Pro Java EE 5 Performance Management and Optimization

Steven Haines

Apress®

Pro Java EE 5 Performance Management and Optimization

ISBN-13: 1-59059-610-2

ISBN-10: 978-1-59059-610-4

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Steve Anglin
Technical Reviewers: Mark Gowdy, Dilip Thomas
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editors: Heather Lang, Nicole LeClerc
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Cheu
Compositor: Susan Glinert
Proofreader: Liz Welch
Indexer: Broccoli Information Management
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski
Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

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, e-mail `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 Source Code section.

This book is dedicated to my wife, Linda, and my son, Michael.
Your love has been my inspiration and the purpose of life.
Thank you for loving me the way that you do!

Contents at a Glance

Contents

About the Author

STEVEN HAINES is the author of three Java books: *The Java Reference Guide* (InformIT/Pearson, 2005), *Java 2 Primer Plus* (SAMS, 2002), and *Java 2 From Scratch* (QUE, 1999). In addition to coauthoring and contributing chapters to other books, as well as providing technical editing for countless software publications, he is also the Java Host on InformIT.com. As an educator, Haines has taught all aspects of Java at Learning Tree University and at the University of California, Irvine. By day, he works as a Java EE 5 performance architect at Quest Software, defining performance tuning and monitoring software, as well as managing and executing Java EE 5 performance tuning engagements for large-scale Java EE 5 deployments, including those of several Fortune 500 companies.

About the Technical Reviewer

MARK GOWDY is the manager of the systems consultants for Java Solutions at Quest Software. He has been consulting and working in Java performance for four years and has been active in the Java industry for over eight years. As a consultant, he has assisted Fortune 500 organizations in finding and resolving performance issues in their Java applications.

Acknowledgments

First off, I would like to thank my personal Lord and Savior, Jesus Christ, through whom all of this has been possible. I would like to thank my mother for her support and for helping me stay focused on writing this book. I would like to thank my technical reviewer, Mark Gowdy, for going the extra mile to ensure the quality of this book on an accelerated schedule. I would like to thank John Newsom and Rini Gahir for their internal book reviews and great ideas, and I would like to especially thank Emerald Pinkerton for her hard work and dedication in promoting this book within Quest Software.

I want to thank the top-quality staff at Apress who have helped make all of this possible: Steve Anglin, Beth Christmas, Stephanie Parker, Heather Lang, Nicole LeClerc, and Laura Cheu.

Many thanks to Dr. Bradstreet and the staff at ICDRC for taking care of my son and giving us hope. God's hands are upon all of you and the great work you are performing.

Finally, I would like to thank you, the reader, for giving this book your serious consideration. Performance management is a new and much needed practice in Java EE, and I hope that this book equips you to take control of the performance of your complex enterprise environments.

Introduction

This book is divided into four parts:

- Part 1: Fundamentals
- Part 2: Application Life Cycle Performance Management
- Part 3: Performance Management in Production
- Part 4: Tips and Tricks

In the first part, we explore the nature of application performance and define what is meant by "performance management." Specifically, Chapter 1 sets the stage by reflecting on the state of the Java EE market, provides insight into why performance management is so difficult in a Java EE application, and defines the role of the Java EE administrator. Chapter 2 defines how we quantify and measure performance and explores the costs of measuring application performance. Chapter 3 is dedicated to the details you need to gather to assess the health of your applications' performance and the mechanisms used to gather them. Chapter 4 concludes the part by diving deep into the underlying technologies used in gathering performance information.

The second part, Application Life Cycle Performance Management, addresses every performance-related task that you perform prior to deploying your application into a production environment. Specifically, Chapter 5 addresses how to ensure the performance of your applications during the architecture phase and the performance testing steps required in application development, QA, and production staging to manage performance as applications are developed. Chapter 6 provides an overview of the wait-based tuning approach for applications and application servers. Chapter 7 looks deep under the hood of an application server, at the important metrics to consider when tuning your application server, showing you how to realize 80 percent of your tuning impact with 20 percent of your tuning efforts. Chapter 8 discusses high-performance deployments and deployment strategies that can be employed to maximize performance while considering high-availability and failover requirements. Chapter 9 concludes this section by discussing performance and scalability testing, specifically how to assess the capacity of your environment.

