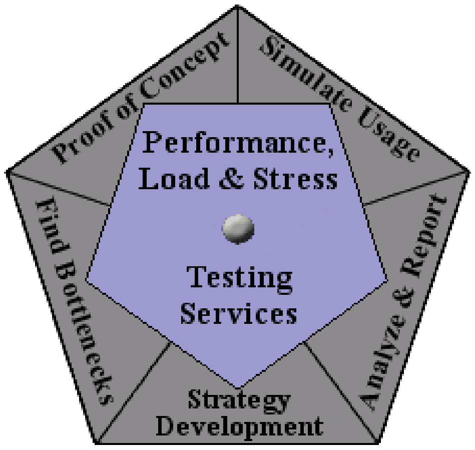
Part 9: Pinpointing the Architectural Tier of the Failure or Bottleneck

Beyond Performance Testing

by:

R. Scott Barber



By now, you've compiled a fairly substantial catalog of information about your failure, slow spot, or bottleneck, but if you're still reading, none of that information has actually told you where the bottleneck or failure lives. It's not fair to assume, for example, that the database needs to be tuned because a query returns slowly. It could be, in fact, that the code that creates the request is stuck in a near-infinite loop. You simply can't tell until you evaluate the offending activity or activities by physical or logical tier.

This is the third of four articles on the theme I call "finding bottlenecks to tune," where we're taking the step beyond just performance testing and beginning to explore how to add real value to the development team.

So far, this is what we've covered in this series:

Part 1: Introduction

Part 2: A Performance Engineering Strategy

Part 3: How Fast Is Fast Enough?

Part 4: Accounting for User Abandonment

Part 5: Determining the Root Cause of Script Failures

Part 6: Interpreting Scatter Charts

Part 7: Identifying the Critical Failure or Bottleneck

Part 8: Modifying Tests to Focus on Failure or Bottleneck Resolution

This article is intended for mid- to senior-level performance testers and members of the development team who work closely with performance testers. If you haven't read Parts 5, 6, 7, and 8 of this series, I suggest you do so before reading this article.

A Refresher on N-Tier Architecture

"All parts should go together without forcing. You must remember that the parts you are reassembling were disassembled by you. Therefore, if you can't get them together again, there must be a reason. By all means, do not use a hammer." — IBM maintenance manual, 1925

Before we can really dig into chasing bottlenecks to and into a specific tier, we should spend a few minutes reviewing some n-tier architecture basics. If you're comfortable with logical and physical architectures, feel free to skip to the next section ("Capturing Metrics by Tier").

One of the things that confused me early in my performance-testing career was the difference between the logical and the physical architecture of a system. I remember one meeting where the developers were talking about the "authentication server." I walked over to the network diagram and asked, "Which of these machines is the authentication server?" In a dismissive tone I was told, "None of them." Not easily discouraged, I asked, "Then where *is* the authentication server?" To which a developer replied, "It's on Web1 and Web4." If that response confuses you as much as it confused me at the time, the rest of this section is for you.

Logical Architecture

Architecture used to be easy. Either you had a client/server (two-tier) application or you had a Web-based application (normally three-tier). During the early days of three-tier architectures, the tiers often corresponded to physical machines (as shown in Figure 1) whose roles were defined as follows:

- Client tier (the user's machine) — Presents requested data.
- Presentation tier (the Web server) — Handles all business logic and serves data to client.
- Data storage tier (the database server) — Maintains data used by the system, typically in a relational database.

The machine that made up the presentation tier came to be known as the Web server because it ran the software used to "serve Web pages."

Figure 1: Three-tier logical architecture

At first, as architectures became more complex, individual machines were added whenever a new tier was needed. Later, tiers began to be made up of clusters of machines that served the same role. See Figure 2.

The truth of the matter is that no one actually uses the term *file storage tier*. They refer to that functionality as "the file server," for the same reason that the presentation tier became synonymous with "Web server" for Web-based applications.

The key to understanding a logical architecture is simply this: In a logical architecture, each tier contains a unique set of functionality that's logically separated from the other tiers. But even if a tier is commonly referred to with the word *server*, it's not safe to assume that every tier lives on its own dedicated machine.

Physical Architecture

So, you may ask, what does the actual physical environment look like? That's an important question when it comes to performance testing — and one that most developers and stakeholders find hard to believe matters to the performance test engineer. The paradigm that most stakeholders and developers hold to is that "testers don't need to know anything but how to access the system from the client machine." This is simply not true when it comes to performance testing. Be persistent and patient in your quest for information. Over time, they'll come to understand.

I've called this section "Physical Architecture," but that's actually one of the least-used terms for what we're talking about. Probably the most-used term is *environment* (that is, the test environment or hardware environment); it may also be called the *network architecture*. Whichever name your organization uses, what we're referring to here is represented in diagrams where actual, physical, tangible computers are shown and labeled with the roles they play and the other actual, physical, tangible computers they talk to. Figure 3 is the physical architecture of the system we looked at logically in Figure 2.

