

Five Steps to Solving Software Performance Problems

Lloyd G. Williams, Ph.D.

Connie U. Smith, Ph.D.

June, 2002

Contents

Introduction	1
Five Steps to Improved Performance	2
Step 1: Figure Out Where You Need to Be	2
Step 2: Determine Where You Are Now	3
Step 3: Decide Whether You Can Achieve Your Objectives	3
Step 4: Develop a Plan for Achieving Your Objectives	4
Step 5: Conduct an Economic Analysis of the Project	5
Cost-Benefit Analysis	5
An Ounce of Prevention	6
For More Information	6
Summary	7
References	7
About the Authors	8

Introduction

We all know that performance—responsiveness and scalability—is a make-or-break quality for software. Poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations.

Nearly everyone runs into performance problems at one time or another. Today’s software development organizations are being asked to do more with less. In many cases, this means upgrading legacy applications to accommodate a new infrastructure (e.g., a Web front-end), improve response time or throughput, or both. One group that we worked with found themselves needing to increase the throughput of a legacy data-acquisition system by a factor of 10. While a ten-fold increase in hardware capacity might have solved the problem, it was prohibitively expensive. They needed to find most of that improvement in the software.†

Another company (we’ll call then Technology, Inc.) found themselves paying 7 figure penalties for not meeting contractual performance requirements. Developers said that the reason was that a key software component would not scale enough to meet that customers higher demand. After months of trying to tune the system, developers informed management that it would not be possible, that it would require re-writing the code using newer object-oriented techniques (“a Way Cool solution” and more fun than tuning the old code). While management was willing to bite the bullet and commit to a re-engineering project, they asked us to confirm that the Way Cool solution would indeed solve their performance problems.

Newly-developed software also frequently exhibits performance problems. In an ideal world, performance would be engineered into software starting early in the development process. The reality, however, is that budget and schedule constraints often lead developers, particularly devotees of the “agile” methodologies, to adopt a “make it run, make it run right, make it run fast” strategy. The result is that, somewhere near the end of the project, performance problems appear. Another group we observed used this strategy on a transaction-processing system. The goal was to complete a transaction in 10 seconds. When the testing group tried to perform integration tests, they discovered that the best response times were about 60 seconds with some transactions taking more than 200 seconds.

Regardless of the source, when performance problems occur, they must be fixed immediately. In response, the project often goes into “crisis mode” in an attempt to tune or even redesign the software to meet performance objectives. In these situations, it is vital to maximize the performance and capacity payoff of your tuning efforts.

This article presents a systematic, quantitative approach to performance tuning that helps you quickly find problems, identify potential solutions, and prioritize your efforts to achieve the greatest improvements with the least effort. The steps below are adapted from the tuning process described in [Smith and Williams 2002].

†. Note: Details of case histories have been disguised to protect confidentiality.

Five Steps to Improved Performance

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

—Sherlock Holmes

Sherlock Holmes never solved a software performance mystery, but he was right. It is a mistake to begin tuning before you have the data you need to isolate problems and select appropriate solutions. Without this information, you may find yourself spending a great deal of effort making software changes that produce little or no improvement.

With the limited time available for tuning, and the extent of improvement that is typically needed, it is vital to focus on those areas that have the highest potential for payoff, rather than to expend effort on improvements that have a negligible overall effect. Remember the group that had response times of 60–200 seconds on transactions that should have taken 10 seconds? They spent more than a week parallelizing components of the transactions to reduce the total time spent interacting with the back-end database. This “improvement” saved less than one second.

Technology Inc. spent six months tuning various portions of the code without achieving their objective because, rather than using measurements for guidance, they also focused their efforts on portions of the code that appeared to be inefficient but actually had little impact on overall performance.

These experiences are not unusual. You may find yourself spending weeks or even months tuning the software without seeing significant improvements. It is usually caused by speculation on the cause of performance problems and a random approach to improvements. The steps below allow you to quickly identify problems, focus your effort on the areas of the software that have the greatest potential for improvement and estimate the relative payoff for potential improvements.

Step 1: Figure Out Where You Need to Be

The cat only grinned as Alice approached... “Cheshire-Puss,” she began, rather timidly... “Would you tell me, please, which way I ought to go from here?”

“That depends a good deal on where you want to go to,” said the Cat.