Once your applications are running in a production environment, you have a new set of challenges to address. Part 3, Performance Management in Production, discusses performance from a production standpoint. Chapter 10 proposes using a performance assessment periodically performed against your production environment to assess its health and identify tuning points in both your applications and environment to improve performance. Chapter 11 presents the theory behind a formal production support workflow to help you efficiently resolve production issues when they occur. Chapter 12 looks to the future of your application by providing strategies to trend analysis, forecasting, and capacity planning. Chapter 13 concludes this part by helping you assemble a full life cycle performance management plan.

The book concludes with Part 4, Tips and Tricks, which includes a chapter on common performance problems and next steps. Chapter 14 presents common performance issues that I have encountered in Java EE environments over the past two years troubleshooting production performance issues for companies ranging from government organizations to Fortune 500 companies, as well as strategies to resolve these issues. Chapter 15 closes the book by providing references to additional resources, an action plan, and a guide to your next steps in implementing performance management in your organization.

Although this book builds on itself, chapter by chapter, you can read any chapter individually to address your needs. Where appropriate, the chapters cross-reference other areas in the book for additional information. For example, if your role is production support, then you might start directly in Part 3 and refer back to Parts 1 and 2 as needed for additional information.

Performance management is a serious practice that has been greatly neglected in the Java EE space, and we are counting the costs in lost revenue, credibility, and productivity. My hope is that this book will empower you to take control of the performance of your applications and enable you to focus on more important things than troubleshooting performance issues—namely, providing your customers with the high-quality applications that they deserve.

PART 1

Fundamentals

CHAPTER 1

An Introduction to Application Performance Management

John was driving home from work on Saturday night; it was late by most people's reckoning, but not by his these days. He's the director of development at Acme Financial Services, and his team has been laboring for two years to migrate the company's legacy mainframe business-to-business transaction processor to a Java EE environment. Acme facilitates the transfer of funds from one bank to another. One bank stops earning interest the second the funds are transferred, while the other starts earning interest as soon as it receives them. Working in business banking, Acme's transferring millions of dollars from point to point: they have no room for failure, because missing funds can add up to hundreds of thousands of dollars in only a couple hours.

Over the past four months, John and his team have worked nights and weekends revalidating the architecture, testing the thousands of use cases that it must support, and ensuring that not one cent is lost in a transaction.

"Honey, you're home!" his wife exclaimed at seeing him arrive bleary-eyed at the early hour of 11:00 PM.

"It's been a hard few months, but it's finally over. I'll have more time for you and the kids, I promise. The guys really put in extra effort to make our deadline. Everything is installed and tested, so when the Eastern European market opens in a few hours, we'll be ready for them." He spoke with the confidence derived from months of building architecture, careful design, detailed implementation, and testing. "We did everything right and have nothing to worry about. Let's just get some sleep; we can celebrate in the morning."

At 4:18 AM, his wife was shaking him awake.

"John, it's Paul on the phone for you, and it sounds important!"

"Hi Paul, what's up?" he said with as much clarity as he could.

"John, you have to come in. We're having problems, and I mean big problems! Japan, uh, you've got to come in!"

"Slow down. Tell me what's going on, one thing at a time." Whenever Paul got excited John could make neither heads nor tails of what he was saying.

"John, the servers are crashing. The market opened about fifteen minutes ago, and ten minutes ago the first server crashed. We brought it back up, and then the next two went down. We're bringing servers up just to watch them fall down. What's going to happen when Western Europe opens in a couple hours and our load triples?"

"Okay, hold on, I'm on my way. I'll be there in twenty minutes. . . ."

What happened at Acme Financial? Are they facing a unique issue? Did they simply fail to test their application well enough, or is the problem larger?

Unfortunately Acme's case is more the rule than the exception. In my line of work, I troubleshoot and diagnose production problems in the enterprise environments of companies like Acme all over the world, ranging from small shops with a handful of developers to Fortune 500 companies employing hundreds, even thousands, of developers. The same situation comes up at each company: developers built and tested an application that is either under duress and not meeting its service level agreements or crashing on a weekly, daily, or even hourly basis.

This chapter will consider the definition and implications of quantifiable performance in a Java Platform, Enterprise Edition 5 (Java EE) environment, some hazards to and pitfalls in ensuring quality, and the role of the Java EE systems administrator in this process. The chapter will also briefly outline numerous details within these topics, to be explored in further detail later in the book, such as particular functions of a skilled Java EE systems administrator.