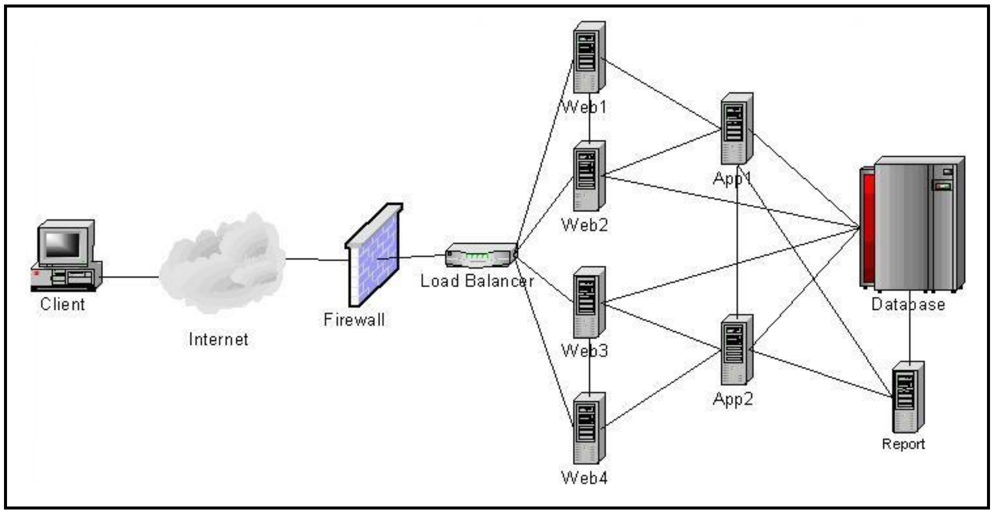


Figure 3: N-tier physical architecture

Figure 3 is very similar to the diagram I was looking at when I asked the question "Where's the authentication server?" I'm sure you now understand my confusion a little better, since there's no machine in Figure 3 labeled "Authentication Server." Instead of trying to explain verbally how the authentication server relates to the physical architecture, let me simply redraw Figure 3 with some additional labeling. See Figure 4.

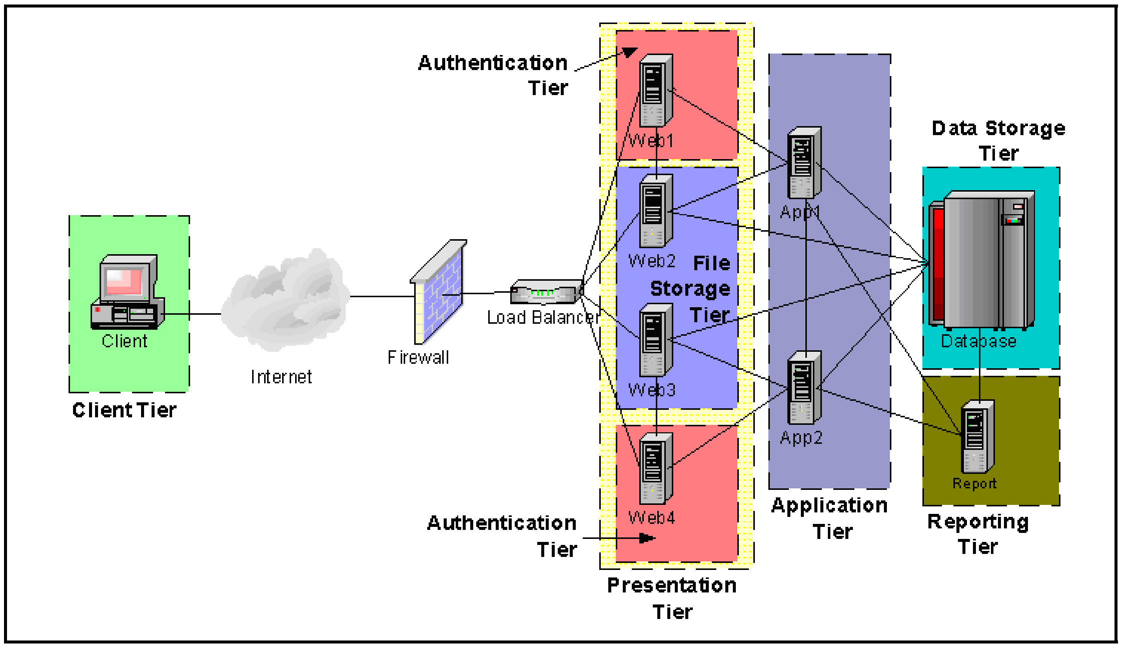


Figure 4: N-tier physical architecture with logical overlay

What we see here is that most logical tiers consist of more than one physical machine (often called clusters). We also see that the machines that make up the presentation tier (Web1 through Web4) are all serving double duty as either an authentication tier server or a file storage tier server. As it turns out, it's just about as common for a logical tier to be spread over several machines as it is for a physical machine to host the functionality of more than one logical tier.

Speaking Intelligently with Your Development Team

The purpose of this section has been to help you speak more intelligently with your development team. Our brief discussion of architecture and visual representations of architecture barely scratches the surface of what I would classify as "stuff that's useful for a performance test engineer to know about architecture." Someday I may write more articles on this topic, but in the meantime, if this is an area you feel weak in, I suggest that you ask your developers to recommend their favorite design and/or architecture books, since they're the ones you want to communicate best with. Be persistent with your questions; don't stop asking if things don't make sense. Try to use their language but don't be afraid to use different terms to clarify meaning, and when words fail, draw pictures.

Capturing Metrics by Tier

I hope you now have a solid understanding of what a tier (both logical and physical) is and can see why it's important to evaluate performance tier by tier. Through our discussions about bottlenecks it should be apparent that the end-to-end response time can never be shorter than the time spent in the slowest tier, no matter how little time is spent in the other tiers. It should also be clear that if you can't identify which tier is holding up progress, tuning becomes an exercise in trial and error. So the next question is, How do you figure out which tier is causing the issue? That's what we'll discuss here.

