

PerformaSureTM

Demo Guide

July 2003

J2EE performance diagnosis



Copyright© 2003 Quest Software Inc.

Note: Quest, the Quest Software logo, Quest Central, PerformaSure and JProbe are trademarks or registered trademarks of Quest Software Inc. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other products are trademarks or registered trademarks of their respective companies.

Contents

Introduction	1
PermaSure Overview	1
Key Features	2
Runs on Production or Pre-Production Test Systems	2
Component-level Instrumentation with Detail Dial	2
Transaction Tag and Follow with Tree View	2
Power Diagnostics GUI	2
Server-by-server Response-time Breakdowns	2
Reporting and Performance Data Export	2
Why PermaSure?	3
PermaSure Addresses J2EE Challenges	3
Shrinking Development Timelines and Iterative Deployment Cycles	3
The Increasing Complexity of Infrastructures	3
Guesswork and Finger Pointing	3
PermaSure is Designed for J2EE Problem Solving	4
PermaSure Provides Visibility Into the Entire J2EE Application	5
PermaSure Advantages.....	5
Transaction-Centric Complete J2EE Application Diagnostics	5
Comprehensive System-wide View	6
Low, Configurable Overhead	6
Collaboration Across Teams	6
Integration with Quest's Detailed Diagnosis Tools	6
PermaSure Supported Platforms and Requirements.....	7
Platform and Environment Support	7
Operating Systems	7
Application Servers	7
HTTP Servers	7
Database Servers	7
Quest Tool Integration	7
PermaSure System Requirements	8
Server	8
Workstation	8
Getting Started with the PermaSure Demo	8
Installing the PermaSure Demo Version	8
Launching the PermaSure Demo Version	9
PermaSure – A First Look	10
Key Components	11
PermaSure Agents	11
PermaSure Nexus	11
PermaSure Workstation	11
The PermaSure Approach to J2EE Diagnosis	11
Introducing PermaSure's Viewer Windows	12
Request Time View	12
Request Tree View	13
Threshold View	14
Metrics View	15
Network Traffic View	16
Diagnosing Example Problems with PermaSure.....	17
Start Workstation, Open Project and See Three Problems	18
Problem 1: Inefficient Java code	19
Problem 2: Misuse of database connection pool	21
Problem 3: EJB Passivation from Metrics evidence	23

For More Information.....	28
About Quest Software	28
Java/J2EE Products Available from Quest Software	29

Introduction

J2EE-based systems are now powering many key IT and customer-facing business systems. But beneath the convenient abstractions of the Java 2 Enterprise Edition (J2EE) architecture lies considerable complexity that can lead to unforeseen performance problems when an application reaches the staging or live production arena. The complexity of the J2EE technology stack makes it difficult to track down the source of performance problems – they may reside in the application code, the database, the configuration of application servers and clusters, operating system, hardware, or in interaction between any combination of these components. It's critical to solve these performance issues as quickly as possible.

That's where PerformaSure comes in. With PerformaSure, companies can diagnose any type of J2EE performance issue and determine its root cause quickly. A powerful, transaction-centric diagnosis tool for distributed J2EE applications, PerformaSure enables companies to diagnose and resolve performance problems across the application servers, databases, and code of a J2EE application.

This Guide provides background information on PerformaSure, an introduction to its capabilities, architectural overview, and an introduction to its user interface. The guide also provides a walkthrough of using PerformaSure to diagnose several “canned” performance problems. If you received the demonstration edition of PerformaSure, you can run through this walkthrough directly, using the product itself.

PerformaSure Overview

Offering a transaction-centric view of performance, PerformaSure is a low-overhead performance diagnosis tool for multi-tiered J2EE applications running under load in test or production environments. PerformaSure enables application administrators, developers, system architects, database administrators and quality assurance staff to measure, analyze and maximize round-trip transaction performance and minimize infrastructure costs.

PerformaSure's unique Tag and Follow technology and “power diagnostics” GUI allows for a unified, graphical view of round-trip transaction performance. By tracing transactions through the J2EE system and reconstructing the execution path, PerformaSure provides broad and deep visibility into the J2EE performance data stack, focusing on end-user response times as well as method-level timing.

Key Features

Runs on Production or Pre-Production Test Systems

Ultra low-overhead agents with automatic sampling enable PerformaSure to run on production systems, as well as test environments. When a live performance problem is detected, operators can collect high-level performance data while incurring minimal overhead (as little as 5-10%) on the live system.

Component-level Instrumentation with Detail Dial

High-level component performance view makes initial investigation easier. Configure the tradeoff between the level of data collected and the associated overhead with the Detail Dial, enabling application administrators to quickly identify the bottlenecked component for deeper analysis by functional experts.

Transaction Tag and Follow with Tree View

PerformaSure's unique Tag and Follow technology provides a transaction-centric view of performance data. Tag and Follow reconstructs the end-to-end execution path of individual user transactions. All physical servers, logical software components and methods are shown in a color-coded Request Tree View to instantly identify bottleneck components.

Power Diagnostics GUI

The PerformaSure Workstation is designed to speed navigation and diagnosis of complex J2EE applications. The intuitive "Zonar" time-navigation feature facilitates investigation by allowing users to quickly fast-forward, rewind or zoom into problematic time periods. Integrated Metrics viewers provide system, app server, and OS metrics across all tiers that are correlated with end-user response times.

Server-by-server Response-time Breakdowns

PerformaSure provides objective data on which end-user transactions are slowest and which server(s) is responsible for these unacceptable response times. Database operations are instrumented to show time spent in each JDBC call.

Reporting and Performance Data Export

Generate detailed performance analysis reports in Acrobat PDF, XML or Excel-friendly CSV format to easily show management and team members the source of any J2EE performance problem. Compare and trend performance metrics over time.

Why PerformaSure?

PerformaSure Addresses J2EE Challenges

PerformaSure helps IT, development and QA groups solve performance problems as quickly as possible in production or test environments.

Shrinking Development Timelines and Iterative Deployment Cycles

Development timelines are always rushed—the business always needs the application in production “yesterday”. IT groups are experiencing a push from upper management to better utilize existing systems—and less money for hardware and software is available. Infrastructure costs are high to begin with, particularly in hardware, database and application server licenses.

PerformaSure fits this model perfectly. Companies can easily integrate PerformaSure into the iterative testing, staging and release lifecycle, as well as use it to quickly diagnose unexpected problems in production. PerformaSure reduces the need to purchase costly hardware or software to solve the problem.

The Increasing Complexity of Infrastructures

As application servers provide more and more services, they necessarily become more complex, resulting in a greater number of application server resources and settings to properly configure and tune. And as systems become more distributed, there are still more factors to consider: network efficiency, clustering setup, third-party systems and interactions between (possibly remote) components. And finally, all this has to work perfectly for users nearby and across the globe.

With PerformaSure, performance teams are not only able to tag and follow transactions through the entire system and back again, they see how the application interacts with all the components in the system. PerformaSure is designed for rapid and accurate investigation of the entire system, starting from the perspective of end-user transactions.

Guesswork and Finger Pointing

Deciding where to start to fix poor performance can explode project timelines. The inherent complexity of the underlying J2EE system makes it easy for IT groups to indulge in finger pointing when it comes to figuring out who should fix a problem. Traditionally, system administrators had operating system monitors, developers had profilers and debuggers, database administrators had database analysis tools, network administrators had network sniffers and so on. But it’s difficult to solve performance problems in

disparate systems with a collection of point tools that don't work together. As a result, everyone is quick to point the blame elsewhere.

IT groups need a diagnosis solution that's designed for J2EE, one that bridges the gap between black-box load testers (which do not provide application-specific performance timing information) and detailed point products (such as code profilers and network monitors). They need a solution that allows users to quickly identify problematic J2EE components (servers, clusters, databases, network), especially from the perspective of end-user transaction response times. IT groups must be able to identify which tier, server and component is causing the problem in order to clearly determine who is responsible for fixing it. In other words, a tool like PerformaSure.