Forrester reported that among companies with revenue of more than $1 billion, nearly 85 percent reported experiencing incidents of significant application performance degradation.[1] Furthermore, in the Network World and Packeteer survey that Forrester references, respondents identified the application architecture and deployment as being of primary importance to the root cause of application performance problems.[2] This means that nearly 85 percent of applications are failing to meet and sustain their performance requirements over time and under increasing load. Formal performance requirements are detailed in service level agreements. A *service level agreement*, or SLA, is a contract that explicitly defines the terms of service that a vendor will provide to an end user. For an application provider, an SLA prescribes the amount of time in which a unit of work must be completed. For example, logging in on a Web site must be completed in less than five seconds.

SLAs can be defined internally by a business to ensure the satisfaction of its end-user experience, such as the speed of at which a Web search engine retrieves results, or it can be a legally binding contract, such as a business-to-business e-commerce application. In the former case, users have been occasionally tolerant of a sluggish application in the past, but increasingly, users now demand better performance, and daily raise the bar on acceptable speeds. A few years ago, a Web request serviced within seven seconds was considered acceptable, and a user would continue to utilize that service. Today however, when a simple request does not respond within three seconds, the user frequently reinitiates the request (thinking there is a problem) or leaves the site for a quicker responding competitor. Even seven seconds is not an option anymore.

In the case of an SLA serving as a legally binding contract, a company uses a provider's services under the promise that those services will, in fact, satisfy the SLA as defined in the contract. The penalty for violating that can be severe, including financial restitution for damages incurred because of the violation or the dissolving of the contract altogether.

Impact of Poor Performance

The impact of poor performance can be quantified in three areas:

---

1. Jean-Pierre Garbani, "Best Practices in Problem Management," Forrester, June 23, 2004.
2. Denise Dubie, "New apps can be a real pain in the net," Network World, July 21, 2003, `http://www.networkworld.com/news/2003/0721appmgmt.html`.

- Lost productivity
- Lost customer confidence and credibility
- Lost revenue

Poorly performing applications can impact productivity in two ways. First, when internal applications (for example, an intranet application) perform poorly, companies are paying their employees to wait for applications to respond. I once worked for a computer hardware manufacturer deciding on the hardware components that would go into the machines and building software bundles to install on them. We used a manufacturing plant to assemble and verify their quality. When a problem was discovered, the line lead would shout, "Stop the line!" All assembly workers would cease building the computers, and we were then called in to troubleshoot and fix problems. Meanwhile the assembly workers sat idle, being paid an hourly wage to watch us troubleshoot problems, and at the end of the day, the number of computers produced was reduced. The loss of productivity for idle workers had to be applied to the manufacturing cost of our computers (our overhead), which cut into our profitability. Similarly, when your employees accomplish less work in the day because of poorly performing applications, it directly impacts your financial overhead and profitability.

Second, when an issue arises in an internal application, those responsible for troubleshooting the problem, who in many cases are developers, must divert their attention from other tasks. This diversion may mean that new features targeted for the next release of a product may be dropped or the delivery schedule may be impacted. Either way, the internal performance issue affects your competitiveness.

Also, poorly performing applications that service other corporate entities directly impact the confidence that they have in both your corporate and personal reputations. When you claim that you can perform a service in a specified amount of time and fail to do so, then losing your credibility is only natural. Consider an employee who commits to delivering a report to you every Friday, but he consistently delivers it Monday afternoon. You grow accustomed to his tardiness, but you know that if you have a task that must be completed by a specific time that he is not the one to give it to. Similarly, a corporation that relies on your services will undoubtedly seek out your competition if your services are not reliable. And as the individual who guarantees and promises these services to your customer, you lose their respect.

Finally, applications that perform poorly can directly affect your revenue by causing you to lose customers. Take one of my own recent purchases for example. Because I travel extensively for my company, I am writing this book, and airplane seats are shrinking on a daily basis, I researched personal digital assistants (PDAs) to which I can connect an external keyboard. Being a technical geek, I did all of my research online, found a couple of models that I was interested in, and then started comparing vendors. My success criteria for selecting a PDA vendor were customer feedback, reputation, availability, and finally price. My search returned 14 vendors, and I connected to their sites to gather information. Two of these vendors did not respond within an acceptable period of time. (My tolerance for something like this is about ten seconds.) I simply skipped those vendors and moved on to the next one on my list. Regardless of how you define performance criteria, your users' perception of your application is really all that matters—and there are ways to mitigate the poor perception of performance, such as a progress bar or a running countdown. I may very well have missed the vendor with the best reputation, price, and delivery schedule, because its application did not perform acceptably or appropriately use mitigating features. This needlessly lost sale is a reality facing businesses at present.

