Maximizing Performance and Scalability with IBM WebSphere

ADAM NEAT

Apress™

Maximizing Performance and Scalability with IBM WebSphere

ISBN (pbk): 1-59059-130-5

Printed and bound in the United States of America 12345678910

Technical Reviewer: Matt Hogstrom

Editorial Board: Steve Anglin, Dan Appleman, Gary Cornell, James Cox, Tony Davis, John Franklin, Chris Mills, Steven Rycroft, Dominic Shakeshaft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Tracy Brown Collins

Copy Editors: Kim Wimpsett, Nicole LeClerc

Production Manager: Kari Brooks

Production Editor: Laura Cheu

Proofreader: Linda Seifert

Compositor: Gina M. Rexrode

Indexer: Carol Burbo

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 1

The Need for Performance

Sometimes, people take decreasing hardware and infrastructure costs for granted. System managers typically neglect that to better an application's overall performance (reducing end user–facing response times), it's just as important to look at the *way* something is configured or tuned than it is to add horsepower (hardware) to your platform.

For WebSphere-based Java 2 Enterprise Edition (J2EE) applications, this is an important concept to understand. Poorly performing applications that are simply masked by ultra-fast hardware will undoubtedly be, from a resource utilization point of view, the catalyst for other problems in the future.

This chapter presents several models that exist to help system and application managers understand and appreciate the financial gains and the business benefits from a well-optimized and correctly scaled environment.

Quantifying Performance

What's the definition of *performance* and an *optimized system*, and why are they so important?

Is high performance all about a warm-and-fuzzy feeling we get when our environments and platforms are operating in an ultra-tuned fashion, or is it more about the direct or indirect financial implications of a well-tuned platform? Maybe it's a bit of both, but the quantified and tangible reasons for performance being so important are primarily financial considerations.

Consider the following example: You're operating a 1,000-user environment on a pair of Sun F4800 servers, and the average response time for each user transaction is around five seconds. If your business stakeholders informed you they were going to heavily market the application you manage—with a suggested 100-percent increase in customers—then you'd probably initially think of simply doubling the Central Processing Unit (CPU), memory, and physical number of systems. Or you'd simply look at how the application operates and is tuned.

From personal experience, most people focus on increasing the hardware. System load from customer usage and computing power typically correlate with one another. The problem with this scenario, apart from the de facto upgrade approach being costly, is that it doesn't promote operational best practices. Poor operational best practices (whether it's operations architecture or operations engineering) are some of the worst offenders in spiraling Information Technology (IT) costs today. Operations engineering and performance go hand in hand.

An operations methodology on one side is about the processes and *methodologies* incorporated into the scope of your operations architecture. These processes and methodologies are what drives proactive performance management and optimizations programs, all of which I'll discuss later in this chapter.

The other side of this is a performance methodology. The performance methodology is basically the approach you take in order to implement and drive an increase in application, system, or platform performance.

Figure 1-1 highlights the two methodologies, operations and performance. Although they're both the key drivers in achieving a well-performing system, too much or too little of either can incur additional costs.

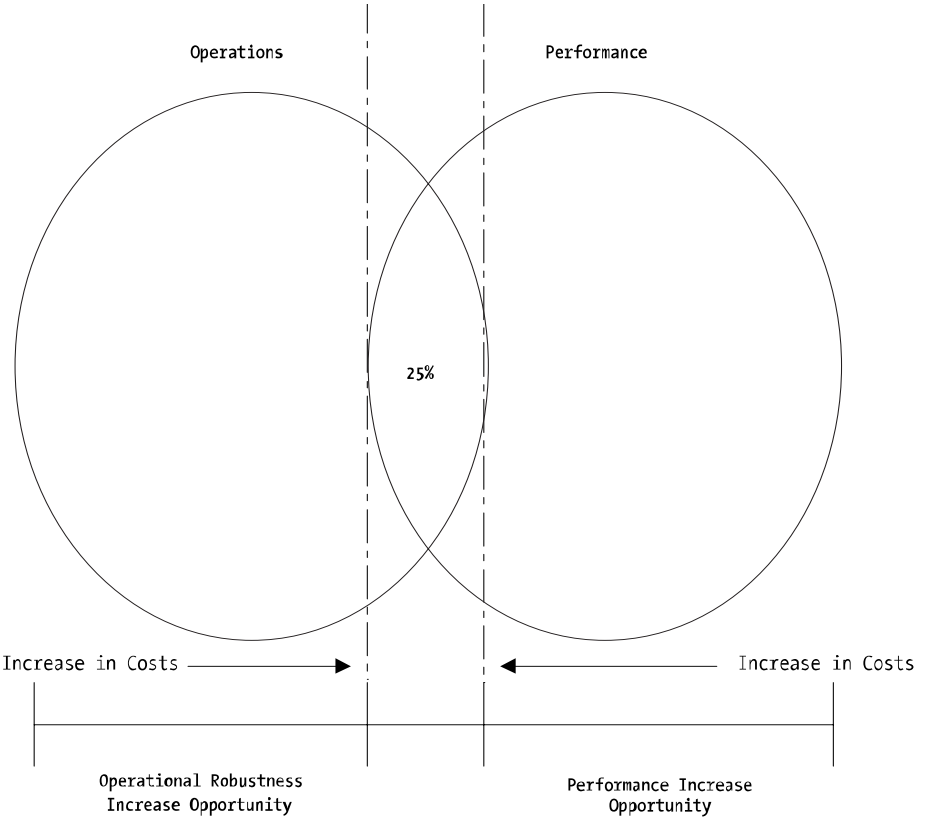

Figure 1-1. The operations and performance methodologies intersect

Figure 1-1 shows a fairly proportioned amount of both operations and performance methodologies. Note the cost scale underneath each methodology.

Let's say that Figure 1-1 represents a model that works to achieve a 25-percent improvement in performance each quarter (every three months). If you wanted to increase this to a 75-percent improvement in performance in a quarter, both operations and performance methodologies would come closer to one another (as depicted in Figure 1-2), and the costs associated with conducting performance management and operational process changes would increase.

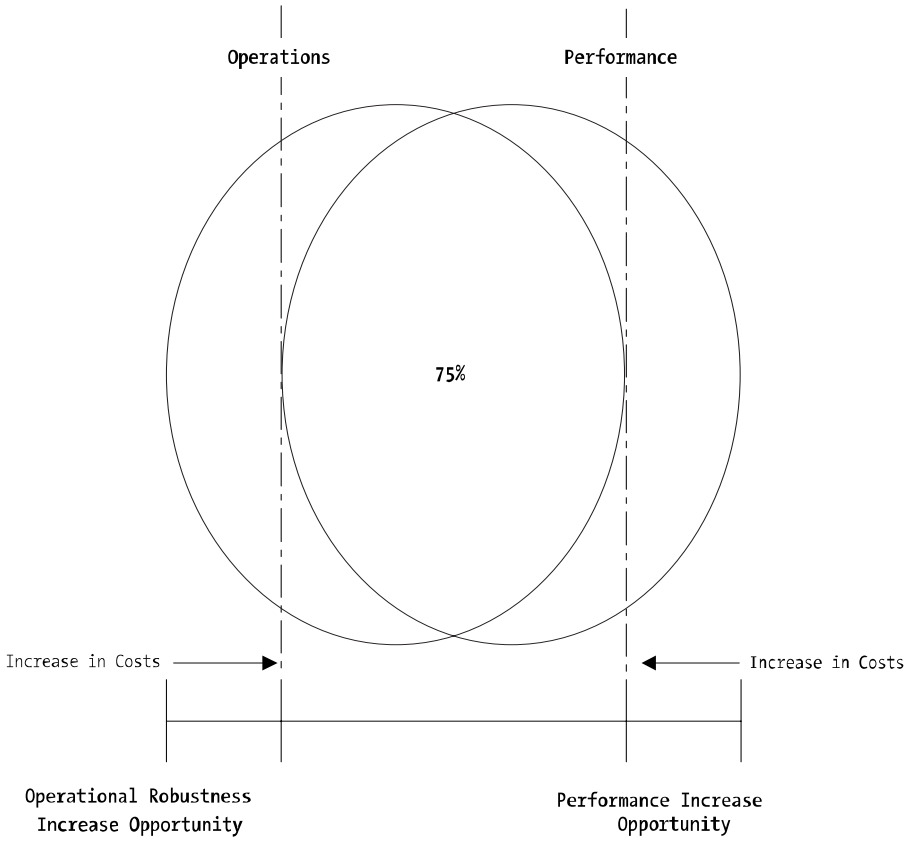

Figure 1-2. The operations and performance methodologies intersect, with an increase in performance

Although there's no problem with attempting to achieve this type of performance increase, costs for both methodologies increase. The question that needs to be answered is, "How much cost is acceptable to achieve a 75-percent increase in performance?"