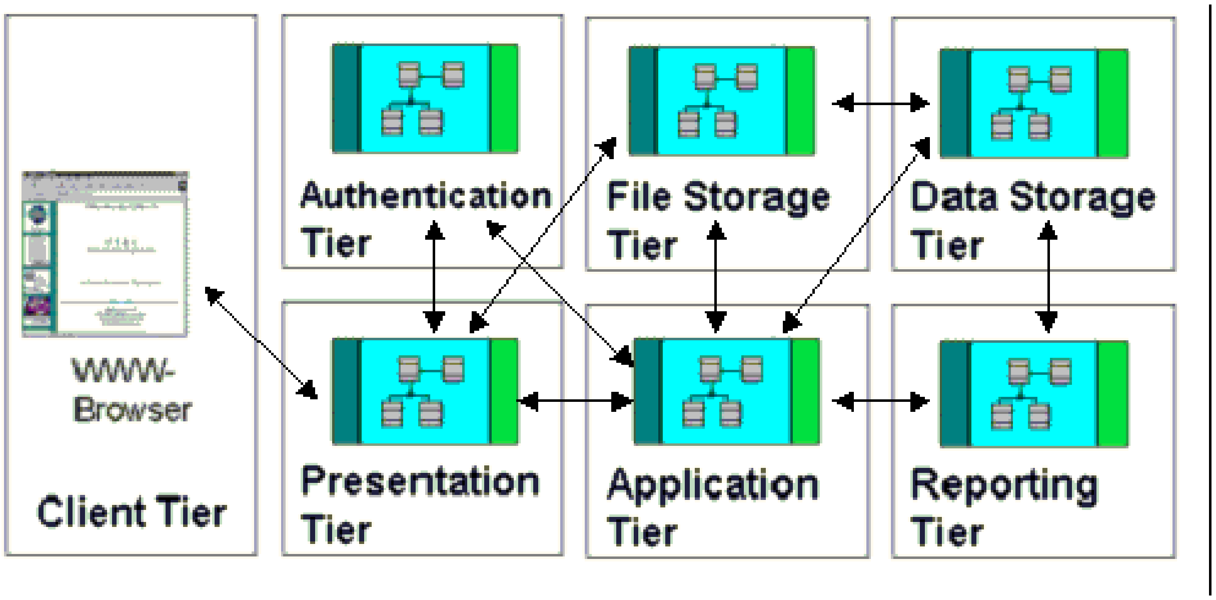
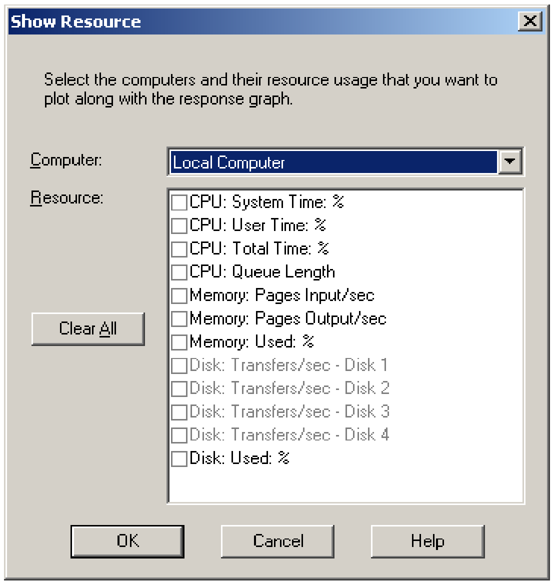


Figure 2: N-tier logical architecture

Capturing Resource Utilization by Tier with Rational TestManager

The first set of metrics we're going to capture on a tier-by-tier basis is resource utilization statistics. In Part 6 of this series I outlined the process for capturing these statistics machine by machine using the Rational® TestManager software in the section titled "Creating Overlaid Scatter Charts in TestManager." Please refer to that article for step-by-step instructions.

You need to have Rational TestAgent installed on each machine whose resources you want to monitor. Figure 5 shows the resources that can be monitored using TestManager. While these are the most commonly monitored resources, they're by no means the only ones that can be monitored.


Show Resource

Select the computers and their resource usage that you want to plot along with the response graph.

Computer: Local Computer

Resource:
- CPU: System Time: %
- CPU: User Time: %
- CPU: Total Time: %
- CPU: Queue Length
- Memory: Pages Input/sec
- Memory: Pages Output/sec
- Memory: Used: %
- Disk: Transfers/sec - Disk 1
- Disk: Transfers/sec - Disk 2
- Disk: Transfers/sec - Disk 3
- Disk: Transfers/sec - Disk 4
- Disk: Used: %

Figure 5: Options for monitoring resource utilization in TestManager

Capturing Resource Utilization by Tier with Other Methods

If you're unable to install the agent software on the machines you want to monitor, or if the resource you want to monitor isn't available in the list shown in Figure 5, you'll have to monitor resources using another method. I briefly mentioned some of those methods when I discussed the component performance chart in Part 8 of the "User Experience, Not Metrics" series. There I said that most operating systems come with resource-monitoring software, like Perfmon for Microsoft and PerfMeter for Solaris. There are countless resource-monitoring tools like these made by third-party vendors. It's usually just best to ask your developers/administrators which resource-monitoring tools they're using, and use them.

The challenge when using a resource-monitoring tool other than the one that comes with TestManager is correlating your results. There are two ways to correlate the data:

- Have someone watch the utilization rates during the test run and make note of abnormal readings along with the time they were noticed, to compare with the test log after the test.
- Have your administrators log the resource data as the test is running. Then import the data of interest from both TestManager and the resource monitor logs into a spreadsheet program like Excel, line up the start times, and create your own charts and graphs (this is how the overlaid scatter charts were created for Part 6 of this series).

Figure 6 shows a small segment of the spreadsheet table used to create the overlaid scatter charts in Part 6 of this series. This is similar to the example Excel table shown in Figure 16 in that article but is a sample from a slightly different set of measurements.

Figure 6: Sample resource utilization spreadsheet table

To populate this table requires these steps:

- Copy the data from the table in the Response vs. Time report output into Excel.
- Convert the "Ending TS" column to "Time into Test" by subtracting the value in the first row from the values in all subsequent rows in the column and then dividing by 1000 (to convert to seconds).
- Copy the time stamp and resource measurements from the third-party tool into Excel.
- Convert the time stamp to "Time into Test." This process will vary greatly based on how your tool logged the time stamp.
- Mesh the two data sets together so that the "Time into Test" values are sequential.
- Generate the desired chart based on the table.