Regardless of whether you are developing business-to-business, business-to-consumer, or internal applications, you need to address the performance and reliability of these applications. The impact of a poorly performing application can vary from mild to severe, but it can always be measured if you take the time to analyze it. Only a proactive approach of implementing a formal, performance-based methodology will maximize your chances of success.

Complications in Achieving Application Performance

If 80 percent of all production Java EE applications are failing to meet their performance requirements, then achieving Java EE application performance must be complicated, but why? This section explores some of the reasons Java EE application performance considerations can be overwhelming.

Evolution of Java Applications

As technology evolves so does the way that we use that technology. Consider the evolution of computer hardware. Today's desktop computers are exceedingly faster and have more memory and storage capacities than they did a decade ago, but how much faster is Microsoft Windows XP than Windows 3.1? The speed difference is minimal, but its capabilities and appearance are far superior. Instead of allowing faster hardware to run existing operating systems faster, the extra processing capabilities have been used to develop more robust operating systems and, as a result, have greatly improved productivity.

The evolution of Web applications has followed a similar progression. The first Web sites served static content: when a vendor added new products to his catalog, he was required to update the physical HTML files that rendered it. This requirement quickly became a management nightmare, so databases were incorporated with scripts built to generate HTML pages from database content. Tools and frameworks evolved to accomplish dynamic Web content generation more efficiently and soon standards emerged.

In 1997, Sun released the servlet specification which enabled developers to build Java programs that used existing code and a robust set of classes to generate HTML pages. But difficulties arose in implementing presentation details inside a Java servlet (for example, changing a font size meant changing Java code, recompiling it, and redeploying it to a servlet container), so Sun released the JavaServer Pages (JSP) specification in 1999. JavaServer Pages enable us to build HTML-looking documents that contain embedded Java code to generate dynamic content. At run time, JSPs are translated into servlet source code, compiled, and loaded into memory. Therefore simple changes to presentation details could be accomplished on the fly without requiring a real person to recompile and redeploy the servlet.

Shortly after, it became apparent that complicated business logic hindered the readability and maintainability of JSPs. Understanding that servlets were very good at implementing application business logic and JavaServer Pages were equally good at generating HTML pages, we, as an industry, began implementing a variation of the *Model-View-Controller (MVC)* design pattern. In MVC architecture, JavaBeans represent data (Model), JSPs performed the presentation (View), and servlets represent application business logic (Controller). This delegation of programmatic responsibility resulted in more vigorous and maintainable applications.

As business requirements utilized new technological capabilities, Sun introduced the concept of Enterprise JavaBeans (EJB) to provide transactional integrity and a strong delegation of responsibilities within the business tier. Servlets are now only responsible for application flow and logic, while Enterprise JavaBeans are responsible for business logic and object persistence. Using Java to build enterprise applications presented both positive and negative effects, and by analyzing those effects we discovered best practices that led to a collection of design patterns. These patterns are equipped to solve more complicated problems, which allowed business requirements to evolve.

Web applications evolved into portals with user-customizable content subscription, a single sign-on, and an advanced user-security model. The next wave of evolution came with the advent of Service-Oriented Architecture (SOA) built on top of Web services. SOA facilitated the integration of disparate systems, including interoperability between applications written in different programming languages and running on different operating systems.

The more that Java EE developers increase what we can do, the more users require of us. This brief historical overview of Java's dynamic Web-content generation evolution demonstrates that as our technology improves, our business requirements evolve to use that technology. Java Web-based applications written in 1997 were infinitely simpler than today's. As the complexity of the code increases, our capability to easily identify performance problems decreases.

Layered Execution Model

The first complication in Java EE application performance is the inherent architecture of the Java EE platform, which necessitates a layered execution model. The benefit gained by embracing Java EE as a deployment platform is hardware and operating system independence. To utilize these benefits, we write our applications to adhere to formal specifications and deploy them to an application server running in a Java Virtual Machine (JVM) on an operating system on a physical computer (hardware). In its simplest form, a Java EE application requires all of these components running on a single machine, shown in Figure 1-1. We refer to this complexity of a single application server instance as *vertical complexity*.