—*Alice’s Adventures in Wonderland*, by Lewis Carroll

While it may seem obvious that, like Alice, you need to know where you’re going before you set out, it’s surprising how many projects don’t have well-defined performance objectives. When we ask what the performance objectives are, we often get a response like: “As fast as possible.” An objective such as this simply isn’t useful. For example, how do you know when (if?) you’ve achieved it?

You should define precise, quantitative, measurable performance objectives. You can express performance objectives in several ways, including response time, throughput, or constraints on resource usage. Some examples are: “The response time for a transaction should be one second or less with up to 1,000 users.” or “CPU utilization should be less than 65% for a peak load of 2,000 events per second.”

When defining performance objectives, don't forget that your needs may change over the product's lifetime. For example, your current performance objective may be to process 10,000 events per second. However, in two years, you may need to be able to process 30,000 events per second. It is a good idea to consider future uses of your software so that you can anticipate these changes and build in the necessary scalability.

Step 2: Determine Where You Are Now

"I have an existential map. It has 'You are here' written all over it."

—Steven Wright

Begin by quantifying the problems and identifying their causes. Which uses of the software are causing problems? Typically, these are the most frequent uses. If you don't have good documentation (like UML sequence diagrams) for these uses, it's a good idea to do this now. Then, identify the workloads (transaction or customer-call volumes, event arrival rates, and so on) that are causing problems.

These are steps in our PASASM approach for Performance Assessment of Software Architectures [Williams and Smith 2002a]. We have found that starting with an understanding and assessment of the software architecture offers better opportunities for achieving performance and scalability objectives. We have found that it is best to understand the software's purpose, architecture, and processing steps to:

- determine if the architecture can support scalability goals,
- determine if problems can be fixed with tuning and which parts to tune (before spending months trying to no avail),
- to project the scalability, throughput and response time with the tuning changes.
- to determine whether MIPS reduction requires re-design or if tuning is sufficient

Focusing on the architecture provides more and potentially greater options for performance improvement than tuning. By focusing on the architecture, you can identify opportunities for changing *what* the software does. For example, by studying the architecture, you may discover that some processing can be moved off the Fast Path and performed by a background process. With tuning, you are only able to make changes to the code. That is you can change *how* the software does something but not *what* it does.

Now you are ready to take measurements of these scenarios under operational conditions. This will allow you to understand the system performance parameters such as CPU utilization, I/O rates and average service time, network utilization, message size, and so on. This, in turn, will allow you to identify the bottleneck(s)—the device(s) with the highest utilization. Then, profile the processing steps in the performance scenarios to identify the "hot spots." The overall impact of improvements gained from addressing these hot spots will be far greater than that of random improvements.

Step 3: Decide Whether You Can Achieve Your Objectives

"Denial ain't just a river in Egypt. "

—Mark Twain (1835-1910)

Before you dive into tuning your software, it's a good idea to stop and see if you can actually achieve your objectives by tuning. If the difference between where you are now and where you need to be is small, then tuning will probably help. You may even be able to achieve your performance objectives without making changes to the code by tuning operating system parameters, network configuration, file placements, and so on. If not, you'll need to tune the software.

Performance remedies range from low-cost (and, usually, low-payoff) automatic optimization options (e.g., compiler optimizations) to high-cost (and, potentially, high-payoff) software refactoring or re-creation using the performance principles and patterns discussed in [Williams and Smith 2002a]. Intermediate options are to optimize algorithms and data structures and/or modify program code to use more efficient constructs.

Once you've identified the hot spots, some simple calculations will help you decide if you can meet your performance objectives by tuning the software. For example, if your bottleneck is a processing step that processes 250 messages per second (4 msec per message) and you need it to handle 500 messages per second, then you need to reduce the time for processing each message to 2 msec. Is this realistic given your measurements and what you know about the software? If not, you may need to explore other alternatives, such as adding more processors. If it is, then you can proceed with confidence as you tune the software.

You might also create software models that depict the end-to-end processing steps for performance scenarios. The models will help you quantify the effects of more complex solutions, particularly those that involve contention for resources.

Models of the Technology, Inc. application showed that the "Way Cool" redesign would provide worse performance than the existing system. However, in constructing the models, we were able to identify a tuning alternative that would get them out of the penalty situation and buy them time to consider what else to do with this application. This tuning opportunity was found because of the models—it had been overlooked in their previous tuning efforts.

