Realtime publishers

The Five Essential Elements of Application Performance Monitoring

Don Jones

sponsored by

QUEST SOFTWARE®

Copyright Statement

Chapter 4: Diving Deep into Your Application Components

With a valid and comprehensive application stack mapped out, you can start monitoring specific components' health. But when something goes wrong with a component, how do you figure out *exactly* what went wrong and start to take action toward fixing it? You could take the traditional approach and have IT specialists dig into the component using domain-specific tools—which can be difficult to do and often leads to specialists arguing with each other about whose components are really at fault. The 5D approach is to use a consolidated toolset that can dive deep within components to identify specific problems. You're moving beyond component health, now, and starting to diagnose specific root causes by looking under the hood.

Moving Further In: Going from Health Problems to Component Diagnostics

In their paper "Magic Quadrant for Application Performance Monitoring," Gartner defines this fourth dimension of APM as "Application component deep-dive monitoring," describing it as:

> …a diverse set of technologies. In addition to the higher-level application health portraits obtained by the first three dimensions, effective diagnosis of performance problems frequently involves "looking under the covers" of the critical elements that hold a modern, highly modular application stack together; such elements include database management systems, application server middleware, message-oriented middleware, off-the-shelf application stack frameworks and even some aspects of the network infrastructure. (Byte code instrumentation is a frequently favored way, for example, of deep-dive monitoring for application servers.)

It's important to note that a *lot* of APM solutions don't really provide this dimension. Many of them stop at the server level; they can tell you when a database server isn't performing well, for example, but they can't dive deeper to tell you which bit of a Java or .NET Framework application that uses that database is holding up performance. As I'll discuss in the next chapter, proper APM depends on having all five dimensions in the 5D approach, and this fourth dimension is especially important for quickly pinpointing the root, actionable cause of a problem.

Using Health to Focus Your Efforts

When an application's performance starts to fall below your goals, where do you begin your troubleshooting and corrective efforts? Obviously, you need to dig into your application at the component level, but what component do you look at first?

Health is one way to focus your efforts. I make a distinction between health and performance:

- *Performance* is raw data. It tells you how, from a performance perspective, a particular component is performing at a given moment. At a simplistic level, this might be something like "80% CPU utilization" or "12ms to complete this query." Performance data is typically contextual, meaning that it requires some interpretation—either by a tool or by your brain—in order to have any real-world meaning. Performance is neither good nor bad; it just is what it is.
- *Health,* however, is what you get when you take raw performance data and apply some kind of threshold to it to define "good" and "bad" performance. "This component is healthy if the query takes less than 20ms to execute," for example. These thresholds are what help focus your efforts, by enabling you to more quickly identify components whose performance is out-of-bounds, abnormal, and so forth.

Take the simplistic example console shown in Figure 4.1. This shows both raw performance—the line charts—as well as health thresholds, which are shown as dashed lines in the middle column of information. Performance values that exceed the predefined thresholds result in an "alert" indication, such as the large red "X," that help draw your attention to a component or application element that isn't performing where it should be.

Figure 4.1: Monitoring application health *and* performance.

Consoles like these are designed to quickly direct your attention to whatever application element is creating the most-immediate performance issues, helping you to triage your troubleshooting efforts. Figure 4.2 shows another console, which may direct your attention to an entire server or help you drill down into individual application elements.

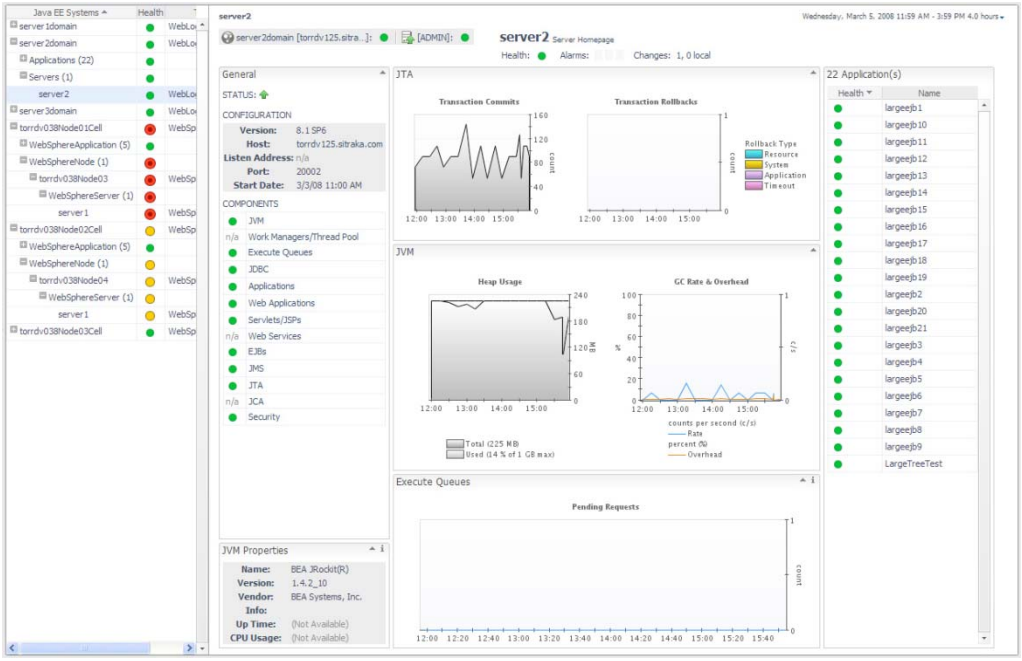

Figure 4.2: Application-wide health monitoring.