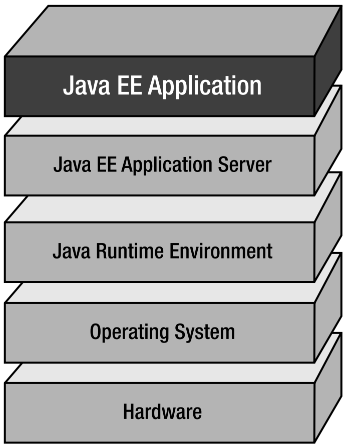

Figure 1-1. *A Java EE application requires a layered execution model.*

Because of this layered model, the location of a performance problem can be in the application code, in the application server configuration, in the JVM configuration, in the operating system configuration, or in the hardware itself. To ensure proper performance of your application and diagnose performance problems, you need to master of each of these layers and understand how to attain their ideal configurations. To further complicate matters, most significant Java EE applications do not run inside of a single application server instance but, rather, run in a distributed environment. In a *distributed environment*, the same layered execution model is spread across multiple machines. Then too, for your application to accomplish anything beyond simple dynamic-content Web page generation, it will need to interact with other systems such as databases and legacy systems. Figure 1-2 puts all of these components together.

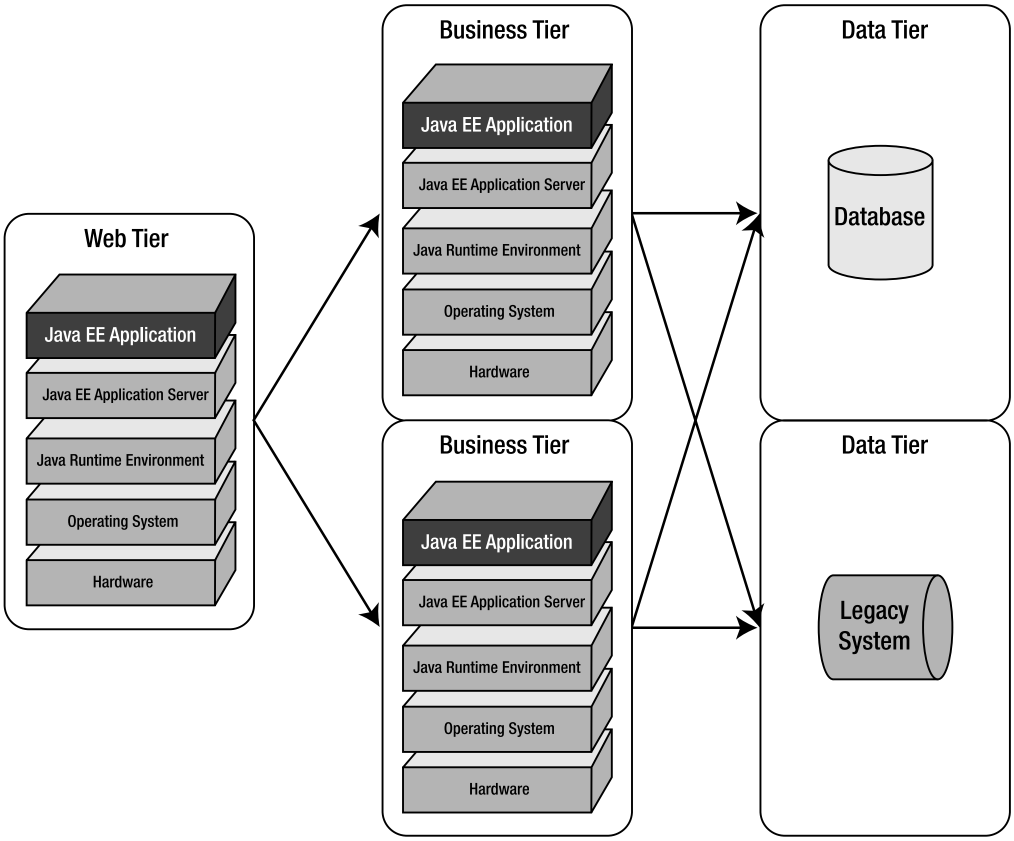

Figure 1-2. *Significant Java EE applications require multiple application server nodes and interactions with other external systems such as databases and legacy systems.*

When your users complain that your application is running slow, identifying the root cause is a daunting task, because the problem can be in any layer in any tier on any application server instance or in an external dependency. We refer to this distributed complexity as *horizontal complexity*. Horizontal complexity issues can manifest themselves when your application is subjected to a significant load: the nature of certain performance problems is to arise only outside the context of a single JVM. Large loads cause seemingly small issues to become large issues.