Step 4: Develop a Plan for Achieving Your Objectives

"If Stupidity got us into this mess, then why can't it get us out?"

—Will Rogers (1879-1935)

Once you have identified potential remedies, you can rank them based on their payoff. Then apply them starting with those that have the highest payoff.

You determine the quantitative performance improvement by estimating the new resource requirements. Techniques for estimating resource requirements include:

- The estimation techniques in [Smith and Williams 2002].
- Benchmark a prototype and use measurements to update your performance models.
- Use the model to determine a performance target for processing steps then tune until you meet that target.

You also need to quantify the effort to make changes. Create a list of options that compares the performance improvement versus the costs to determine the best candidates. Note that you may not always choose the option with the best performance improvement because other factors may

make it less desirable. For example, the option with the best performance improvement may be risky because it uses an unfamiliar new technology. Or, a change with a high performance payoff may negatively impact another important quality such as maintainability or reliability.

You may elect to correct problems with hardware. Be sure to use the models to evaluate this option too. Sometimes the problem is caused by software and no amount of hardware will correct it. For example, a section of single-threaded code may cause processing to slow to a crawl. Adding more or faster processors will not help in this situation.

Step 5: Conduct an Economic Analysis of the Project

“A billion here, a billion there—pretty soon it adds up to real money.”

—Senator Everett Dirksen (1896-1969)

Upon completion, gather data on the time and cost for the performance analysis, time and for software changes (include coding and testing), hardware costs if applicable, software distribution costs when applicable, and all other costs of the project. Then gather data on the effect of the improvements. These may include savings due to deferred hardware upgrades, staff savings (ie. if fewer staff members are required to use the software), revenue increases (ie. when you can increase the number of orders processed), and so on.

It is also a good idea to review the changes made to see if they could have been predicted earlier in the software's life. For example, when SPE techniques are used during software development models predict the impact of software architecture and design decisions before coding begins. If your problems could have been prevented, compare the costs and benefits of detecting and correcting problems earlier versus detecting them after problems occurred. Then use this data on your next project to determine whether it is worthwhile to spend a little more effort during development to prevent problems.

Cost-Benefit Analysis

Using this process can provide a high payoff for a relatively small cost. The group that needed a ten-fold increase in the throughput of their legacy data-acquisition system followed this process to tune the software rather than simply increase hardware capacity. A ten-fold increase in hardware capacity would have required buying nine new machines. By tuning the software, they were able to meet their performance objective with only four additional machines. The cost of the cost of the quantitative study (including an extensive architecture assessment was approximately \$250,000. The software changes required approximately 2 months of effort from four people. By avoiding the purchase of five additional machines, they saved over \$5,000,000. This savings was for a single system. Since there was a redundant backup as well, the total savings was over \$10,000,000.

What about the group that needed to reduce a 60–200 second transaction time to 10 seconds—the one that followed a random approach to performance improvement? After several months of effort, they “got close.” In the end, they passed the system on to the group responsible for the next release to complete the performance fixes.

Technology, Inc. is no longer in business.

An Ounce of Prevention

Once you run into trouble, tuning the software is likely to be your only choice. However, it's important to realize that a tuned system will rarely, if ever, exhibit the level of performance that you could have achieved by designing in performance from the beginning. For example, in tuning the data-acquisition system, one significant improvement required extensive architectural changes and was deemed infeasible. Had that change been made, it would have been possible to meet the performance objective with one less machine for both the main system and the backup—an additional savings of over \$2,000,000.

The key to achieving optimum performance is to adopt a proactive approach to performance management that anticipates potential performance problems and includes techniques for identifying and responding to those problems early in the process. With a proactive approach, you produce software that meets performance objectives and is delivered on time and within budget, and avoid the project crises brought about by the need for tuning at the end of the project.

Software performance engineering (SPE) is a systematic, quantitative approach to proactively managing software performance [Smith 1990], [Smith and Williams 2002]. SPE is an engineering approach that avoids the extremes of performance-driven development and “fix-it-later.” With SPE, you detect problems early in development, and use quantitative methods to support cost-benefit analysis of various solution options. SPE is a software-oriented approach: it focuses on architecture, design, and implementation choices. It uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements. It also includes techniques for collecting data, principles and patterns for performance-oriented design, and anti-patterns for recognizing and correcting common performance problems.