Again, the goal here is *health,* not performance *per se*. You want a dashboard that quickly gets your eyes on the problem—in Figure 4.2, those red dots are the things that need your attention. Once you know more specifics about *where* the performance problem is occurring, you can switch to more domain-specific tools to troubleshoot further.

Stopping the "Toss it Over the Fence" Mentality

I frequently see IT teams waste a lot of time "pointing fingers" or "tossing the problem over the fence" between IT disciplines or silos. For example, when an application is exhibiting poor performance, the database administrators (DBAs) may blame the network, while the network engineer points his finger at the software developers, who blame their programming framework, and so forth.

The problem is that these individual disciplines tend to look at *performance* data without necessarily translating that into *health* information and without looking at it in the context of the application. They're all using domain-specific tools as their starting point, which means that everyone is looking at different numbers, from different places, having different meanings. The result is often that nobody can agree on what's "broken," and so nothing gets fixed—the problem just passes from silo to silo.

A top-level, application-wide component monitoring dashboard can help get everyone on the same page. In Figure 4.2, for example, it's very clear which component is causing problems—and the IT person responsible for that component can get on with the actual troubleshooting rather than trying to pass the problem to another team member.

Using Domain-Specific Tools to Gather More Information

Once you know what's causing or contributing to the problem, your IT team can start using their domain-specific tools to gather more information, narrow down the root cause of the problem, and start implementing a fix. This may involve tools that are independent from your actual APM solution, but it may also involve tools that are a part of the APM solution.

For example, if the root component seems to be the database server, your DBAs may turn to vendor-specific tools to check performance, monitor query execution time, and so forth. A good APM platform may also provide tools to help them do so—and, in fact, *should* provide these tools. Figure 4.3 shows an example of this.

Figure 4.3: Database-specific troubleshooting within an APM solution.

An advantage of having this level of troubleshooting built right into the APM solution is that it provides faster and easier drill down. From the main dashboard, IT experts can quickly access more detailed diagnostics without having to launch a separate tool, configure a troubleshooting session, and so forth.

Every element of the application will, at some point, need this kind of domain-specific troubleshooting; there's no one-size-fits-all way to troubleshoot every component. For example, cache sizes and query execution times are critical when troubleshooting database performance but meaningless when troubleshooting a virtualization host's performance. The more discrete elements an APM solution can troubleshoot directly, the more value it will offer your team. For example, Figure 4.4 shows how the same solution that troubleshoots an Oracle database can provide distinct and domain-specific troubleshooting for VMware hosts as well.

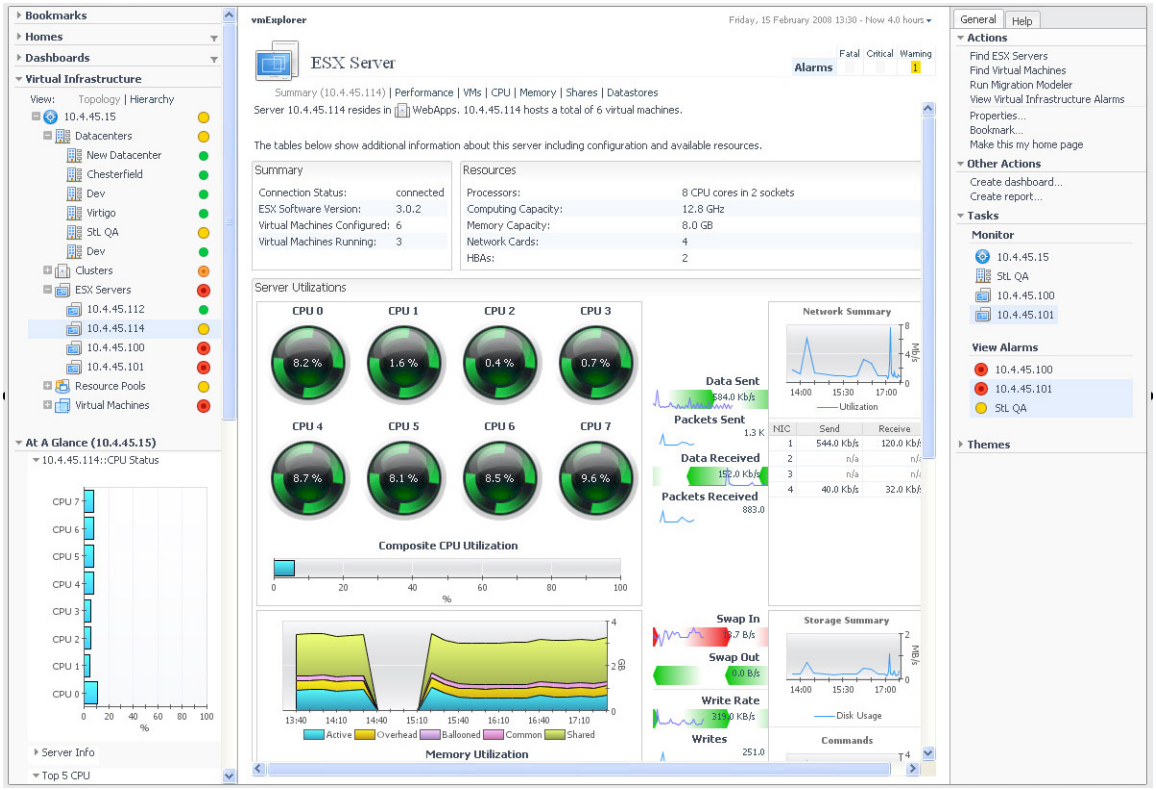

Figure 4.4: Drilling into VMware performance within an application.

