uniserver	143
4.10.2	Multiqueue Versus Multiserver	143
4.10.3	The Envelope Please!	145
4.11	Generalized Servers	146
4.11.1	Infinite Capacity (IS) Server	147
4.11.2	Exponential (M) Server	148
4.11.3	Deterministic (D) Server	148
4.11.4	Uniform (U) Server	149
4.11.5	Erlang- k (E_k) Server	149
4.11.6	Hypoexponential (<i>Hypo-k</i>) Server	150
4.11.7	Hyperexponential (H_k) Server	150
4.11.8	Coxian (<i>Cox-k</i>) Server	151
4.11.9	General (G) Server	152
4.11.10	Pollaczek–Khintchine Formula	153
4.11.11	Polling Systems	155
4.11.12	Queues and Scalability	157
4.12	Review	159
	Exercises	159
5	Queueing Systems for Computer Systems	161
5.1	Introduction	161
5.2	Types of Circuits	162
5.3	Poisson Properties	164
5.3.1	Poisson Merging	164
5.3.2	Poisson Branching	165
5.3.3	Poisson PASTA	166
5.4	Open-Circuit Queues	166
5.4.1	Series Circuits	167
5.4.2	Feedforward Circuits	167
5.4.3	Feedback Circuits	168
5.5	Jackson’s Theorem	171
5.5.1	Jackson Network Traffic	173
5.5.2	Jackson Node Traffic	173
5.5.3	Routing Requests in PDQ	175
5.5.4	Parallel Queues in Series	177
5.5.5	Multiple Workloads in Open Circuits	180
5.6	Closed-Circuit Queues	181
5.6.1	Arrival Theorem	182
5.6.2	Iterative MVA Algorithm	183
5.6.3	Approximate Solution	185

5.7	Visit Ratios and Routing Probabilities	189
5.7.1	Visit Ratios and Open Circuits	189
5.7.2	Visit Ratios and Closed Circuits	191
5.8	Multiple Workloads in Closed Circuits	192
5.8.1	Workload Classes	192
5.8.2	Baseline Analysis	192
5.8.3	Aggregate Analysis	194
5.8.4	Component Analysis	197
5.9	Operating Systems and Schedulers	199
5.9.1	Time-Share Scheduler	199
5.9.2	Fair-Share Scheduler	201
5.9.3	Priority Scheduling	204
5.9.4	Thread Scheduler	206
5.10	Rules for Applying Queueing Models	207
5.10.1	MVA Is a Style of Thinking	207
5.10.2	BCMP Rules	208
5.10.3	Service Classes	209
5.10.4	Limitations	210
5.11	Review	212
	Exercises	213
6	Linux Load Average	215
6.1	Introduction	215
6.1.1	Load Average Reporting	216
6.1.2	What Is an “Average” Load?	217
6.2	A Simple Experiment	218
6.2.1	Experimental Results	219
6.2.2	Submerging Into the Kernel	221
6.3	Load Calculation	222
6.3.1	Fixed-Point Arithmetic	223
6.3.2	Magic Numbers	224
6.3.3	Empty Run-Queue	226
6.3.4	Occupied Run-Queue	226
6.3.5	Exponential Damping	228
6.4	Steady-State Averages	231
6.4.1	Time-Averaged Queue Length	232
6.4.2	Linux Scheduler Model	232
6.5	Load Averages and Trend Visualization	235
6.5.1	What Is Wrong with Load Averages	235
6.5.2	New Visual Paradigm	235
6.5.3	Application to Workload Management	237
6.6	Review	237
	Exercises	238