Capturing Response Time by Tier with Rational TestStudio

One of the most common criticisms of load-generation tools is that you're unable to tell where the reported time was spent — for instance, in the database or on the Web server — without additional research. This isn't a criticism unique to the Rational® TestStudio software; none of the mainstream load-generation tools are able to tell you right at test execution which tier the time was spent in. It *is* possible to capture response times tier by tier using TestStudio, but this isn't an insignificant undertaking. Still, if you have a strong suspicion that a particular tier contains a bottleneck or are confident that you want to isolate your load tests to a specific tier, you'll want to do it.
Because this method is so specific to the environment you're testing and the tier you want to isolate, I'll illustrate by example rather than trying to come up with a list of "if-then" rules that would be sure to miss some quirk of the application you're actually testing. Let's assume, then, that we have a system with a simple architecture where each physical machine represents a logical tier, and there's a single load-generation machine (master station) as shown in Figure 7.

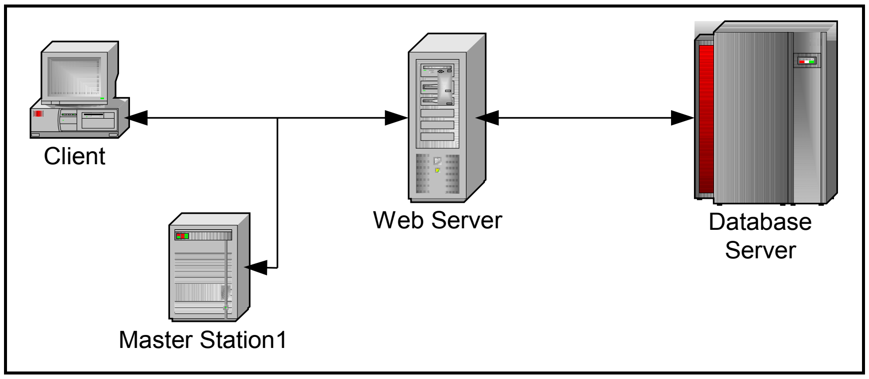


Figure 7: Basic load-testing environment

Now, let's further assume that through testing we've determined that only transactions that interact with the database cause symptoms of poor performance. We've further established that these aren't failures, the symptoms span multiple activities, and the entire system is affected by the symptoms. By monitoring resources, we've found that the database often shows 100% CPU utilization and runs out of memory, and that there's often a queue of requests to enter the database under loads significantly below the target load.

Based on this, we decide with our team that we want to test just the database server under load and eliminate the Web server response times from the equation. There are actually two ways to do this using Rational TestStudio. The first way involves either building a test harness to access the database or writing custom scripts by hand (that is, not using recording) to send SQL commands to the database. While the latter is possible, it's rarely done, due largely to the time and energy required for such an effort. If we decide to go this route, we'll want to configure our environment simply, as shown in Figure 8.

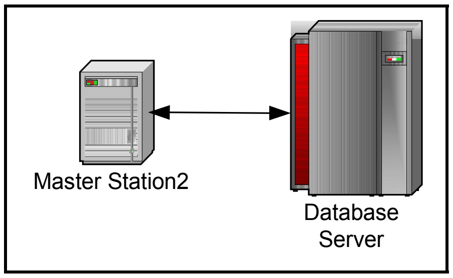


Figure 8: Data storage tier isolated for a load test

A second way is to use Master Station2 to capture the traffic against the database generated by the load test as it's being executed by Master Station1 (see Figure 9). This does require additional licenses but will give us a recorded script to edit that contains the entire load being placed on the database in a way that's easy to play back and evaluate. In this case, every command ID will represent a request sent to the database by the Web server. Executing these scripts and reviewing the response times for these command IDs will show us conclusively how much of the end-to-end test time is being spent in the database. It will also show us exactly how much time each request takes. This information is almost always what the database administrator needs in order to find and/or tune the issue.

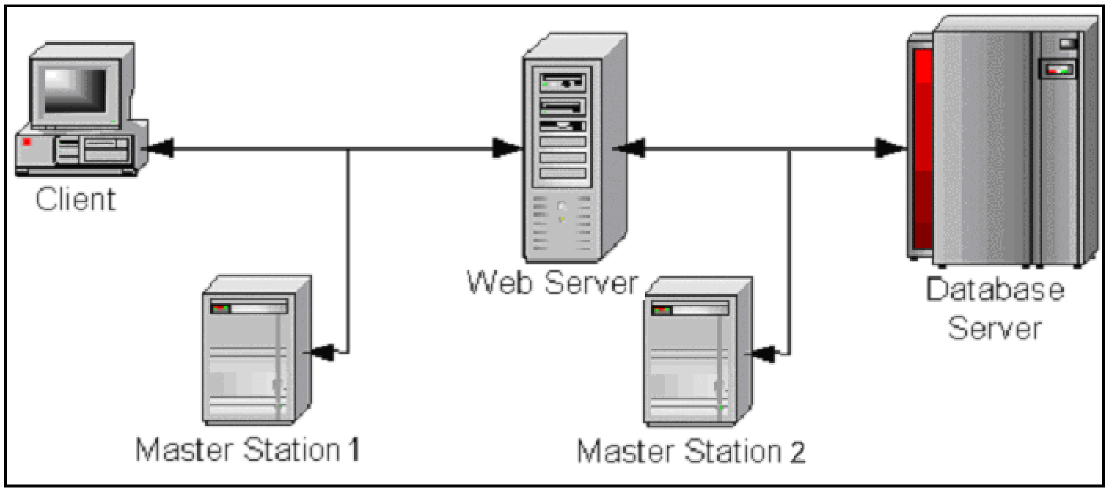
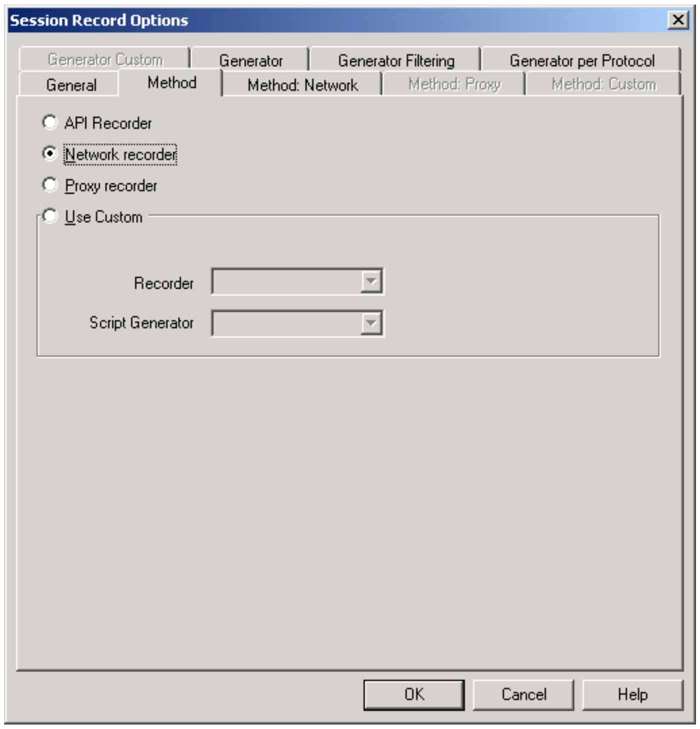