A bit later in this chapter, I'll examine some of the specific capabilities that you should look for in an APM solution with regard to application component deep-dive.

The 5D Approach: Flow-Up and Flow-Down

Every application can be modeled as a stack or diagram of interconnected components, such as the fairly simple application model in Figure 4.5. In fact, as I discussed in the previous chapter, having a clear and accurate model of your application is crucial to the 5D APM approach because it helps you clearly understand how the different components affect each other and rely upon each other from a performance perspective.

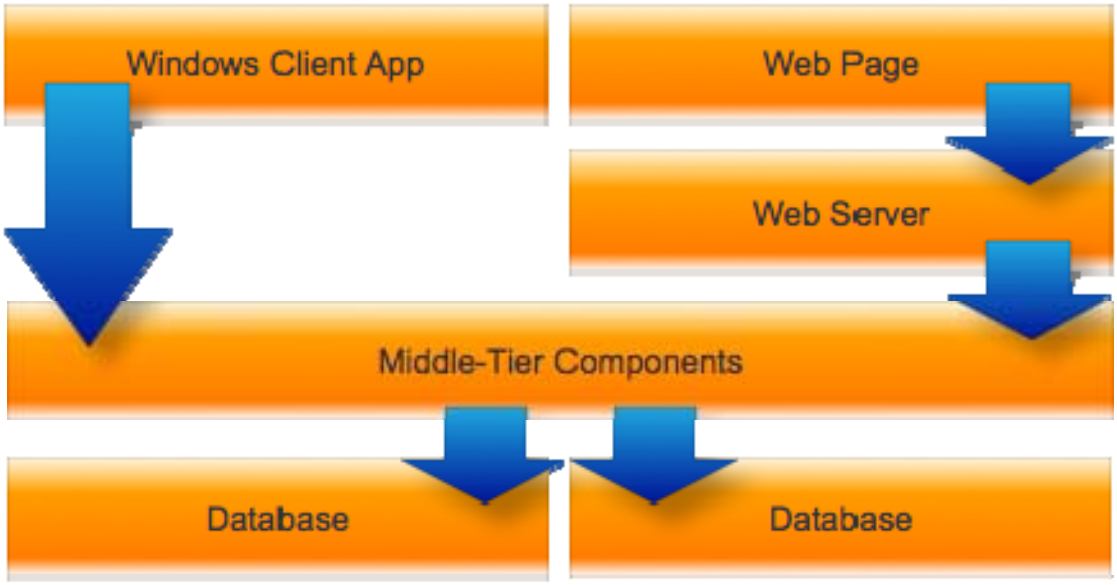

Figure 4.5: Simple application model.

There are two ways to approach performance problems using this model: flow-up and flow-down.

Problems Flow Down the Model

We usually say that problems flow *down* the model, from the user's perspective to the most deeply-buried components that are furthest from the user. For example, in Figure 4.6, a user is experiencing poor response times from a Web page that they're trying to access. From the end user's perspective, that's the *problem* with the application: The Web page is slow. In other words, the end-user experience (EUE) metric is telling us that users aren't having a good time of things.