PerformaSure is Designed for J2EE Problem Solving

Developers write and test application code in an environment quite unlike that in which the application will eventually be deployed—away from the complications of realistic databases, networks, clusters and interactions between the application components that other team members have developed. When application components are assembled in pre-production and even deployed to production, these complicated factors can lead to dramatic and unanticipated performance problems. Teams are discovering that algorithmic issues, database interaction and network efficiency problems that appear under full scale user load on the distributed system are among the first problems they need to resolve. Most diagnostic solutions are either too high in overhead (profilers) or too light on detail (load testers) to provide a solution for J2EE.

PerformaSure is designed for problem-solving in full scale J2EE environments, in both staging and production. In either environment, PerformaSure captures performance data as the system works under load. Its unique Tag and Follow technology and power diagnostics GUI makes sense of the reams of metrics data to allow teams to trace a bottleneck to its root cause, quickly and with certainty. PerformaSure helps performance analysts identify the following kinds of problems in single-server or clustered, distributed systems:

- Inefficient servlets, JSPs, EJBs, Java classes and methods
- Slow running SQL statements
- Inefficient EJB/DB interaction
- Application server configuration and deployment settings
- Remote method invocation (RMI) and object serialization problems
- Operating system parameters and settings
- Limiting hardware components
- Poorly performing third-party components
- JVM heap usage and configuration problems

- Load balancing problems
- Cluster configuration and performance problems
- Excessive network traffic

PerformaSure Provides Visibility Into the Entire J2EE Application

The big question in diagnosing J2EE performance issues is what component or interaction between components is the source of the bottleneck. Most tools do not provide detailed application-specific information. Still fewer tools provide visibility into all of the components of a J2EE application, including issues relating to application infrastructure configuration and JDBC/database interaction. What are the real limits to performance and scalability? Where are transactions spending their time? Which components are misusing the database or the network and why? How is the application server or operation system configuration impacting performance?

With PerformaSure, performance teams have visibility into the entire distributed application and its interaction with all the components of a distributed J2EE system, including Java components (EJBs, servlets, JSPs, Java classes), memory usage, network usage, application server configuration and tuning, and database interaction. The key to most performance problems lies in the application's interactions with each system element, so this is a critical capability of PerformaSure.

PerformaSure Advantages

PerformaSure fills an important and unsatisfied need by providing deep application diagnostics from a transactional, system-wide perspective for both test and production systems. PerformaSure gives performance teams objective data, allowing them to collaborate in problem solving for even faster time-to-repair. And PerformaSure is part of Quest Software's complete solution for J2EE application performance management.

Transaction-Centric Complete J2EE Application Diagnostics

Taking a transaction-centric approach is the fastest way to identify the most important bottlenecks – those that affect end-users. PerformaSure was designed for transaction-centric investigation and diagnosis. With its unique Tag and Follow technology, PerformaSure breaks down transaction timing by component, server or cluster to show an end-user point of view. PerformaSure collects and correlates a full complement of vital performance metrics for breadth and depth of J2EE diagnosis. As well, PerformaSure lets analysts drill down to the transaction and method in question to find the root cause.

Comprehensive System-wide View

PerformaSure provides a rich graphical user interface and views that guide users through all performance data quickly and intuitively. In addition, powerful time navigation with PerformaSure's Zonar technology allows analysts to fast-forward and rewind through, as well as zoom in and out of, all collected performance data.

Low, Configurable Overhead

PerformaSure allows users to select a specific transaction to tag and follow for a more focused low-overhead investigation. Configurable investigation options and flexible settings for the non-intrusive Agents installed on each component mean it's easy to target investigation without changing or recompiling code.

Collaboration Across Teams

The comprehensive system-wide view, intuitive interface and powerful navigation make PerformaSure a natural catalyst for sharing diagnostic intelligence between teams in an IT organization. What better way to demonstrate that a particular performance problem originates in database calls than to show a developer and a database administrator the specific transaction, where it's slowing down, and why? Guesswork and finger pointing, the symptoms of a lack of comprehensive performance data, are no longer necessary in the presence of hard facts. The right expert can get to work and solve the problem more quickly than it usually takes to begin a dispute. With PerformaSure, all members of a performance team have the data they need to swiftly fix a problem and get back to work. As well, geographically dispersed teams can access PerformaSure Workstations from remote locations, allowing them to view performance data concurrently.

Integration with Quest's Detailed Diagnosis Tools

Detecting, diagnosing and resolving J2EE performance problems can be too big a job for one expert or one tool. Detecting problems in production isn't enough if you can't automatically gather performance data from the live system. And often the culprit behind performance problems lies outside or deep within the J2EE application components – such as deep within the application code itself, or in complex database queries.

PerformaSure functions as part of an integrated J2EE application performance management solution, made up of Foglight, PerformaSure and JProbe. Foglight monitors J2EE applications in production 24x7; when a problem surfaces, it can seamlessly trigger PerformaSure to capture diagnostic data from the live system, allowing production staff to move rapidly into the diagnostics phase of the repair cycle. When the application code is

at fault, PerformaSure hands off to JProbe to zoom in on deep performance, memory, and threading problems. PerformaSure can also launch Quest's database analysis tools for Oracle and DB2 to diagnose complex database issues.

PerformaSure Supported Platforms and Requirements

Platform and Environment Support

Operating Systems

- Sun Solaris
- Windows 2000, XP
- IBM AIX
- HP-UX
- Linux

Application Servers

- BEA WebLogic Server
- IBM WebSphere Application Server

HTTP Servers

- Sun ONE HTTP Server
- Apache HTTP Server
- Microsoft IIS
- BEA WebLogic HTTP Server
- IBM WebSphere HTTP Server

Database Servers

- Any JDBC-compliant RDBMS, including Oracle, IBM DB2, MS SQLServer and Sybase

Quest Tool Integration

- Foglight with J2EE cartridge
- JProbe Suite
- Spotlight on Oracle
- Spotlight on DB2
- SQLab Vision for Oracle
- SQL Tuning for DB2

For the latest details on supported platforms, visit <http://java.quest.com/performasure>.

PerformaSure System Requirements

Server

- 750 MHz class processor
- 1 GB RAM
- 10 GB disk for data storage
- 100 Mbps network adapter
- Sun JDK 1.3.x or 1.4

Workstation

- 750 MHz class processor
- 256 MB RAM
- 500 MB disk
- Sun JDK 1.3.x or 1.4

Getting Started with the PerformaSure Demo

You may have received the demonstration version of PerformaSure with this document. The demo version is identical to the standard release of PerformaSure except that it has been packaged for easier evaluation. Normally, PerformaSure's server-based components need to be installed and configured on every machine running components of the J2EE system, and then a performance data session must be captured while the J2EE system itself runs under user load. The demo version skips these requirements by including a pre-recorded data session and project for you to work with.

Installing the PerformaSure Demo Version

The installation program sets up the PerformaSure client, the server-side components, and the pre-recorded data session on your computer. Perform the following steps to install the PerformaSure demo on your computer:

1. Launch the PerformaSure installer.
If you received the Demo on CD-ROM, insert it into the CD drive on your computer and the installer should launch automatically. If it doesn't, launch the installer manually by locating and executing the `demo_installer` file on your computer.
2. Proceed through the installer panels to accept the license agreement, choose the installation directory and specify other options.