Would it mean that because of the performance, the volume of transactions could increase, amounting to more customers or users utilizing the application and system? Would this increase potential sales, orders, or basic value to your end users? If this is the case, then the sky is the limit!

Take the following example: Suppose each user transaction netted $100. If your system was able to facilitate 500 transactions per day, then an increase in performance of 75 percent (using Figure 1-2 as the example) would increase the transactions from 500 to 875 transactions a day. This would equate to an additional $37,500. However, to achieve that additional 75 percent of transactions, what does it cost you in additional resources (such as staff hours and staff numbers) and overhead (such as additional time spent analyzing performance logs and models)?

This is the dilemma you face in trying to achieve application and system optimization: How much is too much?

One of the key messages I try to convey in this book is that performance management isn't simply about turbo-charging your environment (applications, systems, or platforms); it's about smart performance management. Sometimes ten seconds is acceptable for a response time for customers and end users; it may cost a further 10 percent of your operations budget to get to six-second response times, but it may cost 90 percent of your operations budget to get to five-second response times.

Therefore, performance management is about understanding the requirements of performance as well as understanding when enough performance is enough.

NOTE This question of "When is enough, enough?" is somewhat analogous to disaster recovery realization analysis. For example, if a company is considering disaster recovery (in its true sense) as a part of its operational effectiveness and availability program, there's little value involved when a particular application or system is used once a month and takes only several hours to rebuild from scratch. The cost associated with utilizing a disaster recovery site may cost less than $1 million, but the financial impact of a "disastrous event" occurring would only be $25,000. In this case, there's little reason to implement disaster recovery planning. In summary, the cost of trying to tune your platform to provide an additional second of response time costs more than the value that the additional performance provides or creates.

Getting back to the earlier thread, not looking at the WebSphere or application platform in order to scale and increase performance opens you up to myriad potential disasters down the track.

Another problem with this kind of de facto upgrade approach is that your developers may fall into the dangerous trap of expecting unlimited processing power if they're used to working in an environment where simply coding to functional specification and practicing for code optimization are the norm.

NOTE That said, if you've already read later chapters or have previously conducted an optimization and tuning review of your WebSphere environment, upgrading hardware to achieve scalability or boosting performance could very well be the right choice. Just remember, any upgrade to CPU and memory will require changes to your WebSphere application server configuration. As soon as you add either of these two components into a system, the overall system load characteristics will change.

As I've hinted at earlier, negative system and application performance doesn't have just a negative effect on your customers or users, it also has a negative effect on the budgets in your IT department and your manager's perception of your effectiveness.

The bottom line is that poorly performing applications are expensive. The number-one rule in operational effectiveness for managing performance is all about proactive management of your system resources.

Many of you reading this book will be able to tune a Unix or a Microsoft server, but the purpose of you purchasing this book is you wanting (or needing) to be able to also tune, optimize, and ultimately scale your WebSphere environment.

Now that you have a synchronized view of what performance really is, you'll look at the art of managing performance.

Managing Performance

So, what is performance management? The contextual way to answer this is to ask yourself, "Do I have, or have I had, performance problems with an application?" If the answer is "yes," then performance management is the discipline that would've (potentially) mitigated those performance issues.

Depending on whether you're facing this question proactively or reactively, essentially performance management is about the implementation of a performance methodology. It can be further explained as being the execution of the methodology to help identify performance problems (or hotspots); identify an optimization strategy; plan, tune and execute the optimization strategy itself; and, finally, analyze and monitor the system.

There are many industry-specific and even WebSphere-specific performance management models and methodologies. Depending on how fine-grained you like your models, many of these non-WebSphere specific models will suit the task accordingly—one performance model fits all.

As depicted in Figure 1-3, once an optimization and performance management strategy is in place, the performance management model follows an ongoing, proactive life cycle.

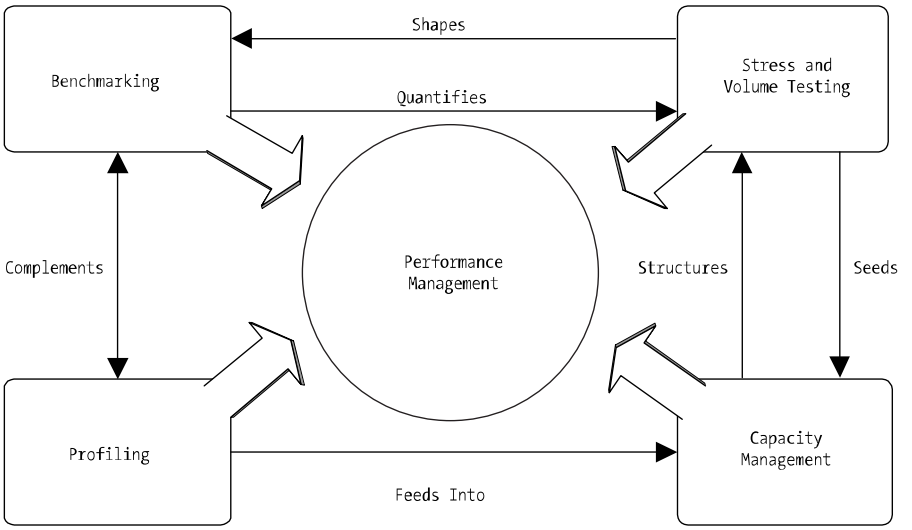

Figure 1-3. Performance management model

Many times system managers implement processes to monitor or tune an environment but don't continue to review, follow up, and make changes where required. A system's characteristics are always changing and growing. Additional customer load, new functionality, and other non-application-specific factors will continuously change the face of your application environment.

It's therefore critical that the process outlined in Figure 1-3 is used continuously to ensure that an ever-changing system is managed for performance.

To highlight this point, Figure 1-4 shows the cost associated with resolving performance problems, both immediately and in the future. As depicted, the longer a performance issue goes unmanaged or unnoticed, the cost associated with resolving or mitigating that performance problem increases drastically.

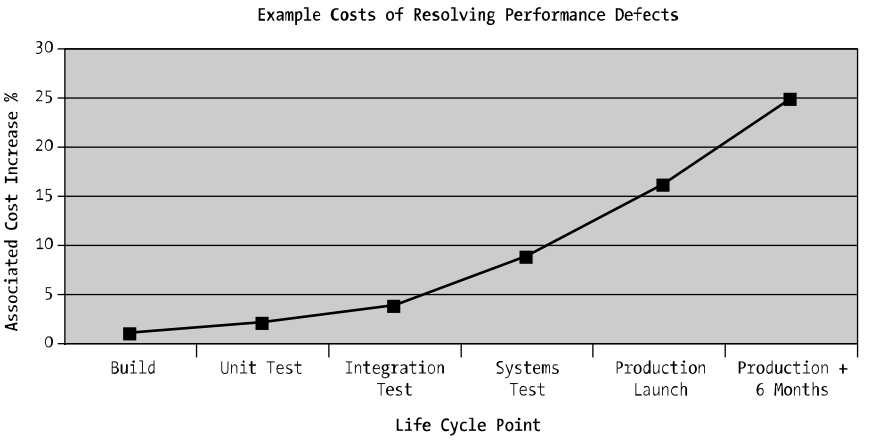

Figure 1-4. Costs for mitigating performance issues, immediate and future

The increased costs come from a number of reasons; the most compelling is associated with the overall life cycle of the platform. That is, if a performance problem becomes apparent yet is left unnoticed or unmanaged, additional layers of application content and functionality will begin to hide the issue. And the longer it takes for you to find and resolve the issue, the more application code you'll probably need to change. Furthermore, the longer it goes unresolved, the more end users, physical systems, and platform complexity that it'll impact.

NOTE The issue of performance problems going unnoticed over time and the costs associated with resolving them in the future is similar to the timings associated with stress and volume testing during application development. The later in your software development life cycle you perform stress and volume testing, the more costly it'll be to uncover and resolve potential problems. For example, if you perform stress and volume testing after application integration testing and find performance problems, then the cost associated with fixing the issue is significant, compared with the cost in fixing the problem at the unit testing stage.

You'll now look at what's considered a valid performance improvement.

What Constitutes a Performance Improvement?

This is one of those questions that has been argued since the dawn of computing! You can view a performance improvement and what constitutes one in many ways. The following are some of the key indicators that can be attributed to or deemed as performance improvements within a WebSphere implementation:

- A decrease in perceived user or customer application response time (in other words, a decrease in wait time between transactions).
- The ability for the "system" to handle more users or customers (in other words, volume increase).
- A decrease in operational costs associated with either or both of the previous points.
- An ability to scale down infrastructure (or consolidate) because of the higher performance of the application on a lesser-powerful infrastructure.
- The bottom line if Business-to-Business (B2B) activity in any given period will have a direct effect on the revenue-generating capability of the application or system (this typically is connected with the second point).