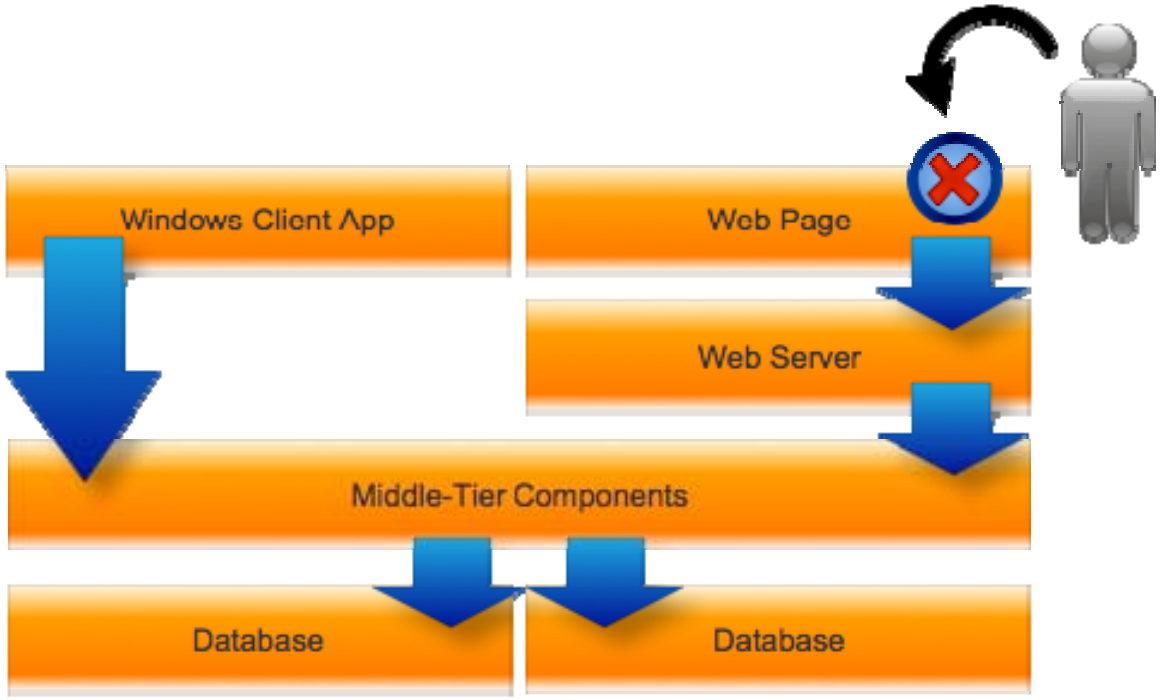

Figure 4.6: Poor EUE is observed at the top of the model.

If you simply took the EUE, which is the first dimension in the 5D approach, your next step might be to check the Web page for errors. After that, you might check the Web server's performance, then check the middle-tier components. The problem is flowing *down* the model, and you'd be investigating each subsequent component to see whether it was causing the problem.

This isn't a *bad* approach, but it can be time consuming. One potentially confusing aspect of the 5D approach, in fact, is its emphasis on the EUE as a top-level metric. In this book's first chapter, I really emphasized the importance of the EUE in measuring application performance. That doesn't mean you start *troubleshooting* with the EUE, though. The EUE is simply your indicator that troubleshooting *is needed;* it will rarely be your starting point.

Root Causes Flow Up the Model

Instead, you have to recognize that the *root cause* of a problem will flow *up* the model. In other words, a bad EUE isn't the problem; it's a symptom of the problem caused by something lower in the application stack. Figure 4.7 illustrates this.

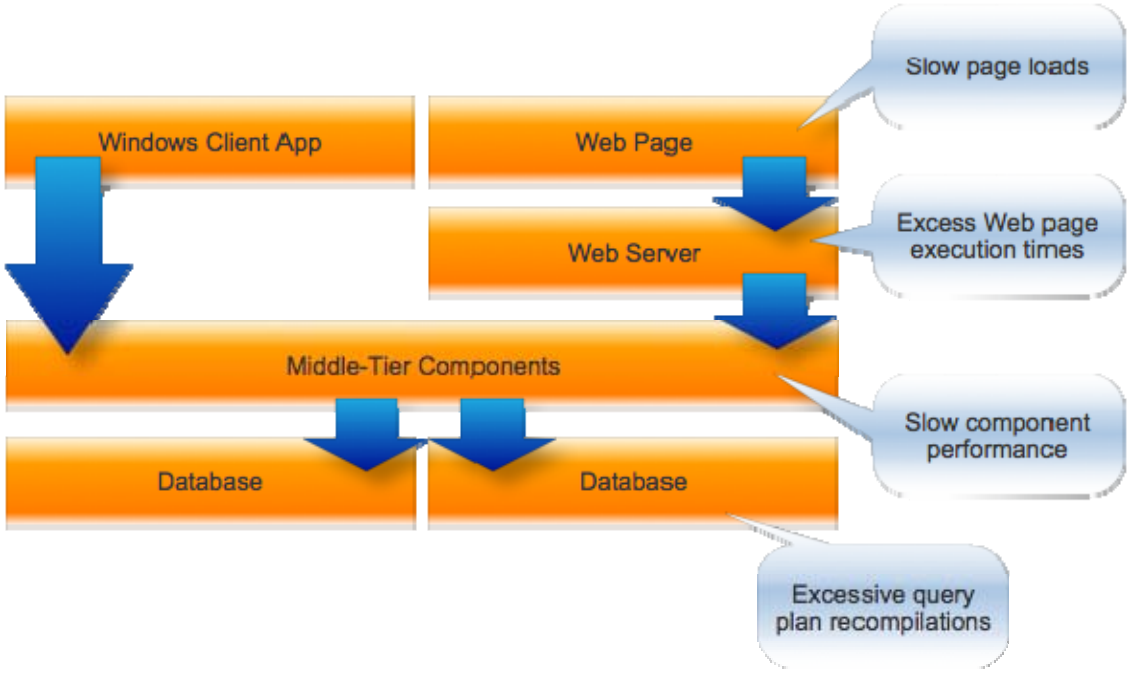

Figure 4.7: Root causes flow up the application stack.

Here, a database server is experiencing excessive query plan recompilations, which slows querying. That results in a middle-tier component performing slowly, as it waits on the database to deliver data to it. That slow middle-tier component causes Web scripts to execute more slowly because they're waiting on the component to deliver whatever the Web page needs to display. That, in turn, results in longer response times and page loads for the end user.

The 5D approach—combined with tools that implement the approach—enables you to see the root cause more directly. Rather than starting at the top and drilling down, these tools let you respond to a poor EUE by simultaneously monitoring the health of every application component at once. By tracing user transactions through the application model—dimensions two and three of the 5D approach—these tools can highlight the problem component immediately, allowing you to focus your efforts and begin deep-diving into the problem component to solve the problem more quickly.