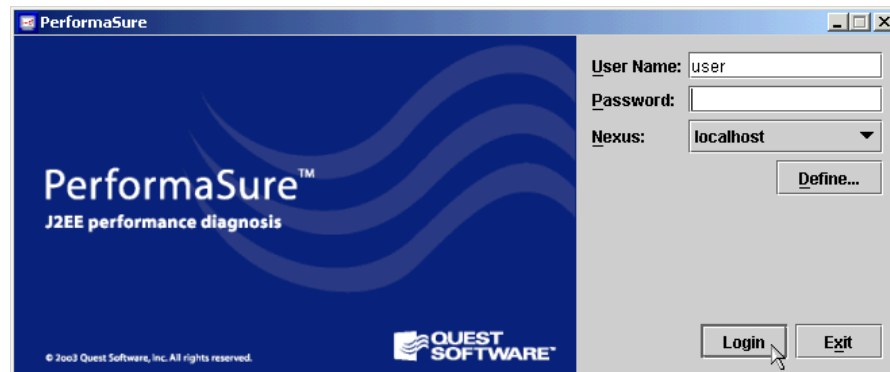
3. Click Finish on the final installer panel to begin the installation.
4. After copying files, a licensing panel appears. PerformaSure will not run unless it has been enabled for your machine. If you received a license file from a Quest representative, you can install it now (click **Add License File** and select the performasure.license file you received). If you do not have a license file you can finish the installation without it and enable the product manually before running it.

When the installer has finished, a PerformaSure Demo program group can be found in the Start menu, containing shortcuts to launch the demo version, run the license manager, and view the online product documentation.

Note: If you did not receive a license file, please contact your Quest representative to obtain one. When you have received it, launch the License Manager (from the PerformaSure program group). Click the **Add License File** button and select the performasure.license file you received; PerformaSure should now be enabled.

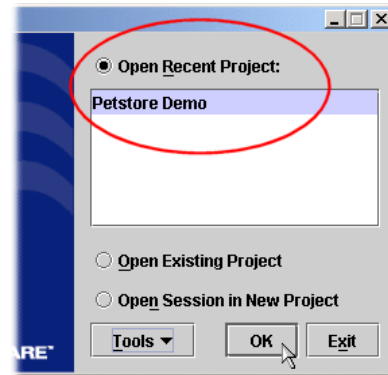
Launching the PerformaSure Demo Version

1. Start the PerformaSure Demo launcher (for example, **Start > Programs > PerformaSure Demo > PerformaSure Demo**).
This launcher starts the PerformaSure Nexus and the PerformaSure Workstation user interface. A console window should open and display several launch and initialization messages. Finally, a Login dialog will appear.

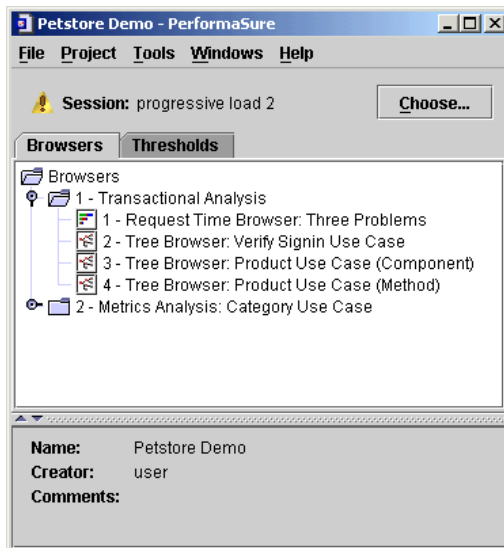


2. Use the default user name (“user”) and password (blank); click the **Login** button to proceed.

- Once logged in, PerformaSure will ask you to open a previous project or start a new project using a performance data session. Select the “Petstore Demo” sample project and click **OK**.



The Project window opens, displaying a list of views. These views are actually starting points for finding three J2EE performance problems in this application; we'll cover them later. PerformaSure is now ready for use.



PerformaSure – A First Look

PerformaSure collects correlated performance metrics for each service request received and processed by a J2EE application. Beginning with the initial HTTP request, PerformaSure follows the path of execution through web servers, application servers, and the database. Collecting performance data is a wholly separate step from analyzing that data to diagnose the root cause of a performance issue.

Key Components

PerformaSure is made up of three main components: Agents, the Nexus and Workstations.

PerformaSure Agents

Highly configurable, lightweight PerformaSure Agents collect vital metrics and timing information, such as method-level timings and system and application server metrics, using Quest Software's unique Tag and Follow technology. HTTP, application server, and/or operating system Agents are installed on each server and tier in the J2EE system.

PerformaSure Nexus

The PerformaSure Nexus is the central hub of data processing, receiving messages from all Agents, performing time correlation and synchronization and writing data to a highly-specialized data store. The Transaction Correlation Engine within the Nexus uses unique technology and method-level timing information from the Agents to reconstruct the end-to-end execution path of end-user transactions.

PerformaSure Workstation

The PerformaSure Workstation is a rich, intuitive graphical user interface for investigation of collected performance data. Designed for problem diagnosis and resolution, the interface provides powerful time and data navigation controls for quicker time-to-repair. The ease with which a user can navigate through data in the Workstation is one of PerformaSure's distinguishing factors; only with this rich user interface can problems be easily identified. Each team member can run the Workstation on their local PC and access one session for collaborative performance diagnosis.

The PerformaSure Approach to J2EE Diagnosis

The PerformaSure approach to the complex task of diagnosing J2EE performance issues is straightforward. First, it captures performance data from the J2EE system as it functions under user load (on either the live production system or a staging/test system). Once performance data is captured, PerformaSure's specialized GUI-based tools help track down the source of performance bottlenecks quickly. These tools are open-ended, enabling analysts to track down all types of J2EE performance problems. Because every complex investigation invariably leads the investigator down different paths of analysis, PerformaSure supports this approach. In general, analysts using PerformaSure follow an investigative process like the following:

1. Collect a high-level performance data session either from the production system or from a test system with a simulated load.

2. Determine which transactions are being executed during slowdowns using the Request Time View. For a problematic transaction, identify which server components appear to be responsible for the bulk of the slowdown. Here the focus is on the end-user perspective, where bottlenecks matter most.
3. Trace the execution path of the transaction using the Request Tree View. PerformaSure's diagnostics GUI helps quickly identify hotspots at a component, class or method level. The investigation can now take different paths, depending on the nature of the bottleneck.
4. Code-level problems are best diagnosed at this point by a deep source code analysis tool like Quest's JProbe, which can drill down to the root cause of algorithmic, memory or threading problems. Further diagnosis of JDBC or database problems is a job for a specialized database analysis tools like those found in Quest Central for Oracle or Quest Central for DB2.
5. Use PerformaSure's intuitive viewers to diagnose general and J2EE-specific problems like application server cluster configuration and limiting OS components. For example, identify periods of time when application resources are being taxed with the Threshold View. Then use the Metrics View to correlate these and different metrics across a time period to analyze, test and validate possible root causes.
6. Once the root cause is identified, fix the problem, test the solution and deploy the fix.

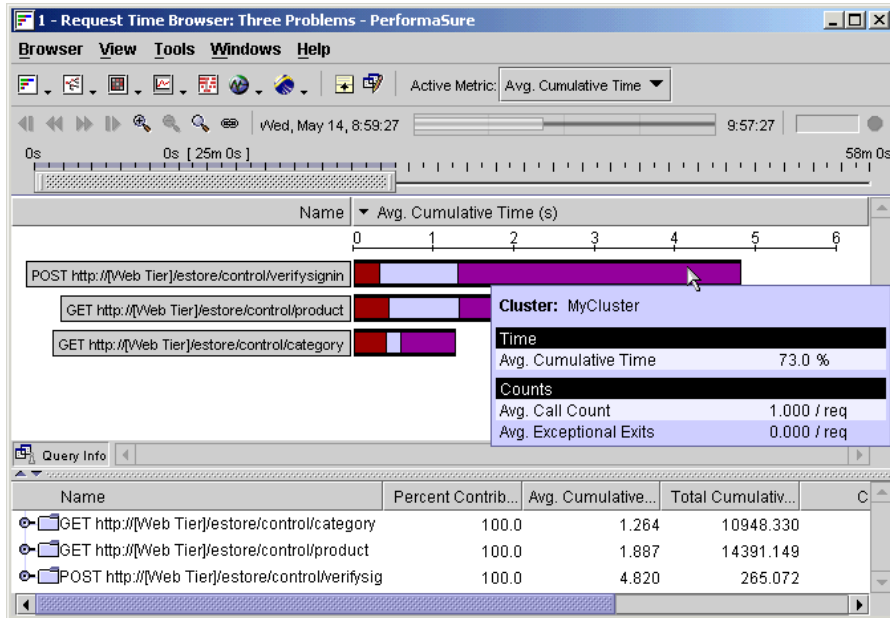
Because J2EE expertise varies within an IT team, the process can be easily split among application administrators, DBAs, developers, architects and quality assurance. This guide will walk through several real-world examples of diagnosing problems, using this approach.