Figure 9: Capturing load-test traffic against the data storage tier

Here are the steps we would follow to capture load-test traffic against the database server in our example scenario:

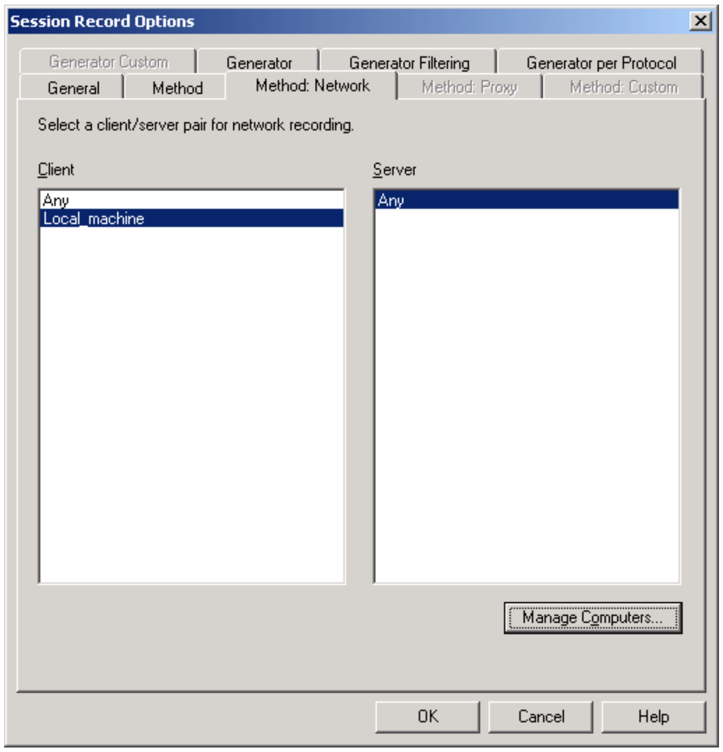
1. Configure a second master station on the same subnet as either the Web server or the database server (in this case, the database server subnet is preferred).
2. Configure the second master station for network recording between the Web server and the database server.
3. From the Robot menu bar, choose Tools > Session Record Options. Click the Method tab and select "Network recorder" (see Figure 10).



Selected setting: Network recorder

Figure 10: Session Record Options window, Method tab

4. Click the Method: Network tab and then click the Manage Computers button (see Figure 11).



Select a client/server pair for network recording.

Client: Any, Local_machine (selected)

Server: Any (selected)

Figure 11: Session Record Options window, Method: Network tab

5. In the Manage Computers window, click New (see Figure 12).

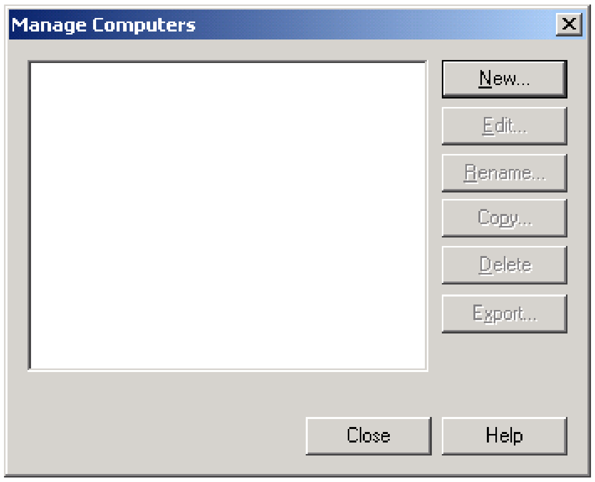
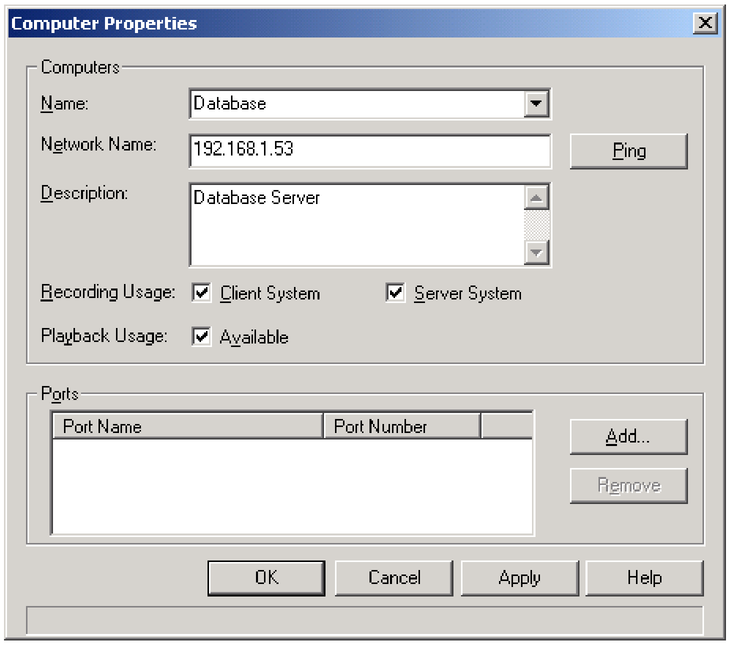