The combination of horizontal and vertical complexities equates to more moving parts in your application environment than a typical Java EE developer can be expected to handle. Because the proper deployment of a Java EE application requires mastery not only of an application server environment, but of the application server topology as well as detailed skills in the configuration of each external dependency, the best operational model is not a single individual, but a team of skilled individuals specializing in each respective arena.

Prebuilt Frameworks

As you may surmise from the previous discussion, the generation of a robust MVC enterprise application is not a trivial task. As a result, several organizations built application frameworks that simplify the demands on the application: the application integrates its business logic into the framework, and the framework manages the application flow and logic. Most of these frameworks have open source licenses, with the more popular ones being Apache Software Foundation's Jakarta Struts and Velocity, and the Spring Framework.

Prebuilt frameworks offer a number of benefits:

- Productivity increases because most of the mundane work of building infrastructure is removed.
- Large open source development communities offer rapid development.
- Wide adoption means that many developers have tested the framework before you, and those who wrote the code have already handled initial troubleshooting.
- Implementation of application enhancement requests is quick. Because prebuilt frameworks are targeted at solving generic problems, changes to your application requirements will most likely already be supported.

While these benefits should persuade you to adopt an existing application framework, incorporating someone else's code into your application has dangers. Unless you spend the time to become an expert on the internal workings of the prebuilt framework, troubleshooting subsequent problems is difficult because using that framework introduces a black box into your application. A *black box* is a component or piece of functionality that you understand how to use but not necessarily how it works: you provide the inputs to the black box, it performs its functions, and it returns its results to you. Therefore when a problem arises you have another layer in your layered execution model to analyze to discover the root of your problem.

Furthermore, if the framework does, in fact, have a performance issue that impacts your business, then you either must fix it yourself or request that the framework provider fix it. In the former case, if your changes are not committed back to the formal framework repository, then you could have problems upgrading to future releases of the framework. In the latter case, your issue might take weeks or months to reach an acceptable resolution.

I fully support implementing prebuilt frameworks in new development efforts, but I also recommend that you spend the time up front to understand the architecture of the framework that you choose. This way, if a performance problem does occur, you will be better equipped to troubleshoot it. Furthermore, I suggest you research the existing frameworks and choose a popular one that best fits your business requirements. The popularity of the framework will help you when it comes time for acquiring bug fixes and obtaining troubleshooting guidance.

Java EE Expertise

Understanding how to use a technology is a far cry from being an expert at using it. In this respect, Java EE is especially dangerous as its specifications define recommended approaches to application design, but they do not force any particular implementation. This was done by design, because although a full MVC Web architecture is more scalable and robust than a stand-alone servlet, it may not be the best tool to solve a problem. The flexibility of Java EE empowers you with all of the capabilities to develop a small, lightweight application or a large enterprise-scale application; the choice is yours.

However, a Java EE developer can develop a functional application that performs adequately in unit tests, but falls apart under heavy loads. Having the knowledge to build the application does not necessarily mean having the experience to build it to meet your business requirements. Because Java EE has been gaining in popularity over the years, particularly as a platform for enterprise applications, more and more developers are moving over to Java EE and becoming acclimated as quickly as possible. Many of these developers may bring bad habits from other programming languages, and some learn enough to build an application, but not enough to comprehend the impact of their implementation decisions.

Java EE is a relatively new technology, so it is not as easy to find a seasoned Java EE architect as it is to find a seasoned C or C++ architect. This shortage in Java EE experts can directly impact the performance of your applications if you do not take precautions to ensure that someone with rock-solid experience leads your team. A competent developer can become competent in any language and environment given proper time to acclimate; just be sure that your architects and team leads are already well acclimated before your project begins.