Although this isn't an exhaustive list, it provides an overview of some of the more obvious and tangible benefits that can be derived from a tuning and optimization strategy.

You'll now explore each of these points in a little more detail.

Decrease in Perceived User or Customer Response Time

A decrease in the perceived user or customer response time is the most obvious of performance improvements. As a system and application management expert, you're exposed to all *those* calls from users or customers who are complaining of poor or substandard application response times.

Although on many occasions this can be because of problems outside the bounds of your control (for example, the many hops and intermediate service provider pops between you and the remote Internet site), a fair degree of an application's performance *is* within your control. Some key components within WebSphere that directly affect performance (and are tunable) are as follows:

- **Queues**: Web server, Web container, database datasource, and Enterprise JavaBean (EJB) container
- **Threads**: The availability and utilization of them
- **Database**: Java Database Connectivity (JDBC) pool, Structured Query Language (SQL) statement controls, and the database itself
- **Transaction management**: The depth of your transaction containment and so on
- **The Java Virtual Machine (JVM)**: Garbage collection, utilization of Java objects within the JVM, and general JVM properties

I'll discuss each of these components—and more—during their respective chapters later in the book. However, these five areas are where you'll focus a fair degree of effort for performance management. Proper tuning and capacity management of these aspects of WebSphere can literally make or break a system's performance, and a little tuning here and a little tuning there of things such as JVM settings and transaction management can increase or decrease performance by more than 100 percent in no time.

Combine this with how well your platform and applications utilize the database pooled connections, Java objects, and other areas of WebSphere such as data and session persistence—and then throw in a well-thought-through methodology—and you can achieve a more than 200-percent improvement in application response time on a poorly performing system.

NOTE I recently was involved with some performance tuning work associated with a WebSphere 4.02 platform that presented customer invoices online. The response time goal for the application was to have 95 percent of all transactions completed—in other words, backend transactions complete, Hypertext Transfer Protocol (HTTP) transfer to customer commenced—in less than five seconds. Initial testing was showing average response times of 17 seconds, which was a concern. I'll use this example as a case study later in the book; however, after incorporating a performance methodology into the analysis effort and tuning the environment based on this methodology, the response times came down to between four and six seconds per transaction, which was deemed acceptable. The moral is that the sooner you can implement a performance methodology, the faster and more effectively you'll be able to resolve performance problems.

Performance management is a time-consuming process; however, its rewards are high. Given this, it justifies working closely with developers to ensure they're writing code that not only functions according to business requirements but also is written with an operations architecture best-practice mindset.

In summary, time well spent brings large rewards: Do the research, do the analysis, and spend the effort. Follow these three mottos, and you'll see positive returns.

Ability for the System to Handle More Users

When a system manager is propositioned to scale up his or her platform to cope with "y" number of additional users, there are two schools of thought people tend to follow; one is mostly wrong, and one is mostly right.

At first thought, being given a requirement to scale a system up "x" fold to accept more customers is typically met with additional processors, memory, and or physical systems. It's the old vertical versus horizontal scaling question. Where this school of thought is mostly wrong (and I'll explain why it's mostly wrong shortly) is that, with the right tuning and optimization approach, you can substantially increase the life of a system without an upgrade (within fair boundaries).

Before you balk and say that's crazy, I need to point out this only works for reasonably sized increments of load. What constitutes reasonable is driven by the nature and characteristics of the application. I've seen a system architecture implemented for a baseline set of users and the call being made to be able to ensure that it'll cope with nearly three times the original load. Tuning and optimization of the platform made this reality.

That said, with the decreasing costs of hardware from IT server vendors, it may in fact be less expensive to purchase more memory or additional Host Bus Adaptors (HBAs) than to perform a thorough review on the application or environment. This can definitely be the case for the midsize upgrades; however, keep in mind that midsize upgrades come in pairs and trios. Therefore, no doubt in the near future, your stakeholders will approach you requiring additional increases in customer load for another functional or capability change. If you get the impression that you can just keep squeezing more and more out of your system by simply purchasing additional hardware, you'll end up painting yourself into a corner.

What I'm trying to say is this: Hardware upgrades aren't always required for additional load of increased performance. In fact, for two out of three situations, hardware upgrades won't be required for additional load or capability enhancements.

Figure 1-5 breaks down the associated costs for an example platform where an increase in hardware has increased support and maintenance costs to provide a two-fold increase in processing capacity.

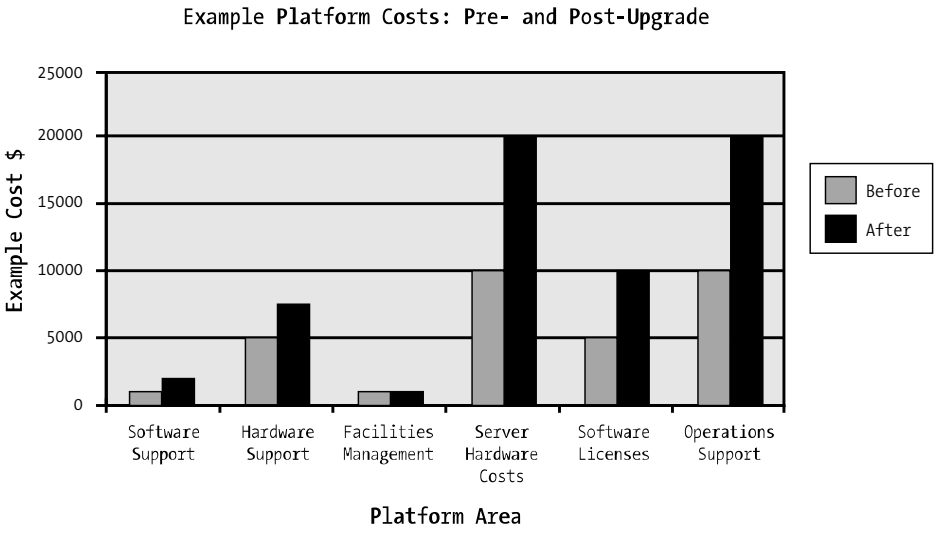

Figure 1-5. Pre- and post-upgrade support costs for an example platform

As depicted in Figure 1-5, the increase in hardware has increased the overall support, maintenance, and license costs for this platform. Because the WebSphere environment hasn't been optimized or hasn't been optimized correctly, the "bang for buck" in dollars per additional transaction is lower.

Figure 1-6, however, shows what can be done in the opposite situation. If a system is optimized correctly, the cost benefit ratio is high. Both these figures are fictitious but do represent the benefits associated with JVM tuning and proper configuration.

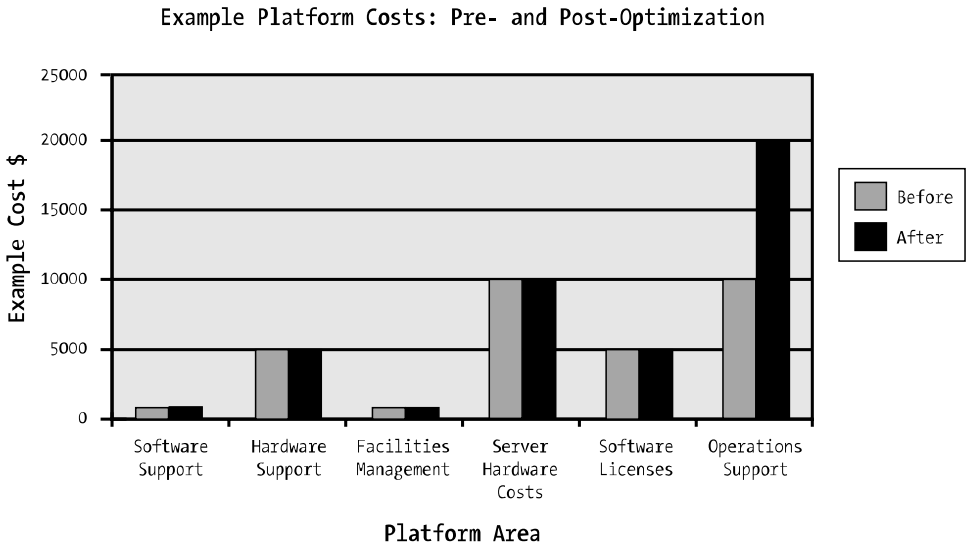

Figure 1-6. Pre- and post-upgrade support costs for an example platform, the optimized approach

As shown in Figure 1-6, the dollar cost per transaction greatly decreases with optimization, and at the same the time, the overall operational costs associated with the platform remain unchanged.