Introducing PerformaSure's Viewer Windows

Diagnosis with PerformaSure is accomplished by working with its graphical viewers (also called browsers). There are five types of highly configurable GUI browsers available, as described below. For more basic information on PerformaSure's rich GUI, please see the *PerformaSure User's Guide* in the product documentation.

Request Time View

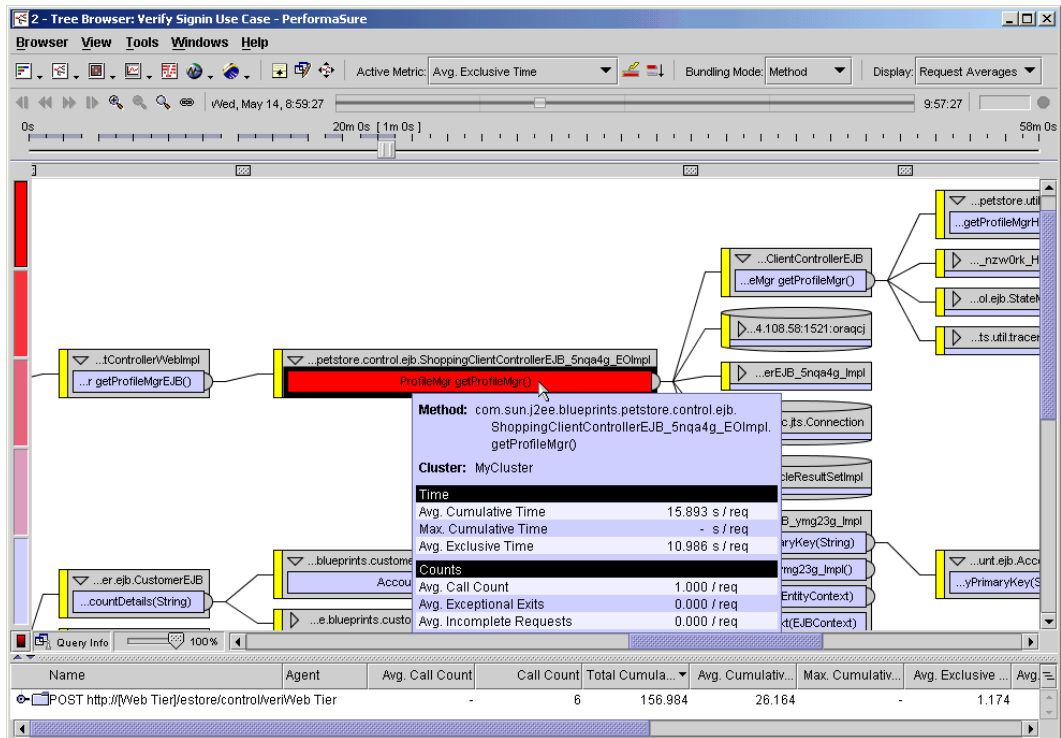
A good starting point for locating bottlenecks, the Request Time view shows the execution time of each transaction or service request, broken down by servers or clusters. The Request Time view's bar chart very quickly shows you the most expensive requests, and more importantly, which server(s) is responsible for the slow response times.



- Identify slow running transactions and the tier, cluster or server responsible
- Eliminate finger pointing and guesswork with objective data
- Drill down with other viewers to determine root cause of bottleneck
- Generate management-level PDF reports of problem transactions and offending servers

Request Tree View

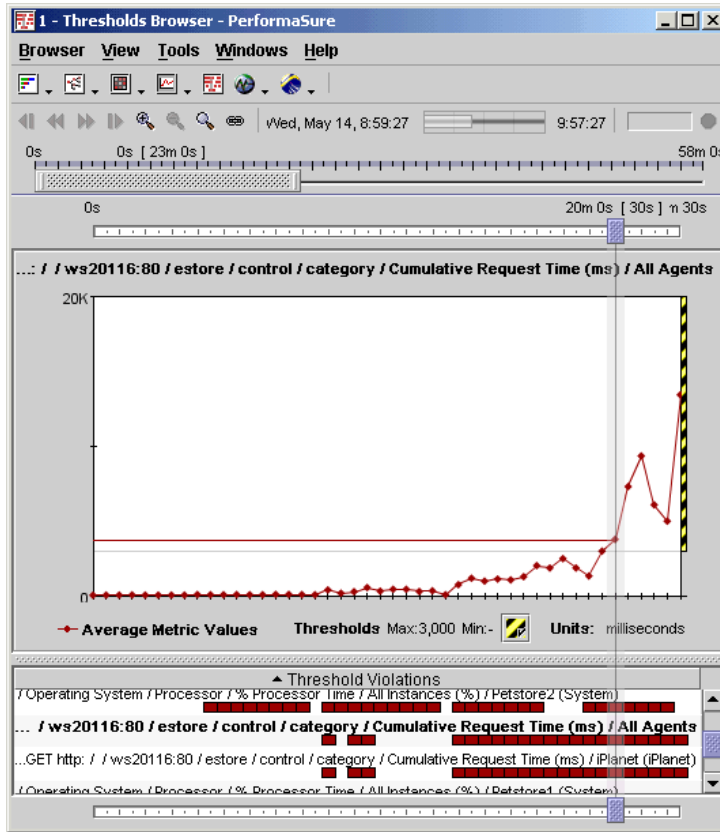
The diagnostic workhorse of PerformaSure is the Request Tree view, which shows the complete execution path of a transaction as a tree graph. This “call graph” is color-coded to highlight the most time-consuming component or method, along the most critical performance path. Like all PerformaSure viewers, the visual state of Request Tree view is saved in the project to enable easy collaboration with others on the team.



- View application architecture and end-to-end transaction execution paths—from Web server to SQL statement
- Identify problematic components by tracking unacceptable method-level timings, call counts and exceptional exits at the application code method level
- Examine cluster performance, load balancing problems, EJB interactions and RMI invocations
- Generate PDF reports identifying problematic components, methods and SQL statements for individual end-user transactions