Development Tools

Development tools are evolving in two ways that *may* negatively impact the performance of Java EE applications. I emphasize the word "may," because, while a good tool can work miracles, a good tool in the hands of an unknowledgeable person can wreak havoc on your environment.

First, tools are being developed to relieve many of the mundane activities performed by Java EE developers. This will undoubtedly improve productivity as long as the developer understands the impact of decisions made inside the tool. During the days of early Windows programming there was a debate between Visual Basic and C. C and C++ programmers argued that Visual Basic programmers did not know how to program, while Visual Basic programmers flaunted their productivity; they could build a robust application in a quarter of the time that a seasoned C++ expert could. The underlying problem was that Visual Basic covered up many details about how the resultant application worked, so that someone who was not familiar with the fundamental structure of a Windows application (for example, the message pump, window messages, or threading models) could develop an application that satisfied the functionality of the business requirements, but performed atrociously. On the other hand, empowering a knowledgeable person with such a tool would increase his productivity. Likewise, many of the underlying details involved in building a Java EE application can be automated and as long as the developer understands the implications of his inputs into that automation process, then he will be more productive and still retain high-performance.

A second evolution to consider is the new breed of Java EE tools coming to the market to facilitate application assembly. The idea is that an application architect will be able to assemble an application from existing components using a graphical tool without writing a single line of Java code. The concept is fascinating, and if these vendors deliver on their promises, then productivity will certainly improve. One of the biggest tools in this market is BEA AquaLogic, a relatively new tool with unknown industry acceptance that could revolutionize enterprise application development if it delivers on its promises. But again, this technology heightens the risk of allowing tools to do our work for us without requiring us to understand what they are doing.

Service-Oriented Architecture and Web Services

Every time new technology enters the software industry, it is met with a combination of skepticism, in wondering if the technology will deliver on its promises, and enthusiasm for its potential impact on the way we develop software. In my experience, no technology has ever met all promises and only time can tell how much impact it has on our lives. One thing is for sure: CIOs like buzzwords and eagerly adopt best-of-breed technologies, even if they are not ready for prime time.

Service-Oriented Architecture (SOA) is an example of a technology that has crossed over from fad into widespread adoption, and is only now beginning to deliver on its promises. SOA promotes the concept that software components should be written as services and expose their functionality openly: any component needing the functionality provided by a service simply calls the service rather than reimplementing the functionality itself. The current practical application of SOA is in the integration of disparate systems. SOA and its implementation on top of Web services make connecting the functionality of a .NET server with a Java EE server and a legacy application incredibly simple. Simply drop a service in front of your functionality and voilà—instant integration.

Please note that SOA and Web services are not the same thing. SOA is a design methodology or architectural concept, while Web services are a collection of technologies that enables SOA. Web services itself is a platform- and technology-agnostic collection of specifications by which services can be published, be discovered, and communicate with one another. SOA is the software engineering concept through which you build applications.

From a technology standpoint, Web services are incredible. But from a management and performance standpoint, they can be tricky if you are not prepared. You now have server platforms with different operating systems running multiple applications and application servers to comprise a single application. Figure 1-3 shows this graphically.

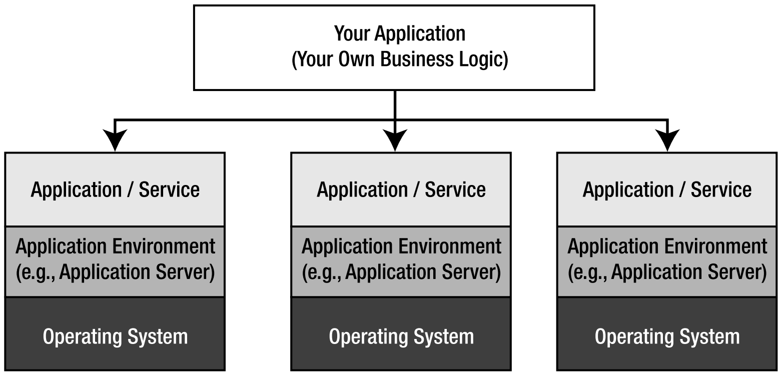
Your Application
(Your Own Business Logic)
Application / Service
Application Environment (e.g., Application Server)
Operating System
Application / Service
Application Environment (e.g., Application Server)
Operating System
Application / Service
Application Environment (e.g., Application Server)
Operating System