It's important to understand that the same can be said for the opposite. If you get the impression that "a tweak here and a twist there" can give your business customer a 500-percent increase of application load or performance—without hardware increases—then this can also paint you into a corner.

Setting expectations and providing a clear outline of your methodology and process for the ongoing life cycle of the system, and its application constituents, is critical to your role as an application or system manager.

Decrease in Operational Costs

Operational costs associated with poor or low application performance are one of those financial burdens that, over time, add up and can suck an IT budget dry. Operational costs are typically associated with management and support, the system's "food and water," and ancillary items such as backup tapes, software concurrency, and the like.

A low-performing application that requires heightened management and support because of increased customer inquiries and support calls, additional system "baby sitting," a reduced ability to perform support tasks during operational hours, and so forth will incur costs at a fairly rapid rate. It can be a "slippery slope" under these conditions, and only through proper risk and program management does a system or application in such poor shape survive.

Figure 1-7 indicates the costs involved in an operational budget for a medium-sized WebSphere environment operating on a four-CPU Unix platform with dual WebSphere application servers.

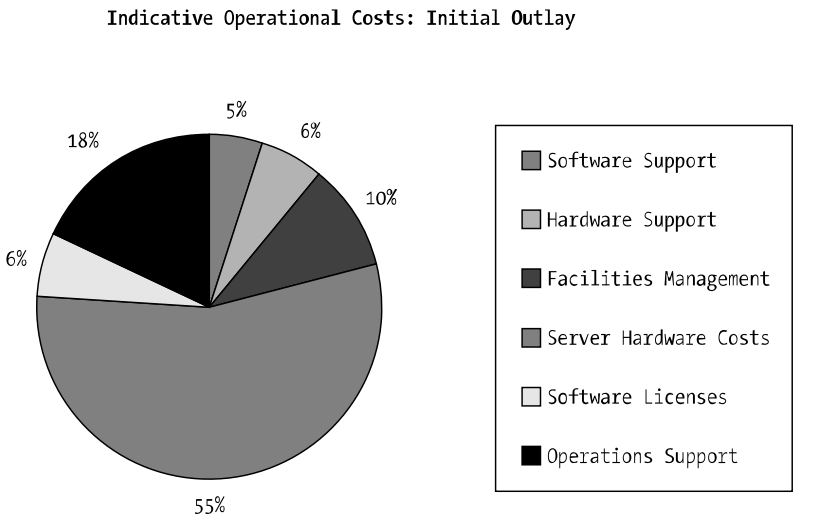

Figure 1-7. Indicative operational costs, initial outlay

The largest of all cost components in Figure 1-7 is the infrastructure costs, followed by license costs. This breakdown assumes a zero-year horizon on the cost outlays (in other words, upfront cost prior to development).

If you look one year into the future, the most significant costs are the maintenance, support, and licensing costs. Maintenance and support as well as ongoing licensing costs are usually driven by the size and hardware platform you're using. Most vendors work on a license and maintenance agreement where the number of CPUs used in your environment drives the total cost.

You can probably see where this is heading—if you have a well-optimized environment with less hardware, your maintenance costs are reduced and so are your license costs.

In summary, a high-performing application will require less support and baby sitting. Operational tasks can be completed during standard operational hours, and support calls regarding performance and slow response times will be

minimized, which reduces the head count for support staff, which in turn directly affects operational support costs.

Ability to Scale Down Infrastructure with an Optimized WebSphere Environment

A system's "food and water"—power, cooling, and hardware maintenance—can also be reduced through WebSphere application optimization and tuning. If fewer physical systems are required, food and water costs are reduced.

This is an important factor in obtaining funding for performance optimization and tuning tasks. As you'll see in the next section, sometimes selling the concept of an optimization and tuning program to business stakeholders (those who hold the strings to the money bag!) is difficult. Unless you reduce the true bottom line costs and can prove that through a structured and clear methodology, these stakeholders will not lay out the funds.

By clearly showing you can reduce operational expenditure costs (by reducing the number of servers, which means less support and management costs), you can easily sell the operational and application optimization and tuning efforts.

The Business Bottom Line

Although some of these points about a businesses bottom line and cost analysis are all somewhat "motherhood and apple pie," the number of installations and environments that can't or won't see the benefits of tuning and optimization is amazing.

Technical people will typically see the benefits from a better-performing system and application. Business people need to see the advantages from a bottom line perspective, and unless you can sell these conclusively, then your optimization plan—or more precisely, its benefits—needs more work.

In some situations, if an application has a transaction response time of ten seconds and the business representatives and customers are happy, do you need to optimize the system further? Unless optimizing the platform will provide a drastic reduction in infrastructure (and all associated costs), then "if it ain't broke, don't fix it."

NOTE I'm sure we've all faced the opposite of this, which is where a stakeholder or business representative demands unachievable performance requirements such as ensuring that 95 percent of all transactions to and from an IBM OS/390 mainframe complete in less than one second!

Essentially, one of the key aims of this book—other than *how to optimize*—is to attempt to show the business benefits of optimizing and tuning WebSphere. Many readers may already be at the point of having identified the benefits of optimization and even having the concept sold to business sponsors. Possibly, you may now need to know how and what to do to execute the optimization and tuning efforts.

Either way, by reading and using this book, you should be able to build and complete your WebSphere optimization and tuning strategy.

Measuring Business Improvements

It's all well and good to understand the obvious technical benefits of optimizing and tuning, but you need to understand the impact, both positive and negative, that this effort entails. You may be able to tune the JVM settings and notice a difference in the performance of the application or overall environment, but what about the tangible benefits to a business? As discussed, business sponsors and stakeholders will want to see benefits from a business perspective. That is, they'll want to see a decrease in operational, support, licensing, infrastructure, and other costs or an increase in customer usage, satisfaction, and, ultimately, transaction rate (which may or may not drive revenue, depending on the type of application you're tuning).

I'll discuss this topic more, in context, throughout the rest of this book; however, it's worth covering some of the key business impacts that are measurable from optimization and tuning efforts.

The following two scenarios show how business benefits are measurable with performance optimization and tuning:

- Scenario 1 looks at a fairly new application that's in pilot mode. Problems have been found with the application's performance, and failure to improve the performance will most likely result in the application being cancelled.

- Scenario 2 discusses the cost benefits of optimization for a platform that's requiring an upgrade in end user capacity. It explores the alternatives and highlights the bottom line cost benefit from optimization.

Scenario 1: A Pilot Application

As you've seen, the outcome of all optimization efforts for a business is an impact to the bottom line (hopefully a positive one). If WebSphere optimization and tuning is conducted properly, you'll see a positive effect on the budget sheets.

The positive impact may not be so clear initially; however, over time the benefit realization will accelerate and become more obvious. An example of this would be an optimization and tuning effort conducted while an application is new or in pilot phase.

A pilot phase typically includes a small handful of test or trial users testing an application or system. Some people call this a *live production shakeout,* but effectively it represents a pilot phase of the application development life cycle. If the pilot phase, making use of only 10 percent of expected total number of users, showed that the response times were hovering around the 90-percent to 95-percent mark of the acceptable Service Level Agreement (SLA), this would provide an obvious case for optimization and tuning efforts.

For example, let's say an acceptable response time for a WebSphere-based application was five seconds per transaction, and during the pilot, with 10 percent of the targeted application load, pilot users were reporting four-second response times.

Now, unless the application operating in WebSphere had externally controlled constraints that meant four seconds would be the norm even for up to 90 percent of the total projected end user load, typical capacity modeling would suggest that transaction response times increase as the user load does. Given that norm, if 90 percent of the acceptable response time was being reached with 10 percent of the projected end user load, then there's a problem!

NOTE Of course, stress and volume testing should have picked this up in preproduction deployment; however, this doesn't always happen. It's amazing how often stress and volume testing isn't conducted or isn't conducted properly. Never go live (or permit an application to go live) into production without proper stress and volume testing.

At this point, there would be several scenarios that may play out.

If the platform had been sized and modeled by the capacity planners and system architects (typically those reading this book) according to the functional and technical specifications, then this would serve as a baseline for operational performance. However, the input to the platform and application performance modeling is just as critical as the output of the model itself.

In this situation, given that the system performance characteristics are broken, proposing an optimization and tuning exercise of the WebSphere application environment is the only sensible course of action.

Let's assume that some stress and volume testing or profiling had taken place once the performance problem had been realized (I'll go through profiling in later chapters of the book). The output of the profiling would identify poorly operating application code and areas within WebSphere that could be tuned to extend the life of the application in its current state.

The alternative would be to purchase additional hardware and, based on the previous example, lots of it. Mind you, as discussed earlier, it may be a dual-pronged plan—a combined hardware and optimization change.