Threshold View

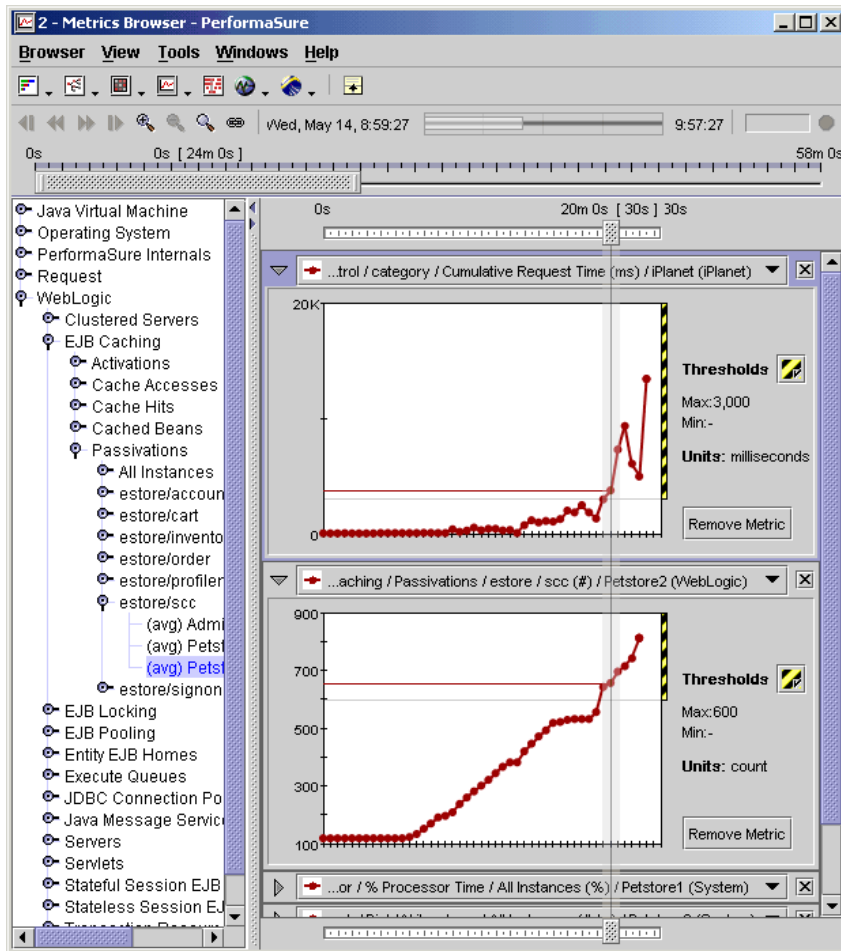
Another possible starting point for performance diagnosis is the Threshold view. This view provides a high-level aggregation of metrics that have exceeded their thresholds. For example, when the customer login function was slow, was the CPU excessively high, or was memory usage maxed out? Or was the server perhaps running low on JDBC connections or EJB caching pools? This view enables analysts to take a high-level view of metrics thresholds, to quickly identify problematic time periods.



- Quickly navigate and zoom into problematic time periods for further investigation
- Identify key system and application server metrics that have exceeded thresholds
- Focus on performance as it relates to end-user response times

Metrics View

The Metrics View provides a unified view for detailed investigation of a range of metrics, including JVM memory usage, operating system metrics and application server runtime metrics. Using a simple drag-and-drop interface, users can plot metrics individually or correlated according to time, graphically comparing performance across system tiers or between servers deployed in a cluster.



- Instantly correlate operating server metrics and application server runtime metrics among clustered servers or across servers from different tiers
- Tune application server configuration and deployment settings to maximize application performance without code changes
- Tune operating system parameters and identify limiting hardware and software components on all tiers
- Graphically identify clustering imbalances

Network Traffic View

The Network Traffic View provides a system-wide view of IP-to-IP network traffic between all servers in the deployed environment. This unified view of traffic allows a performance analyst to immediately identify which servers are sending and receiving unexpectedly large amounts of data at any given time, and drill down on problem time periods for further analysis.

- View incoming and outgoing network traffic from all deployed servers

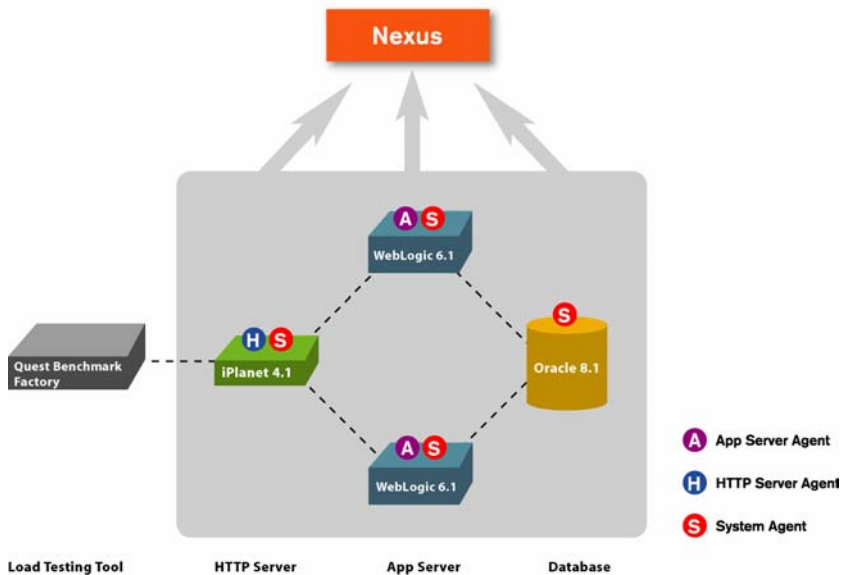
- Eliminate network bottlenecks due to poorly formed queries or excessive object serialization
- Diagnose network congestion between tiers based on end-user requests

Diagnosing Example Problems with PerformaSure

Now we're ready to go through some real examples, using PerformaSure to diagnose several canned performance problems in a J2EE application. But first, a bit about the application and environment.

The J2EE application in question is the BEA J2EE Pet Store application. Well-known and easily available, BEA's version of Pet Store has several performance issues that we can easily discover and diagnose.

The application was set up in a test environment made up of a small WebLogic server cluster, an HTTP server, and a database server.



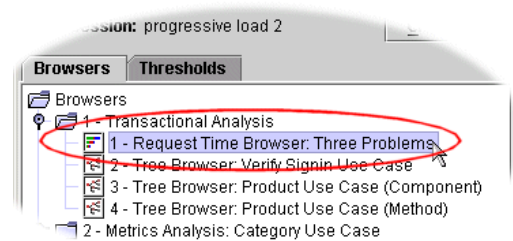
PerformaSure Agents were installed on each machine in the environment and configured to communicate with a PerformaSure Nexus machine. With all components of the application running, we captured a detailed performance metrics session with PerformaSure, using Quest's own Benchmark Factory to pump a simulated user load through the system. This brings us to the present – we have a recorded performance data session, and are ready to analyze it. Let's assume that one of our colleagues has done

some initial analysis and has created a project containing some initial views on the session. This gives us a jumpstart and will help us get right to analysis more easily.

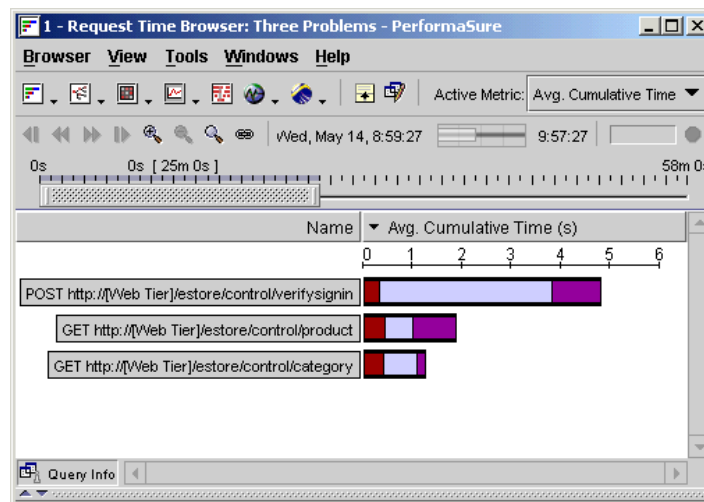
Start Workstation, Open Project and See Three Problems

If the PerformaSure Workstation is not already running, launch it, log in, and open the “Petstore Demo” project as described earlier.

Opening the project brings up the Project Browser window. Several folders and saved views are defined here, courtesy of our fictional colleague.



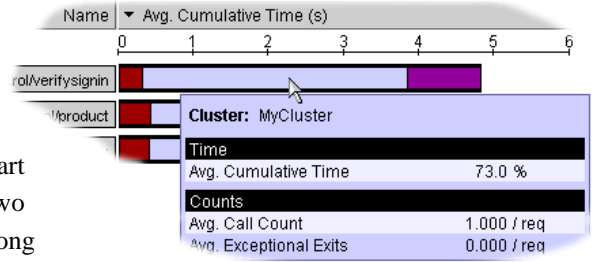
A good starting point for investigating performance bottlenecks in a J2EE application is the Request Time view, so let’s double-click the “Request Time Browser: Three Problems” item in the **Browsers** > **1 - Transactional Analysis** folder. This brings up the Request Time view.