Caution: Good Health Does Not Necessarily Equal a Good EUE

I want to offer a caution here. An APM solution that truly implements the 5D approach will *directly* measure the EUE, either through direct user transaction mimicry—having an automated process visit a Web page, for example, to see how fast it loads—or by monitoring user transactions (either real or simulated) as they flow through the application. You cannot *derive* the EUE; it must be directly measured to be useful. I've seen APM solutions that claim to offer EUE metrics, when in fact all they do is add up the individual component health information and aggregate that as an "EUE." That's not an EUE; it's an aggregate. Depending on how you've set up individual component thresholds, every component could be "healthy" and still, in total, deliver an unacceptable EUE. Make sure that your APM solution is *directly* monitoring the EUE, not deriving it from lower-level performance data.

Deep-Dive Areas and Techniques

In the next few sections, I want to talk about some of the major capabilities that you should look for in component deep-dive features. In general, you may find that solution vendors don't offer a "one-size-fits-all" package that includes every one of these; more commonly, you'll find vendors offering a base APM framework, to which you add deep-dive capability modules. That lets you add, for example, Oracle or Java deep-dive capabilities if you need those technologies, while perhaps omitting capabilities that aren't in use by your organization. I think that almost every organization will need all of these general *categories,* although every organization will vary in the exact brands (for example, SQL Server vs. Oracle or Java vs. .NET Framework) that they need to monitor. I'll keep the capabilities descriptions focused at the category level for this reason.

Database Management Systems

Relational database management systems (RDBMSs) are complicated pieces of software with dependencies on the underlying operating system (OS), storage, memory, processor, and other resources. They're a very specialized OS in and of themselves, with multiple in-memory caches, execution procedures, and so on—monitoring and troubleshooting them can be challenging. Figure 4.8 shows some of the basic monitoring capabilities I'd expect—many of which are often difficult to dig out with RDBMSs' native administrative tools.

Figure 4.8: Key RDBMS monitoring elements.

Of particular importance:

- General statistics such as server processor consumption, memory utilization, and disk queue length (the amount of data waiting to be written to disk) are important top-level items to monitor.
- Database cache sizes—such as procedure caches, page buffers, and so forth—help you ensure that the server has the right amount of memory to maximize performance. Cache hit ratios are also important.
- Monitoring in-database processes like replication.
- As RDBMSs are almost always bottlenecked at the storage layer, detailed views into disk utilization are key. This should include physical read and write operations as well as read-ahead caches, lazy writes, log flushes, and so on.
- Other capacity information—database sizing, log file sizing, active connections, and so on—can help provide proactive alerts when specific capacity thresholds are reached.

Having all this information in a dashboard-style health view—rather than in raw performance numbers—can help direct attention to areas that require it most urgently.

You also need *deeper* statistics than just these server-level ones. A good APM solution can also trace specific query statements, offering performance monitoring at an extremely granular level. Some APM solutions can even analyze slow-running queries and offer performance optimization suggestions, such as redesigned indexes. Although many RDBMS platforms provide native tools to accomplish this, having everything rolled up into a single APM solution can help you immediately pinpoint a slow-running query as the root cause of a poor EUE and a breached SLA.

Application Server Middleware

In most cases, you'll always need some kind of general server monitoring—processor utilization, memory utilization, disk throughput, and so forth. Any server participating in the application will need this level of monitoring, likely in addition to other, more specific forms of deep-dive monitoring that are covered in the following sections. Figure 4.9 shows the type of information you're looking for.

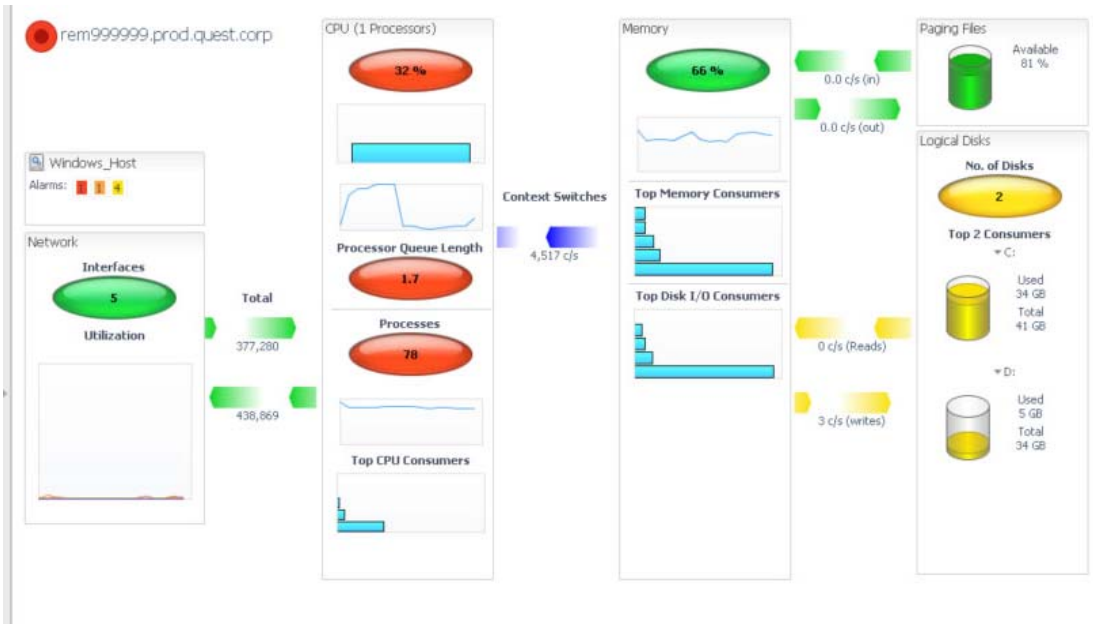

Figure 4.9: Monitoring application servers.