7 Performance Bounds and Log Jams 239

7.1 Introduction 239

7.2 Out of Bounds in Florida 239

 7.2.1 Load Test Results 240

 7.2.2 Bottlenecks and Bounds 240

7.3 Throughput Bounds 241

 7.3.1 Saturation Throughput 241

 7.3.2 Uncontended Throughput 242

 7.3.3 Optimal Load 243

7.4 Response Time Bounds 244

 7.4.1 Uncontended Response Time 244

 7.4.2 Saturation Response Time 244

 7.4.3 Worst-Case Response Bound 246

7.5 Meanwhile, Back in Florida 246

 7.5.1 Balanced Bounds 247

 7.5.2 Balanced Demand 247

 7.5.3 Balanced Throughput 248

7.6 The X-Files: Encounters with Performance Aliens 249

 7.6.1 X Window Architecture 249

 7.6.2 Production Environment 250

7.7 Close Encounters of the Performance Kind 251

 7.7.1 Close Encounters I: Rumors 251

 7.7.2 Close Encounters II: Measurements 251

 7.7.3 Close Encounters III: Analysis 252

7.8 Performance Aliens Revealed 254

 7.8.1 Out of Sight, Out of Mind 254

 7.8.2 Log-Jammed Performance 256

 7.8.3 To Get a Log You Need a Tree 256

7.9 X11 Window Scalability 258

 7.9.1 Measuring Sibling X-Events 258

 7.9.2 Superlinear Response 259

7.10 Review 260

Exercises 260

Part III Practical Application of PDQ

8 Pretty Damn Quick—A Slow Introduction 263

8.1 Introduction 263

8.2 How to Build PDQ Circuits 263

8.3 Inputs and Outputs 263

 8.3.1 Setting Up PDQ 264

 8.3.2 Some General Guidelines 266

8.4 Simple Annotated Example 266

 8.4.1 Creating the PDQ Model 266

8.4.2	Reading the PDQ Report	268
8.4.3	Validating the PDQ Model	268
8.5	Classic Queues in PDQ	271
8.5.1	Delay Node in PDQ	271
8.5.2	$M/M/1$ in PDQ	271
8.5.3	$M/M/m$ in PDQ	271
8.5.4	$M/M/1/N/N$ in PDQ	272
8.5.5	$M/M/m/N/N$ in PDQ	272
8.5.6	Feedforward Circuits in PDQ	272
8.5.7	Feedback Circuits in PDQ	272
8.5.8	Parallel Queues in Series	276
8.5.9	Multiple Workloads in PDQ	276
8.5.10	Priority Queueing in PDQ	276
8.5.11	Load-Dependent Servers in PDQ	278
8.6	Review	289
	Exercises	289
9	Multicomputer Analysis with PDQ	291
9.1	Introduction	291
9.2	Multiprocessor Architectures	292
9.2.1	Symmetric Multiprocessors	293
9.2.2	Multiprocessor Caches	294
9.2.3	Cache Bashing	295
9.3	Multiprocessor Models	296
9.3.1	Single-Bus Models	297
9.3.2	Processing Power	298
9.3.3	Multiple-Bus Models	300
9.3.4	Cache Protocols	302
9.3.5	Iron Law of Performance	305
9.4	Multicomputer Models	307
9.4.1	Parallel Query Cluster	307
9.4.2	Query Saturation Method	311
9.5	Review	316
	Exercises	316
10	How to Scale an Elephant with PDQ	317
10.1	An Elephant Story	317
10.1.1	What Is Scalability?	318
10.1.2	SPEC Multiuser Benchmark	319
10.1.3	Steady-state Measurements	321
10.2	Parts of the Elephant	321
10.2.1	Service Demand Part	322
10.2.2	Think Time Part	322
10.2.3	User Load Part	322
10.3	PDQ Scalability Model	323