Figure 12: Manage Computers window

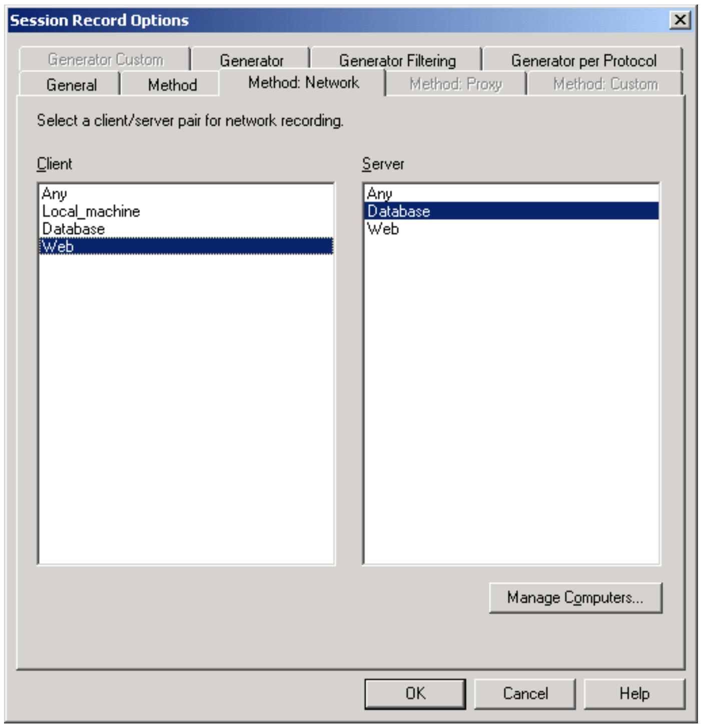
6. In the Computer Properties window, fill out the information about the database server (see Figure 13). You'll likely need to get this information from a systems administrator. The name is any name you assign; the network name is the actual machine name or IP. Click the Ping button to ensure the master station can communicate with the server, then click OK.



Name: Database
Network Name: 192.168.1.53
Description: Database Server
Recording Usage: [✓ Client System] [✓ Server System]
Playback Usage: [✓ Available]

Figure 13: Computer Properties window

7. Follow steps 4, 5, and 6 for the Web server.
8. Return to the Method: Network tab and select the database server as the server machine and the Web server as the client machine (see Figure 14), then click OK.


Session Record Options

Select a client/server pair for network recording.

Client: Any, Local_machine, Database, Web (selected)

Server: Any, Database (selected), Web

Figure 14: Method: Network tab with client and server options selected

9. Ensure that no other people or systems are accessing the database and that you have the correct protocols selected, then start recording on Master Station2.
10. Launch the load test containing the transactions you want to capture on Master Station1.
11. Stop recording after the load test has completed executing on Master Station1.
12. View, edit, play back, and analyze the new script against the database server.

While the steps seem straightforward enough, this method is actually pretty complicated. Some things to remember before and while attempting this method are as follows:

- You'll have to use network or proxy recording on Master Station2. Network is generally easier, if the tiers you're interested in are on the same subnet and on a Rational-supported network configuration. Proxy recording is generally considered to be difficult to configure.
- You may not have the proper license for the communication protocol between the tiers you want to isolate, which means you'll have to either interpret socket traffic or obtain the proper protocol license.
- Editing these scripts, even with a supported protocol, is often complicated because you might have no realistic way of knowing which client-side activity generated a request. For example, I once tested an application where each client-side activity was generating two identical database requests. This type of testing didn't help to track that down.
- You'll be collecting response times for only one tier per test; you won't be collecting the response times for each tier during the same test.

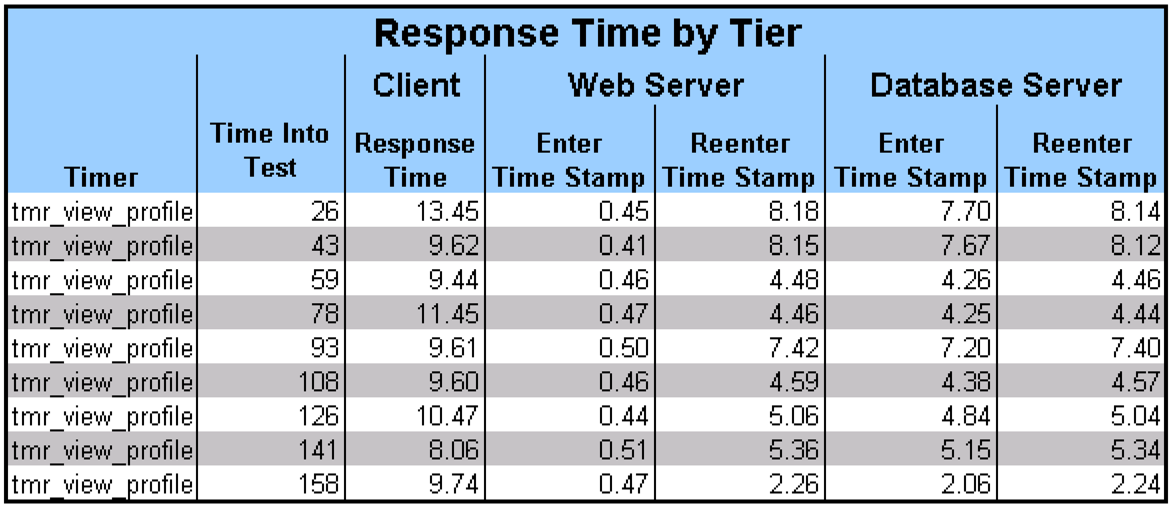
Capturing Response Time by Tier with Other Methods