Measuring the business impact of either solution is straightforward. The baseline cost would be initial implementation. Additional effort to analyze and ensure the application operates within the guidelines set out in the SLA would require funding. Further, depending on the outcome of the analysis, additional hardware or funding for an optimization and tuning effort would be required.

The costs associated with the Scenario 1 project are as follows:

- Initial hardware purchase: $250,000
- Initial development costs: $300,000
- Projected ongoing costs: $50,000 per annum

Let's look at the two options available to this problem: a hardware upgrade or optimization effort. You can assume that to achieve the SLA required with an additional 90-percent load, you'll need at least a 100-percent increase in additional infrastructure or a 100-percent increase in platform optimization (which effectively means a 100-percent improvement in performance capacity).

The following are the costs for Option 1, a hardware upgrade:

```
Additional projected ongoing costs:     $175,000 per annum
    (additional support for
    operations and hardware)
Additional hardware required:           $200,000
____________________                    _________
TOTAL                                   $375,000 for year one
```

The following are the costs for Option 2, an optimization and tuning effort with minimal hardware upgrade:

```
Projected ongoing costs:                    $75,000 per annum
    (additional $25,000 per annum
    for increased ongoing
    optimization and tuning program)
Projected costs for tuning effort:          $25,000
Projected costs for code changes:           $50,000
Additional hardware required:               $75,000
____________________                        _________
TOTAL                                       $225,000 for year one
```

Figure 1-8 depicts these two options side by side.

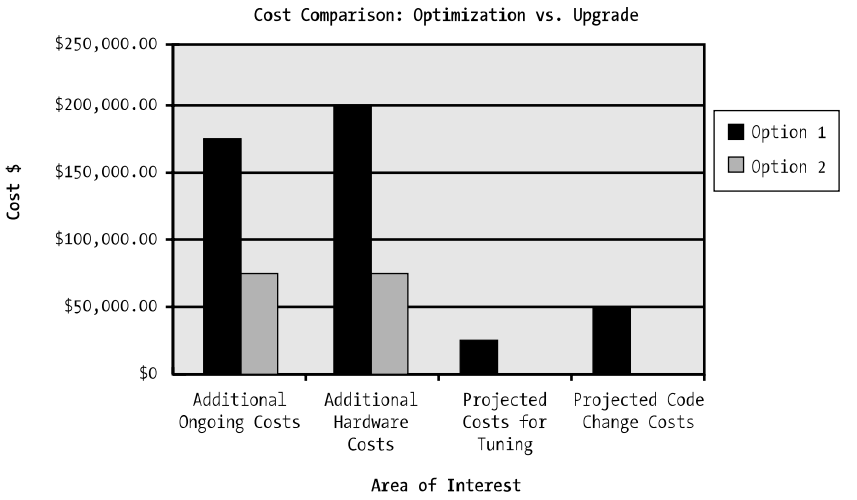

Figure 1-8. Comparing costs of both options

The two options side by side are fairly self-explanatory. Although these costs are examples, they do show the differences in associated costs for a scenario such as this. Specifically, a 100-percent improvement in performance through optimization is in most cases achievable. Even 200 percent isn't a difficult task.

Essentially, the business benefits of improving performance through an optimization and tuning program far outweigh the benefits or costs associated with an upgrade in hardware. The business benefit, although driven by a reactive requirement (in other words, post-deployment cleanup), is $150,000.

Figure 1-9 depicts a second-year view on this scenario. This figure indicates the ongoing savings associated with the initial outlay. The key factor that's driving the increase in cost savings is the ongoing cost for hardware support.

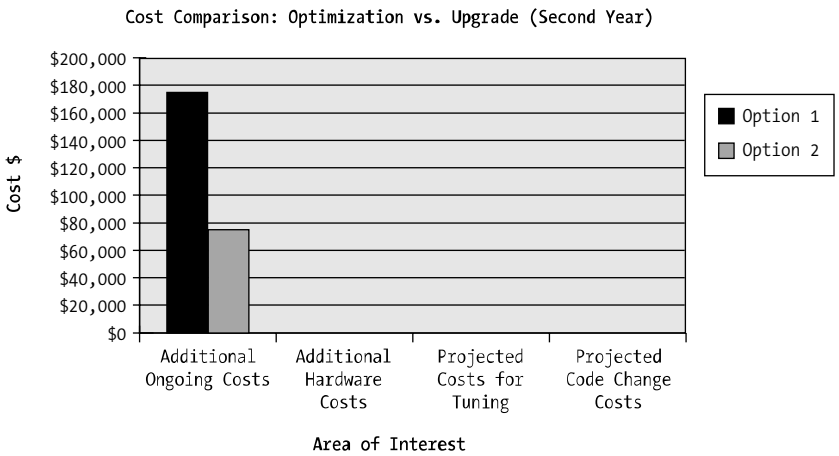

Figure 1-9. The second-year cost comparison of both options

Scenario 2: A Request for Additional User Capacity

Scenario 2 looks at a platform with an e-commerce application, one that has been operating within a WebSphere environment for some time.

In this scenario, if a requirement comes from business sponsors or the company's marketing group that indicates additional users are being targeted through a marketing campaign, you'll need to investigate the state of the WebSphere and application platform to determine the impact and whether any action is required.

As previously mentioned in Scenario 1, two clear options exist in this type of situation. You can undertake a straight hardware increase, based on capacity modeling, or you can facilitate an effort to investigate and ultimately optimize and tune the WebSphere platform.

Typically, you can undertake a baseline of an optimization and tuning strategy quickly and with fewer overhead and impact than a production hardware upgrade. If an optimization and tuning methodology is in place and performance management is a key part of this particular scenario's operations architecture, undertaking a review of the WebSphere platform's performance will incur a minimal incremental cost. Even optimization and tuning that doesn't involve developers can significantly increase the performance of a WebSphere implementation. Modifying JVM settings, transaction management controls, queues, and similar aspects of the environment can result in a large result-to-effort ratio.

In this situation, activating a series of performance monitors that capture the characteristics and workload of the WebSphere environment will provide the key indicators for a review. Once metrics have been captured, then analysis and review can take place. This typically takes one to five days, depending on system complexity and size (at least highlighting several of the key areas of the platform).

Next, planning the WebSphere optimization and tuning effort based on the analysis obtained from the metrics provided by the performance monitors and then planning, testing, and implementing the changes typically requires an additional five to ten days for a midsize system.

All in all, this effort takes approximately ten days; assuming a system manager rate of $500 per day, the outlay would be $5,000.

Again, let's assume an increase in performance of 100 percent was obtained—for this example, let's say the analysis found that several Java database connection methods were serialized, causing all database transactions to queue up behind one another. This is a common problem I've seen in smaller environments where connection pooling within WebSphere wasn't used and a custom-written pool or database connection manager was employed.

Without proper analysis of what was happening "under the covers" using a sound performance methodology, a hardware upgrade would've been costly. A hardware upgrade may not have even resolved the issue, given that the example states the problem is a serialization of the connection beans to the database.

One may have looked at running additional WebSphere instances to allow more concurrent database connections (meaning a higher volume of serialization and bank of database requests but still a higher throughput of queries). This would require additional CPU and memory on the WebSphere application server to provide capacity for running multiple WebSphere application server instances.

Additional hardware costs mean additional licenses and support. Figure 1-8 highlights the difference between the optimization approach and the hardware approach—both options would achieve the same outcome. The hardware costs detailed in Figure 1-8 are based on public pricing details available on the Internet (to be nonbias, I've averaged the cost per CPU and memory between several Unix server vendors, based on entry-level servers). Again, please keep in mind these costs are examples only and are an attempt to show the business improvements and the measurement of these improvements over time; however, their impact and realization is accurate.

Therefore, in summary, measuring business improvement or the effect on a business's bottom line when tuning and optimizing is simply about weighing these tangible costs. I don't want to imply that to gain performance the only option is to optimize and tune and not to perform hardware upgrades. This isn't the case; in many situations—and I'll go through more of these later in the book—a combined approach of hardware and optimization is the most logical.

TCO and ROI Equal TCI: Total Cost of Investment

Now that you've seen the outcome of an optimization and tuning effort, you'll take a step back and look at what other forms of analysis models are available to determine the best alternative for your program.

Total Cost of Ownership (TCO) and Return on Investment (ROI) are among some of the most overly used acronyms within the industry today. Both are high-level, analytical models that provide the unwary system manager with an "all bases covered" methodology hype. Vendors have been flogging their wares for more than a decade now using TCO and ROI statements to quantify the value of their product.

With that said, it's not that either of these two models can't provide value. In fact, used correctly and in the correct context, they're both powerful, independently or together.

Let's take a closer look at both models.