Some of the key items your APM solution should track:

- Network utilization—It's actually not unusual for modern servers to outperform less-modern network connections; any server built within the past couple of years, for example, can easily overwhelm a 100Mbps Ethernet connection, which is why servers typically utilize Gigabit Ethernet or better connections.
- Processor utilization—This item is of course important, but so is processor queue length—the number of instructions waiting to be executed by the processor. A long queue length is often a better indicator of an overloaded server.

- Memory utilization—This item is important primarily in terms of which processes are using it. You should establish baselines for what looks "normal" in your application, and use those to create health thresholds that will alert you to abnormal conditions.
- Memory—This item is related to disk throughput in OSs that use paging files to create virtual memory. Tracking the page file utilization as well as paging speed and throughput is another critical performance element.
- Overall disk throughput—reads and writes as well as queue lengths—can help you determine whether a disk subsystem is being overwhelmed and acting as a bottleneck.

These numbers are often easy to pull using native tools, such as Windows' System Monitor console. However, having them integrated into an APM dashboard—which is what Figure 4.9 illustrates—can help you visualize the connections between the components, and provide better *health*-based monitoring.

There's more to monitoring an application server than just keeping an eye on the *server* and its OS; there's also the *application* part of the stack—typically an application written in a framework such as Java or .NET. For the standalone portions of such applications, read ahead to "Off-the-Shelf Application Frameworks;" however, there are server-based applications that run, for example, Java as middleware components. WebLogic, WebSphere, and other products are examples of these, and you'll want an APM solution that knows the specifics of those platforms and how to monitor them so that you can get the *entire* picture of your application servers' performance.

Message-Oriented Middleware

Modern applications can rely intensely on the ability for components to communicate with each other, often including asynchronous message queue-based communications. Monitoring the message-oriented middleware—whether it's IBM WebSphere MQ, Microsoft Message Queue Services, or something else—is a critical part of maintaining the performance of such applications. Figure 4.10 shows some of the major elements you'll need to monitor. These include the overall throughput of the queuing product as well as each individual message queue that the product manages. You typically want to see small queue lengths (or "depths"), meaning that messages are being picked up and processed roughly as quickly as they're being placed on the queue.

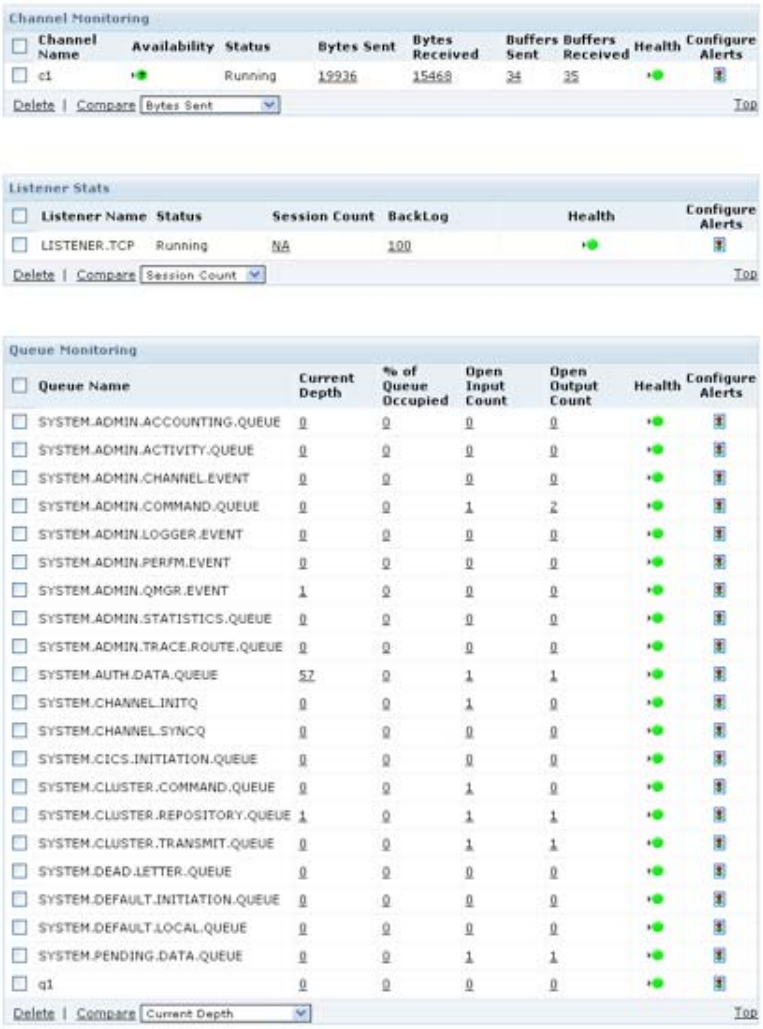

Figure 4.10: Monitoring message queues.

Off-the-Shelf Application Frameworks

Being able to deep-dive within application programming frameworks—Java and .NET Framework are two of today's most popular ones—lets you "see inside" your application components and ensure that they're performing well at a fairly deep level. For example, consider Figure 4.11.