There are several other ways to capture response time by tier, but they all involve either third-party tools or for your system to be instrumented to collect (log) data. If you don't already have a third-party tool, I highly recommend that you get one as a complement to TestStudio; however, I feel compelled to caution you that these tools are generally very specific to your application architecture. For instance, some of these tools will collect information only on J2EE applications, others only on .NET platforms. I suggest you do a Web search on the phrases "performance monitoring," "application performance management," "performance analysis tool," and/or "performance profiler" and compare the tools available to your specific application.

If you don't have, and won't be getting, any third-party tool to complement TestStudio, there's one other way to capture response time by tier. This method involves close coordination with your developers and administrators and is also very specific to your application. The basic steps are as follows:

- Identify the tier(s) you want to capture response times for.
- Work with your developers and administrators to configure logging on those identified tiers to capture the time stamp of the arrival and/or departure of the transaction(s) you're interested in.
- Ensure that all computers in the system *and* the load-generation machines have their clocks synchronized.
- Execute the load test.
- Parse the arrival and departure time stamps from the log files.
- Correlate those time stamps with the end-to-end response times from TestManager using a spreadsheet program like Excel.
- Convert those time stamps to response times — generally by averages — and put them into charts and graphs.

This isn't a simple process, but it does provide lots of useful information. I'll show you a sample table and graph I created using this method (see Figures 15 and 16).



Response Time by Tier

		Client	Web Server		Database Server	
Timer	Time Into Test	Response Time	Enter Time Stamp	Reenter Time Stamp	Enter Time Stamp	Reenter Time Stamp
tmr_view_profile	26	13.45	0.45	8.18	7.70	8.14
tmr_view_profile	43	9.62	0.41	8.15	7.67	8.12
tmr_view_profile	59	9.44	0.46	4.48	4.26	4.46
tmr_view_profile	78	11.45	0.47	4.46	4.25	4.44
tmr_view_profile	93	9.61	0.50	7.42	7.20	7.40
tmr_view_profile	108	9.60	0.46	4.59	4.38	4.57
tmr_view_profile	126	10.47	0.44	5.06	4.84	5.04
tmr_view_profile	141	8.06	0.51	5.36	5.15	5.34
tmr_view_profile	158	9.74	0.47	2.26	2.06	2.24

Figure 15: Response-time-by-tier base table

Figure 15 is a spreadsheet table containing the base data needed to extract a response-time-by-tier graph. The basic steps to create this table are as follows:

- Copy the data from the table in the Response vs. Time report output into Excel.
- Convert the "Ending TS" column to "Time into Test" by subtracting the value in the first row from the values in all subsequent rows in the column and then dividing by 1000 (to convert to seconds).
- Copy the time stamps and labels from the log files into Excel. This process will vary greatly based on how your tool logged the time stamp.
- Convert the time stamp to "Time into Test." This process will vary greatly based on how your tool logged the time stamp.
- Subtract the "Time into Test" value from the time stamp.
- Mesh the two data sets together so that the "Time into Test" values are sequential and grouped by timer/label.
- Generate the desired chart or graph based on the table.

Figure 16 is the graph created from the averages of the values in the table in Figure 15.

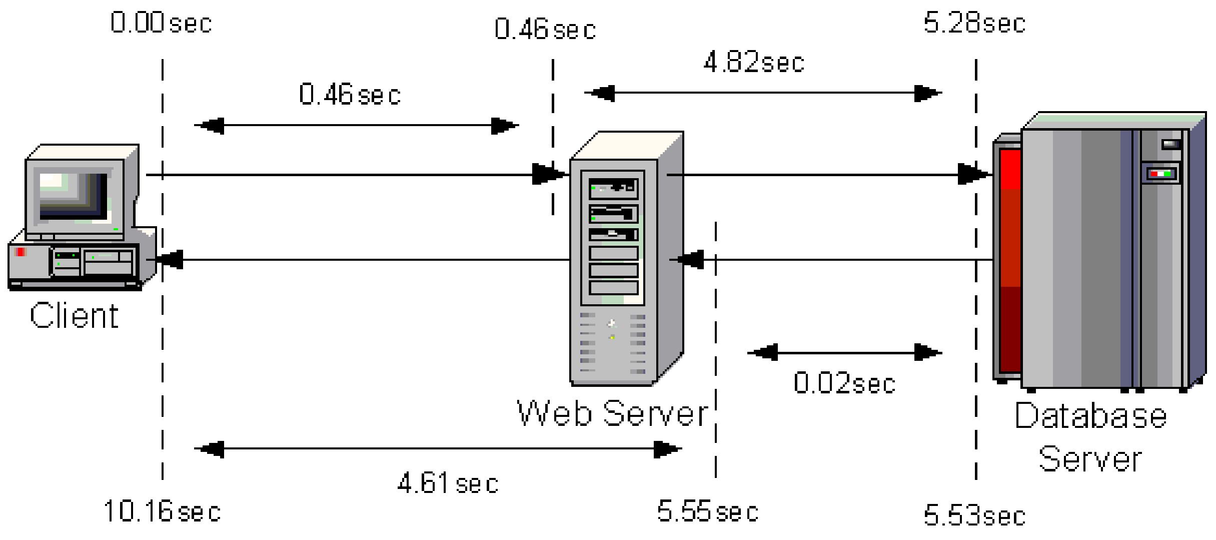


Figure 16: Response-time-by-tier graph

If you look closely at Figure 16, it becomes clear that about 8 seconds of the 10.16-second response time is being "lost" in the Web server (4 seconds each way). In this case, further research showed that the actual problem was a misconfigured router on the Web server's subnet that was imposing an artificial 4-second delay.