10.3.1	Interpretation	324
10.3.2	Amdahl's Law	326
10.3.3	The Elephant's Dimensions	328
10.4	Review	329
	Exercises	330
11	Client/Server Analysis with PDQ	331
11.1	Introduction	331
11.2	Client/Server Architectures	332
11.2.1	Multitier Environments	333
11.2.2	Three-Tier Options	333
11.3	Benchmark Environment	335
11.3.1	Performance Scenarios	335
11.3.2	Workload Characterization	337
11.3.3	Distributed Workflow	339
11.4	Scalability Analysis with PDQ	340
11.4.1	Benchmark Baseline	341
11.4.2	Client Scaleup	346
11.4.3	Load Balancer Bottleneck	349
11.4.4	Database Server Bottleneck	349
11.4.5	Production Client Load	349
11.4.6	Saturation Client Load	350
11.4.7	Per-Process Analysis	350
11.5	Review	353
	Exercises	354
12	Web Application Analysis with PDQ	355
12.1	Introduction	355
12.2	HTTP Protocol	355
12.2.1	HTTP Performance	360
12.2.2	HTTP Analysis Using PDQ	361
12.2.3	Fork-on-Demand Analysis	361
12.2.4	Prefork Analysis	363
12.3	Two-Tier PDQ Model	369
12.3.1	Data and Information Are Not the Same	369
12.3.2	HTTPd Performance Measurements	369
12.3.3	Java Performance Measurements	369
12.4	Middleware Analysis Using PDQ	372
12.4.1	Active Client Threads	372
12.4.2	Load Test Results	374
12.4.3	Derived Service Demands	375
12.4.4	Preliminary PDQ Model	375
12.4.5	Adding Hidden Latencies in PDQ	379
12.4.6	Adding Overdriven Throughput in PDQ	381
12.5	Review	384

Exercises	384
13 Virtual Machine Analysis with PDQ	387
13.1 Introduction	387
13.2 The Virtual Machine Spectrum	388
13.3 Micro-VMM Scale: Hyperthreading	390
13.3.1 Controlled Measurements	392
13.3.2 PDQ Model of Micro-VMM	395
13.4 Meso-VMM Scale: Hypervisors	397
13.4.1 Performance Monitoring Tools	404
13.4.2 Controlled Measurements	405
13.5 Macro-VMM Scale: Clouds and P2P	407
13.5.1 Macro-VM Polling	408
13.5.2 Scalability Analysis Using PDQ	409
13.6 Cloud Computing Models	411
13.6.1 Fixed-Size Bounds	413
13.6.2 Harmonic Bounds	416
13.6.3 Scaled-Size Bounds	417
13.6.4 Erlang Model	418
13.6.5 LogP Model	419
13.7 Summary	421

Part IV Appendices

A Thanks for No Memories	425
A.1 Life in the Markov Lane	425
A.2 Exponential Invariance	426
A.3 Shape Preservation	428
A.4 A Counterexample	428
B Compendium of Queueing Equations	431
B.1 Fundamental Metrics	431
B.2 Queueing Delays	432
C Units and Abbreviations	433
C.1 SI Prefixes	433
C.2 Time Suffixes	433
C.3 Capacity Suffixes	433
D Perl PDQ Manual	435
D.1 Introduction	435
D.2 Perl PDQ Module	435
D.2.1 PDQ Data Types	435
D.2.2 PDQ Global Variables	436

D.2.3	PDQ Functions	437
D.3	Function Synopses	437
D.3.1	pdq::CreateClosed	437
D.3.2	pdq::CreateMultiNode	438
D.3.3	pdq::CreateNode	438
D.3.4	pdq::CreateOpen	439
D.3.5	pdq::CreateSingleNode	439
D.3.6	pdq::GetLoadOpt	440
D.3.7	pdq::GetQueueLength	441
D.3.8	pdq::GetResidenceTime	441
D.3.9	pdq::GetResponse	442
D.3.10	pdq::GetThruMax	443
D.3.11	pdq::GetThruput	443
D.3.12	pdq::GetUtilization	444
D.3.13	pdq::Init	444
D.3.14	pdq::Report	445
D.3.15	pdq::SetDebug	446
D.3.16	pdq::SetDemand	447
D.3.17	pdq::SetTUnit	448
D.3.18	pdq::SetVisits	448
D.3.19	pdq::SetWUnit	449
D.3.20	pdq::Solve	449
D.4	Perl Scripts	450
D.5	PDQ Scripts	451
D.6	Installing the PDQ Module	451
E	Solutions to Selected Exercises	453
	Bibliography	457
	Index	465