The costs and benefits of using SPE are discussed in [Williams and Smith 2002b].

For More Information

For more information contact:

Software Engineering Research
264 Ridgeview Lane
Boulder, CO 80302
(303)938-9847
FAX: (303) 443-5279
boulderlgw@aol.com

Performance Engineering Services
PO Box 2640
Santa Fe, NM 87504
(505) 988-3811
FAX: (786) 513-0165
cusmith@perfeng.com
www.perfeng.com

More information on solving and preventing software performance problems is also in the book *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software* [Smith and Williams 2002]. The course *Performance Solutions: Solving Performance Problems Quickly and Effectively*, taught by Dr. Williams and Dr. Smith provides training on the techniques described in this paper. See www.perfeng.com for details on content and schedule.

Summary

Performance—responsiveness and scalability—is a make-or-break quality for software. Poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations. When performance problems occur, they must be fixed immediately. In response, the project often goes into “crisis mode” in an attempt to tune or even redesign the software to meet performance objectives. In these situations, it is vital to maximize the performance and capacity payoff of your tuning efforts.

This article has presented a systematic, quantitative approach to performance tuning that helps you quickly find problems, identify potential solutions, and prioritize your efforts to achieve the greatest improvements with the least effort. The steps are:

1. Figure Out Where You Need to Be
2. Determine Where You Are Now
3. Decide Whether You Can Achieve Your Objectives
4. Develop a Plan for Achieving Your Objectives
5. Conduct an Economic Analysis of the Project

Using this approach has been shown to provide a high payoff for a relatively small cost.

Once you run into trouble, tuning the software is likely to be your only choice. However, it's important to realize that a tuned system will rarely, if ever, exhibit the level of performance that you could have achieved by designing in performance from the beginning. The key to achieving optimum performance is to adopt a proactive approach to performance management that anticipates potential performance problems and includes techniques for identifying and responding to those problems early in the process. With a proactive approach, you produce software that meets performance objectives and is delivered on time and within budget, and avoid the project crises brought about by the need for tuning at the end of the project. Software performance engineering (SPE) provides a systematic, quantitative approach to proactively managing software performance.

References

[Smith 1990] C. U. Smith, *Performance Engineering of Software Systems*, Reading, MA, Addison-Wesley, 1990.

[Smith and Williams 2002] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Boston, MA, Addison-Wesley, 2002.

[Williams and Smith 2002a] L. G. Williams and C. U. Smith, “PASASM: A Method for the Performance Assessment of Software Architectures,” 2002 (submitted for publication).

[Williams and Smith 2002b] L. G. Williams and C. U. Smith, “The Business Case for Software Performance Engineering,” www.perfeng.com.

About the Authors

Dr. Lloyd G. Williams is a principal consultant at Software Engineering Research, where he specializes in the development and evaluation of software architectures to meet quality objectives, including performance, reliability, modifiability, and reusability. His experience includes work on systems in fields such as process control, avionics, telecommunications, electronic funds transfer, Web-based systems, software development tools and environments, and medical instrumentation. Dr. Williams has been a pioneer in the application of Software Performance Engineering (SPE) to object-oriented systems. He is the author of numerous technical papers and has presented professional development seminars and consulted on software development for more than 100 organizations worldwide.

Dr. Connie U. Smith a principal consultant of the Performance Engineering Services Division of L&S Computer Technology, Inc., is known for her work in defining the field of SPE and integrating SPE into the development of new software systems. Dr. Smith received the Computer Measurement Group's prestigious AA Michelson Award for technical excellence and professional contributions for her SPE work. She also authored the original SPE book: *Performance Engineering of Software Systems*, published in 1990 by Addison-Wesley, and approximately 100 scientific papers. She is the creator of the *SPE•ED*TM performance engineering tool. She has over 25 years of experience in the practice, research and development of the SPE performance prediction techniques.

Together, Drs. Williams and Smith have over 50 years of experience in software development. They have worked together for more than 15 years to help clients design and implement software that meets performance objectives. They have published numerous technical papers and articles, and are the authors of *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, published by Addison-Wesley.