Total Cost of Ownership: An Overview

Since its inception back in 1986 by an analyst at Gartner Research, TCO has been increasingly used as *the* benchmark for determining the overall, medium to long term costs associated with a particular piece of technology. Whether it's software, desktops PCs, servers, or an entire IT project, TCO has been used as the comparative vehicle.

Although TCO has some flaws in its methodology (which you'll see shortly) when used to compare apples to apples, it can provide a well-balanced view of the total cost associated with some form of IT technology investment. This may be a new hardware purchase, a support arrangement, a technology or architectural selection, or an optimization strategy.

By indicating some flaws with TCO, this alludes to the TCO model being an open one. There are little bounds to it, and the outcome is only as good, as valid, or as comprehensive as the input and depth of comparative data used as the basis for the analysis. TCO is also typically only used to model something on a timeframe longer than the implementation itself.

Another example of where TCO breaks is in an example of conducting TCO on the purchase of office desktop PCs. If a TCO analysis on two PC vendors takes into account the cost of the PCs, software, support and installation, and power, how does TCO perform if one vendor's PCs are lower quality yet cheaper? Yes, the TCO based on the purchase price, software, installation, and power is potentially lower, but unless hidden costs can be included into the TCO equation, then it's of little value.

To use a WebSphere-based example, if you attempt to perform a TCO analysis on a WebSphere application platform and attempt to model the TCO based on two differing optimization methodologies, your input must be comprehensive. The inputs should include not only the time and effort to build and design the optimization strategy but also the costs associated with potential changes to hardware, support processes, operational processes, and so forth.

However, if you again use this and change the perspective of time to model two proposed architectures for a new WebSphere installation, you'd include cost items such as these:

- Hardware (physical servers and disk arrays)
- Infrastructure software (initial costs for operation systems and other middleware)
- License costs (WebSphere, operating system, database)
- Support costs (ongoing vendor support)
- Deployment costs (operational staff time and effort)
- Water and feed (power, cooling, fire retardation)

These factors start to provide a good basis for a correct TCO model during implementation.

Another important factor is what I term as *soft costs*. These are TCO aspects such as potential cost reductions through support staff options (for example, lots of lesser-experienced personnel or fewer experienced personnel), maturity of vendors (for example, response times to support calls and professionalism), and availability assurances (from vendors). Most often, these elements don't carry a direct dollar value. Instead, they're used as weighting metrics, and I'll explain this in more detail shortly.

I'll now put all of this into some context.

Tables 1-1 and 1-2 compare two basic systems. This example attempts to model the TCO of a WebSphere implementation to determine the optimal system in terms of performance and scalability.

Table 1-1. TCO Matrix, Hard Metric: Implementation Option 1

TCO Element	WebSphere Implementation Option 1, First-Year Costs
Hardware costs	Two × Intel Pentium 4 servers = $30,000
Infrastructure software	Operating systems and Enterprise Application Integration (EAI) middleware = $15,000
License costs	WebSphere, Oracle, and operating system = $10,000
Support costs	Hardware and software support = $15,000
Operational support costs	24×7 support = $50,000
Deployment costs	Integration team × four personnel = $20,000
Water and feed	Power, air conditioning, and fire-retardant system = $5,500
TOTAL First Year	**$145,500**

Table 1-2. Example TCO Matrix, Hard Metric: Implementation Option 2

TCO Element	WebSphere Implementation Option 2, First-Year Costs
Hardware costs	Three × Intel Pentium 4 servers = $45,000
Infrastructure software	Operating systems and EAI middleware = $20,000
License costs	WebSphere, Oracle, and operating system = $15,000
Support costs	Hardware and software support = $25,000
Operational support costs	24×7 support = $50,000

Table 1-2. Example TCO Matrix, Hard Metric: Implementation Option 2 (Continued)

TCO Element	WebSphere Implementation Option 2, First-Year Costs
Deployment costs	Integration team × two experienced personnel = $20,000
Water and feed	Power, air conditioning, and fire-retardant system = $8,500
TOTAL First Year	**$183,500**

As you can see from Table 1-1 and Table 1-2, Option 1 for the first year is the cheaper alternative—or, it has a lower cost of ownership. If you added scalability as a soft metric and worked on a 75-percent usage growth per annum, the outcome would look different. For example, if these examples were specified to handle 100 concurrent users, the Option 1 (Table 1-1) servers may each be operating at 50-percent load and Option 2 (Table 1-2) at 35-percent load.

If the annual plan to increase concurrent users serviced by the application was an additional 75 percent (an additional 75 users), the Option 1 servers would be nearing 90-percent utilization, or saturation point. The Option 2 servers would be operating at approximately 60-percent load each (getting high but reasonable). Option 1 therefore would require a doubling of infrastructure in the second year, as well as additional software and licensing to service the additional servers.

Figure 1-10 compares the costs of these options for two years.

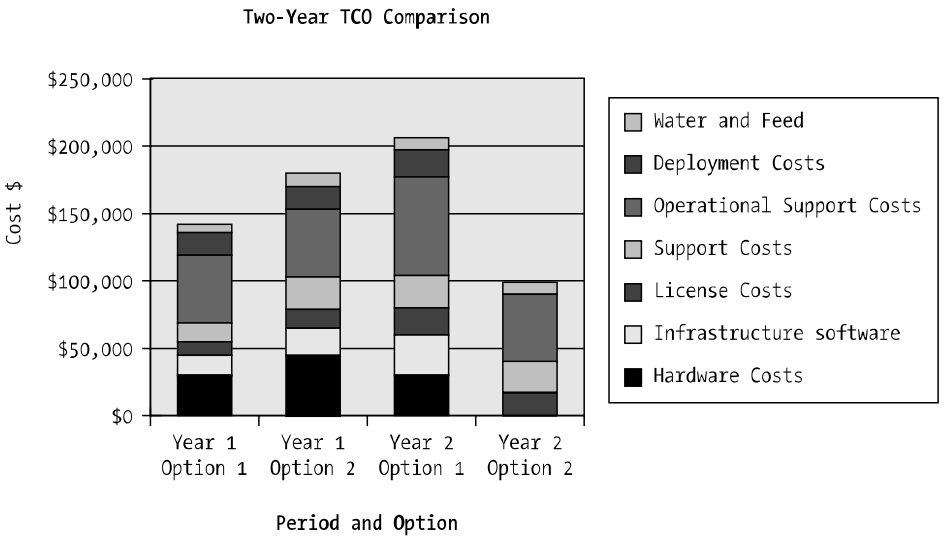

Figure 1-10. A two-year TCO comparison of both options

Therefore, as depicted in Figure 1-10, a two-year view on the TCO, based on the inputs from Tables 1-1 and 1-2, would show that Option 1 would become more expensive over time than Option 2.

The reason is this: Option 2 doesn't need an upgrade in the second year. The costs are fixed and known. When an upgrade is required, this would incur a deployment cost (see Tables 1-1 and 1-2 for associated costs) for each upgrade. To then include soft metrics such as the impact to availability and scalability of the solution, Option 2 starts to pull away from Option 1, boasting an increasingly better TCO. Obviously, three servers are more scalable—albeit to a point—than two servers, and the additional server provides a higher level of availability.

Although this may not accurately represent a real-world scenario, the concept is sound. Projects and people typically underestimate the hidden costs such as development and training when formulating TCO.

This section showed that when performing analysis on an optimization and tuning plan to determine the most cost-efficient or cost-rewarding direction, you should consider the context in which you model the TCO. Use apples-to-apples comparisons, and don't be afraid to use complexity or the perceived value of something (for example, vendor track record) as a consideration.

Now that you're starting to learn about cost rewards or cost efficiencies, you're starting to get closer to the ROI model.

Return on Investment: An Overview

An ROI of a positive of negative nature is an important aspect to be aware of with any system or application implementation. Business cases tend to use a number of derived ROI models such as Net Present Value (NPV).

ROI is best used to predict or model the *future* cost efficiency of an investment, such as the deployment of a WebSphere-based application on a commodity Intel platform versus a higher-grade environment such as a Sun UltraSPARC III platform.

ROI differs from TCO in that TCO was designed to model the outright total cost of something (such as a total WebSphere platform), and ROI was designed to model the financial benefit or return of something in the future.

For example, you may use an ROI model to determine the financial benefits and general return (a.k.a. bang for buck) between two WebSphere optimization strategies—one being to simply install more hardware, such as CPUs and memory, and the other being to physically tune the WebSphere application server engine.

Let's focus on that example for a moment.

Using the set of metrics discussed in the previous section, let's use a scenario where the Chief Information Officer (CIO) has requested an audit of the systems and applications that run within your WebSphere environment. The audit is to perform a review of the cost per transaction for the applications

operating within the WebSphere application environment for which you're responsible. The cost for each transaction is made up of system cycles (CPU power), processing capacity of the application server, and time taken to complete the transaction.