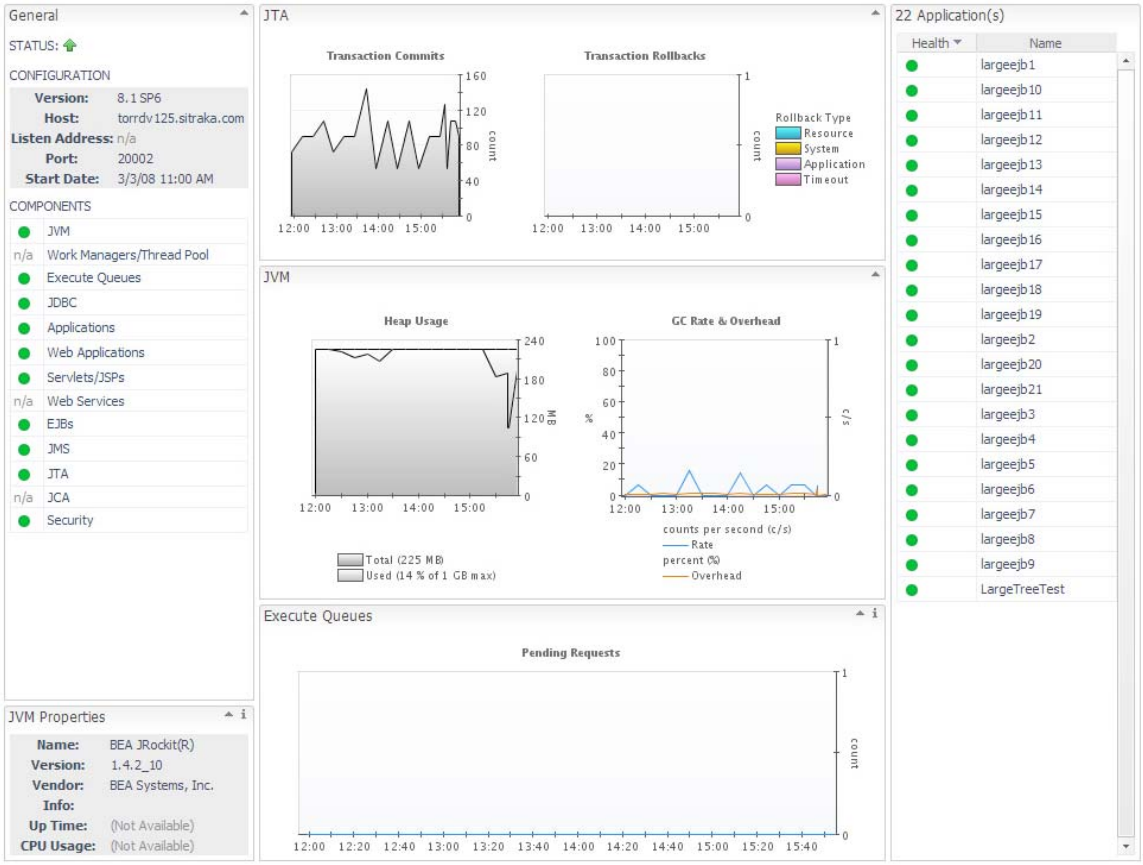

Figure 4.11: Monitoring inside a Java component.

Here, you can see overall Java Virtual Machine (JVM) information in the lower-left, along with detailed statistics in the main portion of the dashboard. Monitoring transaction commits, rollbacks, heap usage, garbage collection rate, and so forth can help you spot poorly-written or underperforming components. For example, high levels of garbage collection can indicate a component that is making poor use of its memory allocation, forcing the JVM to collect released memory more frequently than you might prefer, and slowing the overall performance of the JVM and the components running inside it.

There's more to monitoring these components than just watching the runtime virtual machine: The best APM solutions also support *byte code instrumentation,* which is able to examine the actual code of the application for performance purposes. This literally puts your APM solution inside the application's code, giving you the deepest dive possible.

Packaged Enterprise Applications

It's common for modern applications to interact with numerous back-end systems—Siebel, PeopleSoft, Oracle E-Business, and so on. As Figure 4.12 shows, a good APM solution can recognize those dependencies and include that back-end system—SAP, in this illustration—in your monitoring efforts.