This view shows us how much time it took to respond to user requests. Three very slow transaction requests are shown. Move the mouse pointer over the transaction name to see more details. The stacked bar chart to the right of each transaction breaks down the time to each tier and server (details pop up if you move the mouse over each section of the bar). Each transaction has performance issues that can be addressed. So let’s go.

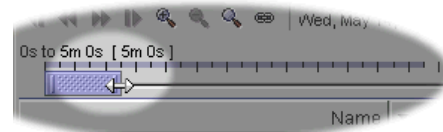
Problem 1: Inefficient Java code

We first notice that the “verifysignin” transaction seems to be spending an inordinate amount of time in one particular area of its execution. That is, one part is taking far longer than the other two parts. Moving the mouse over the long light purple bar, we see the transaction is spending 73% of its time in the WebLogic application server cluster (“MyCluster”). This indicates a possible problem with the application code.

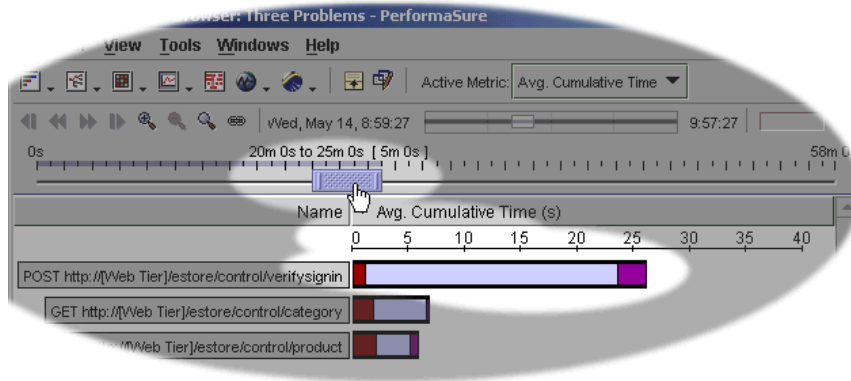


This view currently shows us the transaction’s average execution time, over the entire session. Before we drill down on the transaction, we should try to isolate the time period where the application code really started to bog down. To do this we can use a smaller time slice and “step” through the run using the Zonar control.

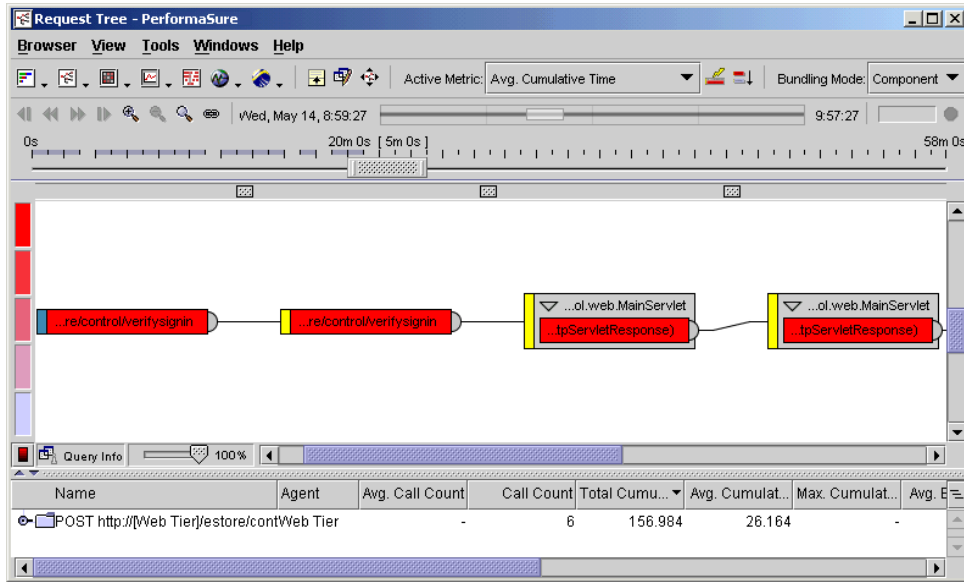
Grab the right edge of the Zonar and drag it left so that it only shows a five-minute time slice. Then move the entire Zonar along the session in five-minute increments. Notice that



at the 20-minute mark the total time jumps to 26 seconds and the application server portion becomes a really huge part of the execution time.



This looks like a good spot to start the next phase of the diagnosis. Right-click the verifysignin transaction name and select **New Tree Browser Using Selections**.




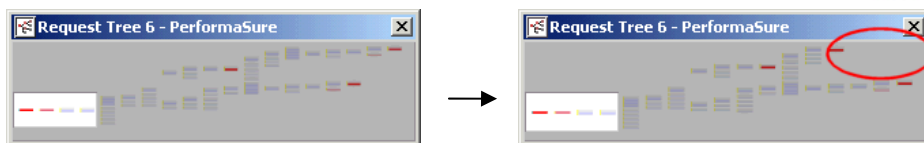
The Request Tree view shows us the execution path of the transaction, intuitively coloring bottlenecked components in bright shades of red. We can configure the view to show execution at different levels, and determine bottlenecks based on any one of a number of different metrics.

By default we see execution at the component level. Let's drill down to the method level and see what's bottlenecked by the Average Exclusive Time metric, which should really highlight the true bottlenecked area of the code.

Using the combo boxes in the Request Tree view window, change the Bundling Mode to Method and the Active Metric to Avg. Exclusive Time. Notice how the call graph changes to reflect the new selections.

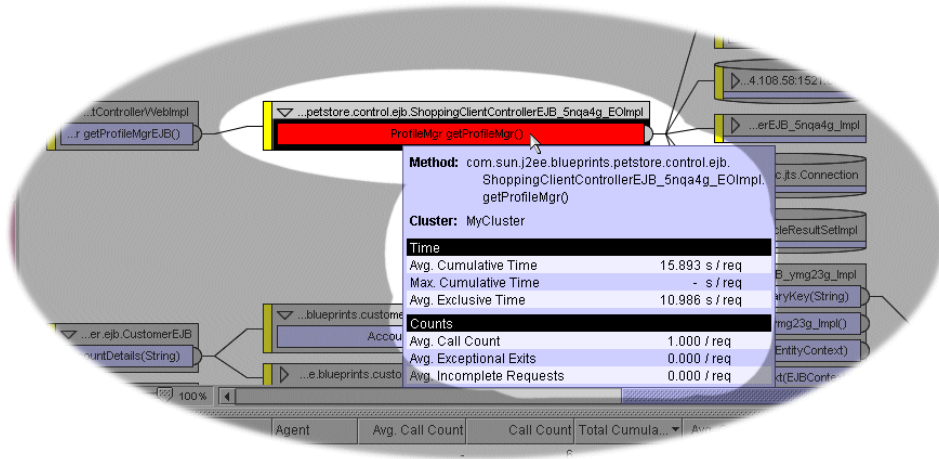


We can use the PerformaSure "Fast Find" feature to prune the call graph down to make the most critical paths easier to spot. To see this feature in action is to open the Panner window (using the  button) and then click the **FastFind** button.



Now there are only three red hotspots in the body of the transaction. Zoom in on the first one, in the middle of the tree. Use the column resizing knob to widen the node to read the

class and package name. When we move the mouse over the `ProfileMgr.getProfileMgr` method, the tooltip shows us that it alone is taking almost 11 seconds (Avg. Exclusive Time metric) to execute!