The audit is completed, and the results are that the cost per transaction is too expensive—$10 per transaction!

Your next step is then to look at optimizing your WebSphere application environment (using this book, of course). After planning an executing a performance and optimization strategy, you perform an ROI analysis.

In this example, if it took the following to optimize your WebSphere environment, the cost for strategy compilation, execution, and reporting, the program would cost $20,000:

- Ten days to build and test the performance and optimization strategy at $500 per day (you and your team's time)
- Five days to test the strategy in an integration/stress and volume environment at $500 per day
- Two days to implement the solution
- Three days to compile the performance monitoring statistics post-solution deployment

The resulting cost per transaction, after a successful reduction of processing overhead within the WebSphere application environment, was $2.50 (or 75 percent). Therefore, assuming 100,000 transactions per month, you can say, simplistically speaking, the ROI of this effort of $20,000 is 97 percent. That is, for an investment of $20,000, the successful optimization effort produced a monthly positive return (saving) of 97 percent from the pre-optimization platform performance characteristic.

Measuring this type of benefit is important. System managers and architects need to be able to report and represent, as well as view and identify, the bottom line return that will be achieved by investing time and effort into a performance methodology and upgrading hardware.

Managing Performance: A Proven Methodology

Although you can use many approaches to address performance and optimization management with WebSphere, I've found one that tends to synchronize well with the WebSphere platform. It also has the added benefit of being relatively straightforward.

In essence, it's a hybrid of a number of old-style methodologies grouped with newer schools of thought and best practices. For most environments, it'll work well; however, be careful to validate and confirm the approach before diving in. Attempting to overlay complex methodologies with something as complex and large as WebSphere is destined for failure. Instead, a straightforward and logical approach is essential. This methodology is also a living one; that is, you should continue to use it proactively during the lifetime of the system.

Before looking at the methodology itself, you should consider some key points for commencing a performance management program. The next sections highlight these.

Performance Management 101

Like in any testing situation, you can break down the basic scope of effort to the aim, the method, the test, and the results. Although it's a discussion about fundamentals, I believe it's important enough to consider in this context.

Considering these four key points, let's outline a brief performance management approach that the performance methodology, discussed shortly, sits on top of.

Problems found in the results typically can be fed back into the aim of another cycle. Using this aspect of performance management, the model should show the process cycle around continuously, being used proactively to identify and help rectify issues.

The Aim of Performance Management

The aim is sometimes not as obvious as you'd initially think. Consider an underperforming system that needs to be fixed. In this case, the aim is to increase performance by a determined factor. That factor is defined by business sponsors, end user demand, legal agreements, and so forth.

Take, for example, a situation where the aim is to resolve an application performance problem, specific to database-related transactions where a certain query is taking five seconds to respond. This query is fundamental to the application's functionality, and its lengthy delay is causing end users to complain.

The aim is therefore to bring the database response times in under an acceptable threshold. Let's assume that the SLA for the full end-to-end user transaction is five seconds, of which in this example the database query itself is consuming the ensure SLA, leaving no overhead for content composition and so on. It can also be said that the aim is the problem you're attempting to solve.

To get the SLA back under its threshold, the system manager has determined that the SQL query must not take any longer than one second. Therefore, the

aim of the performance management approach is to perform analysis on the application environment to bring the database query response time down from five seconds to one second.

The Method of Performance Management

Now that you have the aim, the performance management approach needs to define a method for analysis and testing (quantification). This part of your management approach should address how the analysis will take place and the tools you'll be using.

Continuing on from the example from the aim section, the method of analysis will be to run a SQL query monitor on the database server to determine what's wrong with the query itself as well as to monitor the output from the WebSphere Resource Analyzer to investigate what Java components, if any, are causing the delay.

The method should also include contingency plans for backing out any monitoring components or changes to environments because of an issue in operating the tests.

Testing Performance Management

The test to conduct the analysis will be to run the monitors in the production environment to get the most accurate data set in the least amount of time. This test will operate for 12 hours, during both peak and low utilization periods. The system manager will monitor it continuously to ensure that the monitoring doesn't affect the actual operation of the production system.

The attributed requirements from the aim should be repeatable within the test. This ensures that an averaged result is obtained so that singular uncontrolled events (external issues and so on) don't impact or interfere with the result.

The output will be captured in raw and binary formats—raw from the SQL Query Analyzer and binary from the WebSphere Resource Analyzer tool.

Performance Management Results

The result of the test is the result of the analysis. The result therefore, driven from the output from the test, will be fed into the performance methodology discussed in the next section. Analysis from the test should provide a clear indication of where the problem lies in the environment and will help to provide a guide for where the analysis phase should start.

Now that you've considered this semiformal approach to performance management fundamentals, let's now look at the methodology and walk through how it works, what it covers, and how to implement it.

The Mirrored Waterfall Performance Methodology

Before looking at what the Mirrored Waterfall Performance Methodology (MWPM) is, first you'll see the key areas of a J2EE-based application server environment that affect an application's responsiveness (in other words, its perceived performance) the most:

- Java/JVM memory and object management, queues
- JDBC/Container Managed Persistence (CMP)/Bean Managed Persistence (BMP), pooling, and Java Message Service (JMS)
- The platform components: database, networks, and their configurations
- The physical server configurations (memory, CPU, disk, and so on)
- The operating system: the kernel itself and all associated settings (for example, networks)

I'll go into these five key areas in more detail in future chapters; however, this list is the foundation of what the methodology addresses.

The methodology works using a directional flow rule, somewhat similar to a waterfall (see Figure 1-11). At the top of the waterfall diagram you have the physical server(s) or hardware, followed by the operating system and its associated configurations. The next level down contains the platform components such as databases and network settings, followed by the JDBC, pooling, and JMS type services. Next come the Java and JVM settings and the queues.

Essentially, how this model works is this: If you want to tune the JDBC Connection Pool Manager settings in WebSphere, you must, according to the model, consider tuning and analyzing the configuration and settings of components in the bottom family grouping—the JVM/Java and queue settings. The rule of the model is that if you need to tune or alter something, you must also consider and analyze all component groups down the waterfall. Like water, you can't go back up the model; you must always work down the model on the left side.

The previous example relating to the JDBC settings requiring changes or analysis on the JVM/Java and query component grouping is driven by the fact that altering the JDBC connections will affect on the flow of transactions through the overall WebSphere environment. This is typically associated with what's known as *queues*. I'll discuss these at length in Chapter 9; however, for the

purpose of this chapter, understand that the concept of queues essentially relates to the number of open connections at each tier and how those open connections are managed.

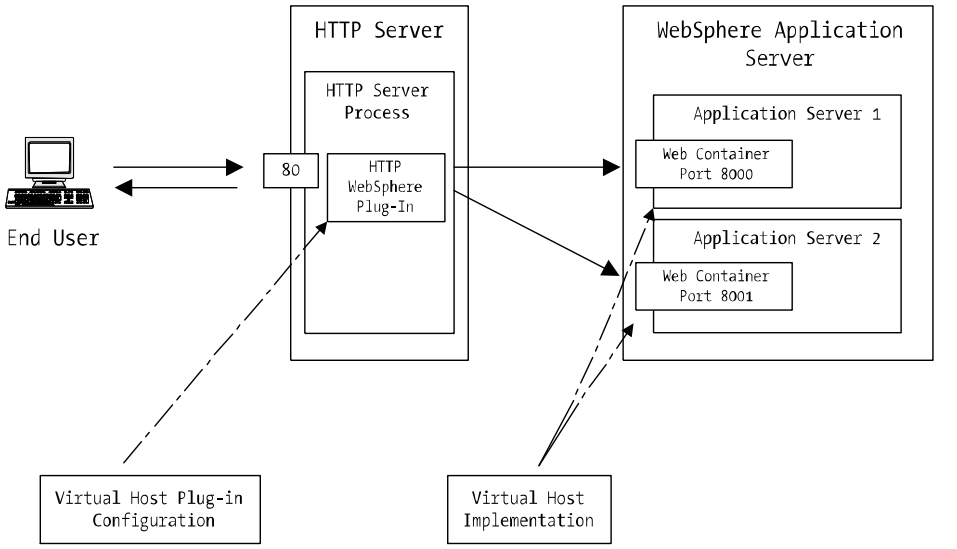

Figure 1-11. The waterfall model

It should be obvious that given the JDBC connections are changing—which will affect either how many or the characteristic of the SQL connections to the database—this will result in a change to the overall balance of the platform tiers.

