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

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 Jackon'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/ppdqerrata.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 endebted to the alumni of Performance Dynamics Company classes, and other diligent readers, who contributed errata for the first edition at www.perfdynamics.com/iBook/ppdqerrata.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 N.J.G.
December, 2010

```
perl -le '@q=("\120\145\162\154","\120\104\121");
$s="\115\141\171\040\171\157\165\162\040@q\040\bs\040\142\145";
$q[0]=~s/e/ea/;$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 LATEX 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* [Gunther 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. Nolazco-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 N.J.G.
July, 2004

Contents

Part II Basic Queueing Theory for PDQ