A Note About Performance-Monitoring Tools

As I mentioned above, there are hundreds of third-party tools available to assist with the capture of resource utilization statistics and response time by tier. There are fewer, but still many, third-party tools that provide a combination of these functions. These are commonly known as performance-monitoring tools. As an example of the kinds of tools available, let's take a look at IBM's Tivoli systems and applications monitoring solutions. If you aren't familiar with this family of solutions, it's officially described this way:

"IBM Tivoli systems and applications monitoring solutions enable you to deploy a single monitoring solution for most or all of the resources in your environment. This allows your system monitors to share a common reporting engine, graphical user interface, and data repository. Building on the science of Autonomic computing, the IBM Tivoli suite of monitoring tools gives you several new capabilities such as the ability to automatically correct many component-level problems before they occur. It can also identify the persistence of problem conditions and feed key operating metrics into other layers of your management technology system."

Tivoli, like almost all of the other performance-monitoring solutions, doesn't deliver with a robust load-generation component, but using it with a load generator like TestStudio greatly enhances the performance-engineering process. The white paper titled "IBM Tivoli Monitoring Solutions for Performance and Availability"does a good job of explaining some of what this particular solution has to offer. It's beyond the scope of this article to go into details about what Tivoli, or any other performance-monitoring solution, can add to your performance-engineering process, but I encourage you and your development team to jointly research a performance-monitoring tool that fits your needs. If your organization conducts performance-engineering exercises often, the time you'll save by obtaining and using one of these tools will far outweigh the cost of the tool in a short period of time.

Interpreting Tier-Specific Metrics

Often, tier-specific metrics leave little doubt as to their meaning, but as you saw in the "response time by tier" example, even these detailed metrics may not hold all of the answers. Now I'll share the methods I've found most useful, individually and collectively, to interpret tier-level metrics.

Look for the Obvious

First and foremost, look for the obvious. In our "response time by tier" example, the obvious was that the Web server (presentation tier, more precisely) was eating up 4 seconds every time data passed through it. In the overlaid scatter chart shown in Figure 20 in Part 6 of this series, it was obvious that the CPU utilization of the application server peaked above acceptable levels shortly before the poor performance began. These are the kinds of clues we're looking for. Unfortunately, in both cases, these were still both symptoms and not causes. In both cases, those symptoms gave me and the development team ideas about where to look next.

Consult the Development Team

Once you either find some obvious abnormalities, symptoms, or clues *or* realize that you haven't found any obvious abnormalities, symptoms, or clues, you should contact your development team and discuss what those findings mean. If you found no clues, maybe it means that the metrics you collected weren't the right ones, or that there wasn't enough load on the system, or that you eliminated a trigger event when you modified your tests. You probably won't know which (if any) of these is the case without help from your development team.

In the cases where you *do* find clues, the development team definitely wants to be involved. These clues are what point to either the next round of tests or to what they'll find themselves tuning in the next hours, days, or weeks.

The point is, when you get this far into your performance testing, you and the development team really form a consolidated performance-testing-and-tuning team. Most development teams aren't used to working like this, so it's up to you to be the team leader and ensure that there's constant two-way communication about tests, results, clues, and suspicions. More than half the time, I find that I'm able to track down a bottleneck not by my keen insight or superior testing knowledge, but rather by listening to developers when they say things like "I wonder if . . . ," "Did you try . . . ?" or "What if we . . . ?" You'll also often find that after you show the results to your development team, one of them will come back to you later and say, "I found it," when you didn't even know he or she was looking for it.

Change Your Test to Prove Your Theory

Once you see your tier-specific results, it'll be almost impossible for you not to form theories about what caused those results. This would be like reading a mystery novel and not trying to guess "Who done it?" before the final chapter — you just can't do it. Instead of trying to wait for the last chapter, I recommend embracing those theories and changing your test to prove or disprove them immediately. Once again, you'll likely need the assistance of the development team, but by this point you should have a good working relationship with them. Besides, most developers I've worked with really enjoy this part of the performance-testing process. Honestly, I have to agree with them. To me this is the fun part; it offers the same excitement a treasure hunt did when I was a kid . . . "I wonder what we'll find if we follow all the clues correctly?!?"

Is It Time to Tune?

Once again, we come to the key question, "Is it time to tune?" By now, we'll have successfully pinpointed the bottleneck about 75% of the time. If you and the development team haven't gathered enough information at this point to be able to tune the system, you have a pretty elusive bottleneck. You may have noticed that in some of the examples we've been following, we aren't ready to tune. In these cases we still have to develop specific tests to exploit the symptoms before we can resolve them. That's our topic in Part 10.

Summing It Up

In this article we looked at some ways to isolate symptoms and metrics by logical and/or physical tier of the system. This process isn't always easy, but it does add a significant amount of information to what we already know about our bottleneck symptoms. It's critical to build a close relationship with your development team as you dive deeper and deeper into the application, if you haven't already. They'll be your best tool for collecting information about, and ultimately finding and tuning, performance bottlenecks.

Acknowledgments

- The original version of this article was written on commission for IBM Rational and can be found on the IBM DeveloperWorks web site

About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.