Secondary to this, because the balance or characteristics of the application platform will change as a result of the JDBC changes, this may change the JVM requirements. How many users (which equates to sessions) will now be required to operate on the platform? Users or sessions drive JVM heap size, so how many concurrent users are on the system, and what type of profile do those customers have?

As you can see, it's all connected.

The second point to make regarding the model depicted in Figure 1-11 is that of the opposite or mirror waterfall model. The right side of the model is the "driver" or mirror aspect. That is, each of the right side component groups drive a change to the next component group up the waterfall, if required.

To put it another way, if after monitoring the HTTP transport queue level in the WebSphere Resource Analyzer it was found that the transport queue was running at 100-percent utilization, then this would therefore mean, based on the model, that the problem with the queues would "drive" the need for a change in the component group one level up the mirror side of the waterfall. In this case, it would drive the need to investigate why the HTTP transport queue was saturated.

Essentially, the right side should only be used as a "finger-pointing" exercise. Use it in the event that something has broken or something is performing badly. It'll help to direct where the source of the problem is.

TIP Remember that if a particular component isn't performing or is broken, it's not going to always be the fault of that component. In the previous example of the HTTP transport queues reaching maximum capacity, the problem or root cause may not be that the HTTP transport queues are set incorrectly, but it may be that the JDBC settings aren't aligned correctly with the entire platform. In these situations, identify the problem, and then identify the root cause using the right side of the waterfall model.

The pitfall in not conducting these "sanity checks" as part of the methodology is that you end up with a turnip-shaped environment. This introduces potential bottlenecks and will most likely cause more headaches than what you had to start!

The shape of your environment, in terms of incoming requests from customers or users, should be carrot shaped, or a long funnel (see Figure 1-12).

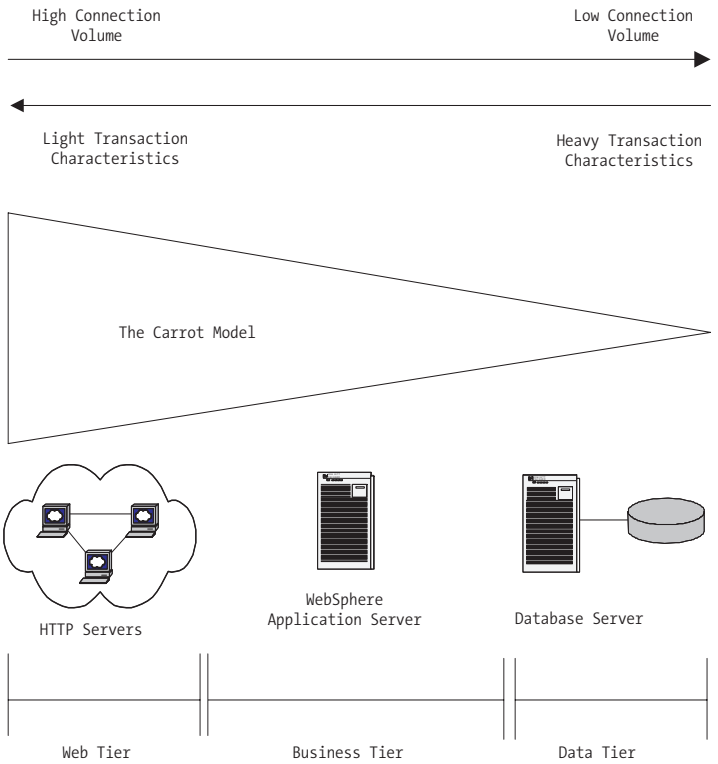

Figure 1-12. Correctly configured environment—the funnel, or carrot-shaped, model

You'll investigate why these environment shapes are important in Chapter 9; however, as a brief overview, you can look at the overall environment from a cost point of view. The pointy end (less volume) of the carrot is the most expensive end, and the larger, less pointy end (high volume) is less expensive.

Most J2EE-based applications will see their environment as having the database tier as the pointy yet more expensive end of the funnel, and the Web tier as the less pointy, less expensive end. Databases transactions are heavy and therefore expensive. Web transactions are lightweight and are somewhat inexpensive.

This boils down to queuing methodologies, a key area of an overall system's performance and one I'll address in detail in future chapters.

Other Considerations

The MWPM (Mirrored Waterfall Performance Methodology) highlighted in Figure 1-11 will be referenced throughout the rest of the book. Therefore, you need to consider a number of other issues with the methodology. Many of these are foundational statements, but many systems managers miss or don't incorporate them formally into their operational models. I'll also cover these in more detail throughout the book; however, they can be summarized as follows:

Educate your developers on the workings of WebSphere—what's a queue, what really is a transaction, what's a small lighter-weight SQL query versus a larger heavyweight query? (See Chapters 9 through 12 for these topics.)

Use this book and best practice development guides to help your developers understand the implications of non-system-friendly code (for example, leaving hash tables open in session state and so on).

Build standards for development, and ensure they're included in the quality assurance and peer review checkpoints.

Implement historic monitoring and reporting. Always ensure you're charting the key components of the system. This should include CPU, JVM, memory, disk, network, and database utilization. I'll discuss some tools to do this later in the book.

Monitor and plan your tuning and optimization approach; in other words, focus on one (problem) aspect of the system at a time. Don't take on too much—conduct the analysis and tuning in small steps.

Implement one change at a time, and monitor. Don't fall into the trap of implementing a whole range of performance tuning changes. Implement one change, monitor for a period of time that gives suitable exposure to system load characteristics (in other words,

don't implement on Friday, monitor on Saturday when no one is using the system, and take that as gospel!), and then analyze. If the implementation was successful, roll out the next change.

The final point you should consider isn't a particularly difficult issue, but it's one that's commonly overlooked: conducting performance management and analysis without the testing or monitoring itself, skewing the results, and affecting the outcome.

Measurement Without Impact

Measuring without impact is difficult. I'll first briefly discuss what this means.

How do you measure something without affecting what you're measuring? Take a simple example of a Unix server. If you have a system that's under fairly considerable load, and you run a script every five seconds that performs a detailed `ps` command (`ps -elf` or `ps -auxw`, depending on your Unix of choice) to determine what's taking up the load, you'll affect the very problem you're measuring.

For those who know a little about quantum physics, you can associate "measuring without impact" with the quantum effect attributable to the uncertainty principle. One of the obstacles with science's quantum mechanics is that it's difficult or impossible to directly measure the state of a quantum bit, or some form of *quanta*. This partially has to do with what's known as the uncertainty principle, but it extends to the fact that to measure something directly, you affect it.

Therefore, measuring the state of some *quanta* would make the observation useless. Although this isn't a book on quantum mechanics, the same problem arises in the higher world of WebSphere optimization and tuning efforts!

You can overcome this problem, but it's important to keep it in mind as you go through the WebSphere performance and optimization process in this book. The way I propose to approach this issue throughout the book is to simply minimize the impact. It's not possible (or extremely difficult) to get around the issue; however, through the careful planning and design of your performance management approach, you can minimize the impact.

At a high level, the methodology I'll use in the book considers the following: Whatever you do in production to measure performance, do in all other environments—specifically, development and systems integration. In other words, control your environments. So often I come across sites testing components in multiple environments, with each environment having slightly different software or patch levels.

Furthermore, keep your performance management monitoring system/application/probes constantly running. This provides a standard and common baseline to measure performance degradation and improvements

under all situations as well as ensure that if there's a common load characteristic for your monitors, its loading factor on the overall system isn't an unknown.

In other words, keep low-level monitoring continuous but nonintrusive. Understand your environment workload and map your monitoring to that. That is, it's no use running debug or monitoring output on every end user transaction if the data you're capturing is too rich in information. Understand what type of transactional workload is operating on your platform (for example, B2B, consumer-based online shopping, and so on), and tune your monitoring tools accordingly, based on transaction rate, depth, and requirements (such as SLAs and so on).

Summary

Throughout this chapter, you explored different aspects of performance management, including a number of optimization models and optimization approaches. These approaches and models will be beneficial to you through the rest of the book and in your day-to-day work when managing WebSphere environments.

As you explored in this chapter, performance management—and hence performance and scalability—can't exist without performance methodologies such as the examples provided within this chapter. I've worked, and continue to work, on both sides of the developer/system manager fence. I know that developers tend to develop to specification and what they consider a high-performing application design.

More experienced developers (and of course architects) understand that high-performing application design goes beyond good code. Good design needs to extend into design and planning between the system people (the readers of this book) and the developers. Writing code that aligns with the WebSphere system manager's configuration and optimization architecture goes without saying, but on many occasions, I've seen this to not be the case. I cover some guidelines on this particular point later in the book.

In closing, although creating and implementing performance methodologies and managing them against a performance management model can incur some overhead, the benefits of having and using these models will be evident in your WebSphere platform's performance and availability metrics.