Figure 1-3. *Developing an application from a collection of Web services integrates different application environments, operating systems, and hardware.*

In order to effectively manage this type of environment, you need visibility at all technology points, including

- Each operating system upon which each service is running
- Each technology component in each layer of the distributed layered execution model that supports the service in Java EE environments
- The performance of the enabling technologies as well as the application components that are supporting the service in non-Java EE environments
- Other external dependencies such as database and external servers that may be hosted offsite
- The network communication behavior between your application and its services

The benefits of using Web services outweigh many of these concerns, but the inherent complexity and verboseness of a Web services implementation are prohibitive to optimal performance. Consider the steps that must be performed for a single Web service call:

1. The caller creates a complex XML file.
2. The caller then transmits that XML file to the service.
3. The service infrastructure translates the XML file into an instruction set that the service understands.
4. The service implements its business logic.
5. The service infrastructure constructs a complex XML document containing the results of the business logic implementation.
6. That resultant XML file is then transmitted back to the caller.
7. Then the results of the service call must be translated back to application-specific values.

If these are the steps involved in using a single Web service, consider the steps for an application built by an application assembler that may access half a dozen Web services to service a single Web request. If one Web service call translates to the construction, transmission, and disassembly of two complex XML documents, then doing this six times requires the construction, transmission, and disassembly of twelve complex XML documents. Regardless of how well-written the code and fast the network, performance is going to be abysmal. So while the technology enables many sought-after capabilities, the inherent complexity of implementing that technology necessitates careful planning and analysis in order to benefit your organization.

With all of these pitfalls, should we simply avoid using Web services? Can we count on their adoption being minimal? Or should we take a proactive yet cautious approach to embracing the technology?

The industry analysts have voiced their approval of the technology:

> IDC Researcher Sandra Rogers in a 2005 study predicted that the worldwide Web services market will hit $15 billion by 2009, driven by major vendors such as IBM, Microsoft, BEA Systems, and Sun Microsystems.[3]
>
> "Gartner's Positions on the Five Hottest IT Topics and Trends in 2005" includes a review of Service-Oriented Architecture and predicts that by 2006 more than 60 percent of the $527 billion market for IT professional services will be based on Web services standards and technology.[4] By 2008, 80 percent of software development projects will be based on SOA.

SOA is not a fad but, rather, a technology that has the potential to greatly increase productivity and save companies millions of dollars if implemented intelligently.

Application Performance

When someone asks you about the performance of your enterprise applications, what do you think they mean? What does performance means to you?

Performance means different things to different people, usually based on their role in measuring and ensuring the performance of their area of responsibility. When we break down the development organization into groups, we call each group a *stakeholder*. And each stakeholder has an area of responsibility that dictates what that person considers to be the definition of performance.

From the perspective of an application support engineer, whose panicked life is framed by user complaints, the primary criterion for performance measurement is application response time. If the application responds in a reasonable amount of time, then users do not complain, and the engineer's efforts can be spent on more interesting tasks.

A Java EE administrator is more concerned with the supporting environment and hence measures performance through resource utilization and availability. The Java EE administrator determines when to add application server instances, when to change configurations, when to add hardware, and so on. The worst time to make major architectural changes to an environment is when users are complaining; when users complain, then it is already too late. Rather it is best to perform a capacity assessment of the environment and correlate current usage patterns with resource utilizations to determine if the application is approaching its saturation point. Recognizing the environment's saturation point and being able to discern how soon it will reach it empowers the Java EE administrator to plan application server architectural changes. Another significant consideration in his job role is the availability of the application servers. If the application servers are not available, then the code execution, database, and network traffic performance levels are meaningless. For the Java EE administrator, then, good performance implies effective resources that are readily available.

A database application programmer's perspective is primarily concerned with the response time of the Structured Query Language (SQL) and how quickly it services database requests as well as different query execution plans. Creating or removing indices, and optimizing SQL queries to meet the demand of the application against the volume of data in the database are also of concern, particularly considering that the most optimal query for small database is not

3. IDC, "Worldwide Web Services Software 2005-2009 Forecast: Let the Races Begin," May 2005, `http://www.idc.com/getdoc.jsp?containerId=33418`.
4. Gartner, Inc., "Gartner's Positions on the Five Hottest IT Topics and Trends in 2005," May 12, 2005, `http://gartner.com/DisplayDocument?id=480912`.