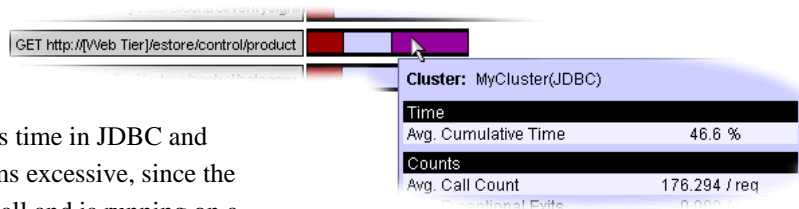
Quickly looking at the two other hotspots further along, we can see that JNDI lookups were being performed for the `ProfileMgr` class. This clinches it. The performance bottleneck is definitely being caused by a code problem in the `getProfileMgr` method. (**Note:** To double-check whether you’re seeing what you should be seeing, go to the Project Browser and open the “Tree Browser: Verify Signin Use Case” item – this shows the Request Tree view configured to this point.)

PerformaSure makes it easy to continue analysis with a more specialized tool. Because this is a code-level problem, we can right-click the method in the Request Tree view and select **Export JProbe Launcher File**. This file can then be used to launch JProbe to analyze this particular method to get to the root cause of the problem and fix it.

Problem 2: Misuse of database connection pool

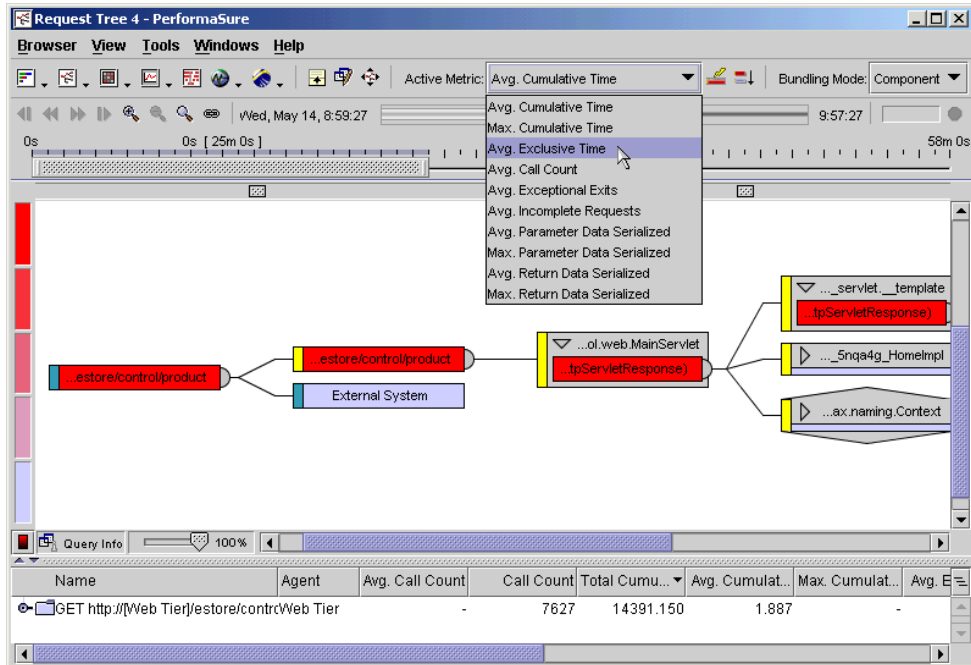
Let us now diagnose a database-related problem. Close the Request Tree view and look at the Request Time view again. Re-set the Zonar to show the entire run.

The second transaction, “product”, seems to be spending almost half its time in JDBC and the database. That seems excessive, since the Petstore database is small and is running on a powerful, dedicated Oracle 8i server.

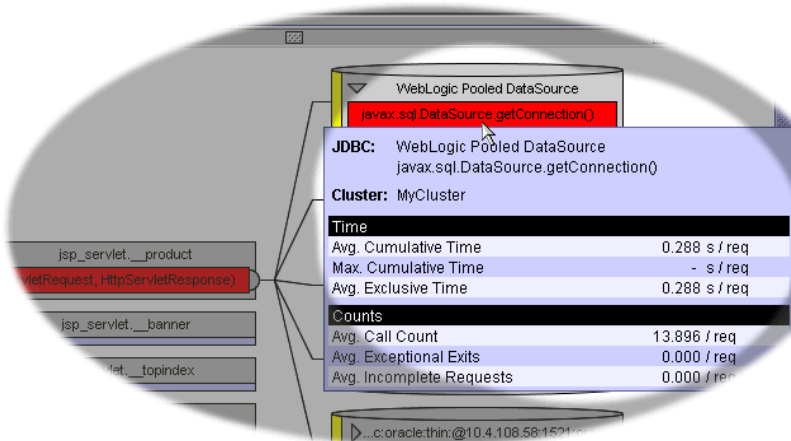


To see what’s going on, let’s right-click the “product” transaction name in the Request Tree view and select **New Tree Browser Using Selections**.

In the Request Tree view, change the Active Metric to “Avg Exclusive Time” to isolate the hotspots better.



It’s now pretty easy to identify the hotspot. Scroll the call graph, if necessary, to find the bright red JDBC nodes (resize the column to view more of the node). We can see that the `DataSource.getConnection` method is being called about 13 times per client request.



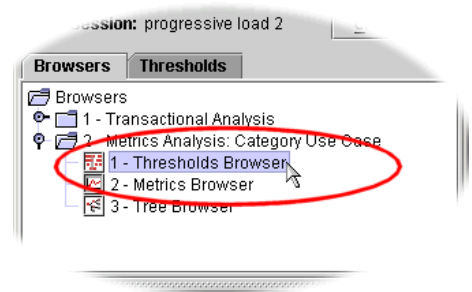
The `executeQuery` method call shows similar information, but we need to go deeper, to examine the actual SQL query being generated. Because PerformaSure is integrated with Quest's deep diagnostic tools for database administrators, we (or our DBA) can now simply right-click the node and launch the appropriate tool. If a tool was installed, we would right-click the `getConnection` method and select **Performance Diagnostics** or **SQL Tuning** from the pop-up menu.

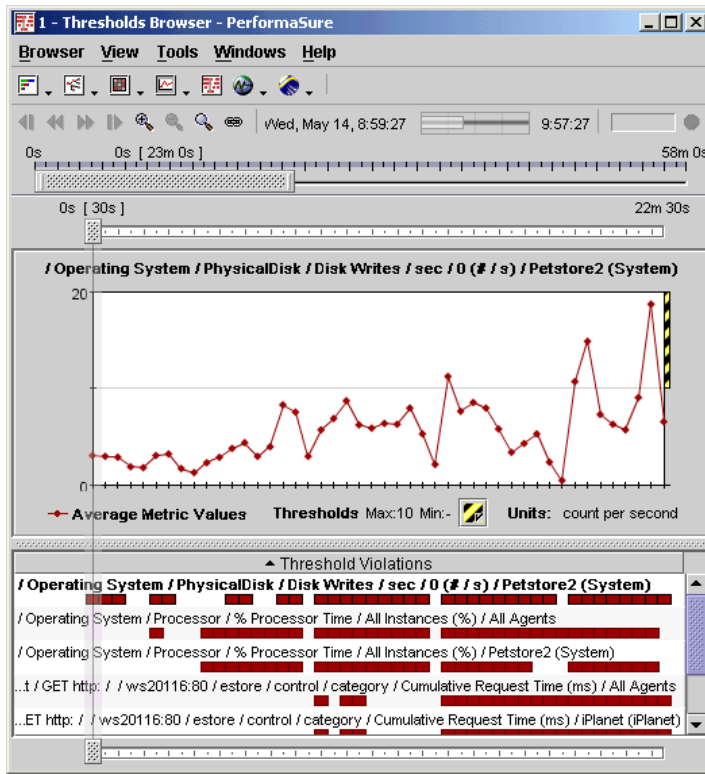
We have tracked down the root cause of this performance issue – the JDBC call being generated by the application server is misusing the app server's database connection pool, causing too many queries to be run. An expert has enough information and context through PerformaSure to determine the best way to optimize the DB connection pool.