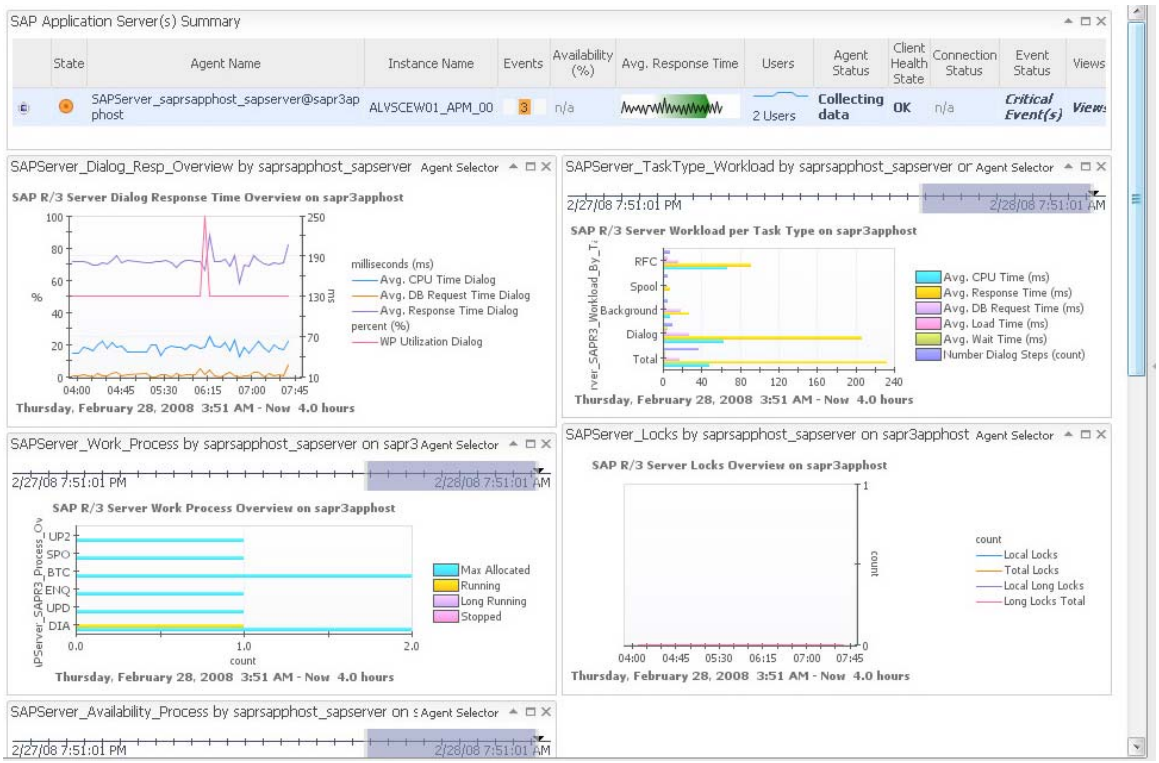

Figure 4.12: Deep-dive monitoring in SAP.

The exact metrics you seek to monitor will obviously differ from one back-end solution to another, but it's important to recognize the role these solutions can play as application dependencies, and to monitor them as if they were any other dependency in your application model. In this illustration, the APM solution is tracking overall response times for SAP access, tracking SAP's workload for various task types, and so on. Essentially, the APM solution is treating SAP as just another application component—which, in this context, is exactly what it is.

Virtual Infrastructure

As more and more of our servers move to virtualization hypervisors, the ability to directly monitor virtualization host performance—whether VMware, Hyper-V, or some other brand—becomes increasingly important. Figure 4.13 shows an example.

Figure 4.13: Monitoring the hypervisor.

This is *in addition* to monitoring the CPU, memory, and other elements *within* virtual machines; by monitoring both the physical hardware and the virtual, you'll be able to determine whether performance problems are the result of a particular virtual machine running out of resources or the result of a virtualization host that's simply overworked. Standard hardware-level measurements typically suffice for virtualization monitoring: processor, memory, and network throughput, plus, of course, storage performance, are the four key factors. Keep in mind that modern hypervisors have the ability to "overcommit," meaning the sum of each virtual machine's memory may exceed the actual physical memory available. The theory is that not all virtual machines will need their fully-configured amount of memory at all times, so the hypervisor reallocates memory on-demand; overdoing overcommit, however, can leave virtual machines straining for the resources they need, negatively impacting performance.

Network Infrastructure

It's also important to have an APM solution that can at least help in monitoring your application's underlying network infrastructure. Modern applications rely heavily on networks for inter-component communications; without a view into that infrastructure, you're missing a major performance dependency. Typically, you'll want to monitor capacity, throughput, and error rates for LAN and WAN devices, such as switches and routers. Depending on what comprises your network, specific capabilities for frame-relay networks, Quality of Service (QoS) traffic, and so on may also be important.

What to Do With Performance Data

All of the performance data gathered by an APM solution can be used for a lot more than just troubleshooting performance problems. In fact, with a good solution and the right additional capabilities, that data can help you be very proactive about managing application performance.

Continuous Monitoring

Continuously monitoring the entire application model is a key capability. You don't just fire up monitoring when something goes wrong; you want it running all the time. That continuous monitoring can net you three specific benefits, which I'll describe in each of the following three sections.

Alerting

Why wait until the EUE goes bad to start troubleshooting a performance problem? An APM can proactively alert you to *degraded* performance—potentially at the component level, before the EUE falls below your established service level. This lets you start solving a problem *before* your end users realize it's a problem.

Reporting

By collecting data over time, you can provide detailed reports on uptime, service level agreements (SLAs), and so forth. You can also use reports to provide trending information—for example, as more users utilize your application, you can make judgments on where you'll need to expand capacity to accommodate that growth.

Baselining

Baselining enables you to take a "snapshot" of what your application's performance should look like under a given load. You can then compare current performance to that baseline for growth trending and to continually examine and revise your health thresholds.

Coming Up Next…

In the next and final chapter of this book, we'll bring it all together. The first four dimensions of the 5D approach can generate a lot of data—so what do you do with it? We'll look at a case study of a slow application and explore how an APM database provides a place for all your data to reside, be correlated, and help lead to a fast resolution of performance problems. We'll wrap up with a sort of shopping list for an APM toolset, defining key selection criteria so that you can begin to intelligently evaluate APM solutions.

Download Additional Books from Realtime Nexus!

Realtime Nexus—The Digital Library provides world-class expert resources that IT professionals depend on to learn about the newest technologies. If you found this book to be informative, we encourage you to download more of our industry-leading technology books and video guides at Realtime Nexus. Please visit http://nexus.realtimepublishers.com.