Problem 3: EJB Passivation from Metrics evidence

So far, our performance diagnosis workflow has been to start with the high-level Request Time view and drill down to the details with the Request Tree view. This approach works well for transactional performance issues. Sometimes, however, J2EE performance issues aren't easily traceable in this manner. There are times when you need to correlate seemingly insignificant clues from many areas of the system in order to find the source of a problem. PerformaSure makes it easy to diagnose a problem by analyzing thresholds and metrics.

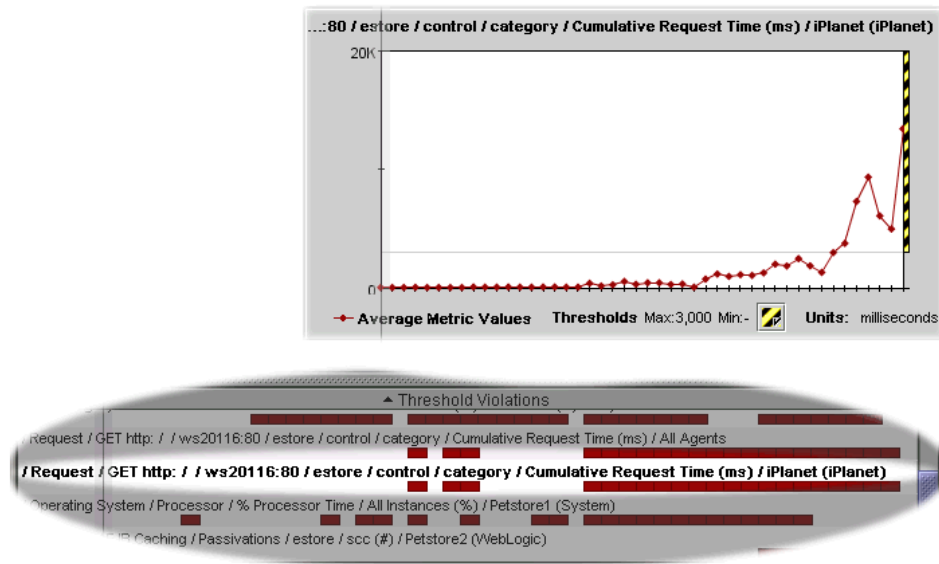
Rather than starting with the Request Time view (close that window if it's still open), we can start with the Thresholds browser. Bring up the Project Browser window and find the **2 - Metrics Analysis: Category Use Case** folder. Open the folder and double-click the "Thresholds Browser" item. This opens the Thresholds view.





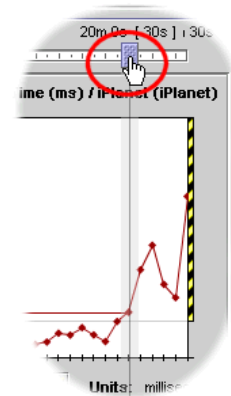
The Thresholds view provides a list of all the metrics whose thresholds were exceeded at some point in the session. There can be many metrics listed (the bottom part of the window), so how does one know where to start?

Logically, it makes sense to start with a problematic transaction. These would be entries in the table beginning with “Request”, signifying metrics on a specific transaction request type. Quickly scanning the list in the Thresholds browser, we notice one. Click the “/Request / GET http://ws20116:80 ... / Cumulative Request Time / iPlanet” metric in the list. The graph changes to show details on this metric.



From this graph, we can see that the response time (represented by the Y-axis) was very fast for most of the run time (represented by the X-axis). But near the end of the run, the response time slows down dramatically. We need to take a look at everything that was happening in the system at the point response time started slowing down.

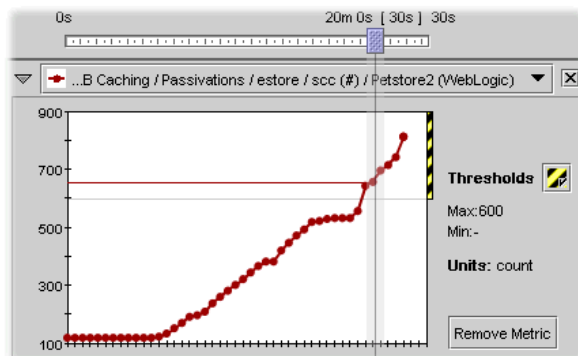
This is easy to do with PerformaSure. Find the Sightline bar that is located over the metrics graph. Click the handle and drag the Sightline to the 20 minute point on the graph. This tells PerformaSure that you want to consider and correlate metrics at this point in the run.



Now we want to overlay *all* of the metrics that were exceeded at the point specified by the Sightline. This is a job for the Metrics browser. To open the Metrics browser with metrics exceeded at the Sightline point, simply right-click the graph and select **New Metric Browser With Highlighted Metrics** from the pop-up menu.

The Metrics browser has two panes. The left pane contains all of the metrics available to be viewed. The right pane contains open metric graphs. By displaying several metrics and using the Sightline, you can make cause-and-effect ties between different metrics.

The first step is to look at the graphs for metrics exceeding their threshold at the time in question (the 20 minute mark). We immediately notice that the number of Stateful EJB Passivations exceeded its threshold at around the same time. This is likely significant because



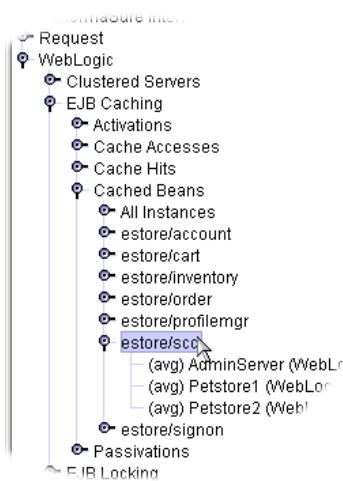
passivation is quite an expensive task.

But doing a metrics-based diagnosis means that we need more evidence to make the diagnosis. We have an idea of what may be happening, but is this causing the slowdown? In a complex J2EE system, a metric like this may be the cause, or it may be just a symptom – something else may be the root of the slowdown, with passivation just a side-effect.

If we look down the set of exceeded-threshold graphs, we see that Disk Writes per second on the application server machine heavily exceeded its threshold around this time – good corroborating evidence on our passivation theory. The other graphs don't seem to provide any evidence on the problematic transaction.

Remember that the graphs shown are only those metrics that exceeded their thresholds at the 20-minute mark. We can easily open other metrics graphs using the list in the left pane of the Metrics browser. For example, we can see how many beans were in the cache at this time by opening the WebLogic group, and double-clicking the **EJB Caching > Cached Beans > estore/scc > Petstore 1...** metric. This opens the graph for that metric.

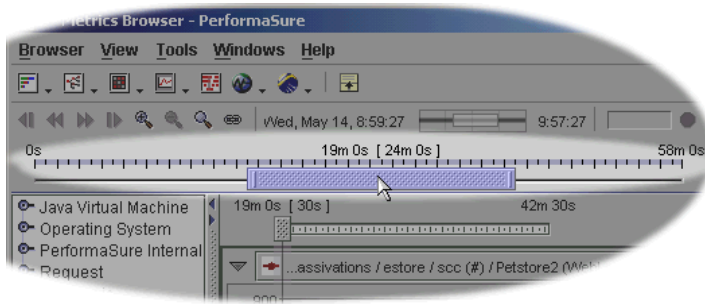
In this graph, we see that the number of beans in the cache is very high. This is more solid evidence that EJB passivation is the root cause.



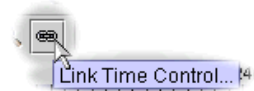
However, to be certain of this diagnosis, we should tie these metrics back to the code running in the application server at the same time. We'll use the Request Tree view, of course, but we can also customize the view so it's tied to the timeframe in our Metrics browser.

From the Project Browser window, open the "Tree Browser" item in the **Browsers > 2 - Metrics Analysis: Category Use Case** folder. Size both the Request Tree view and the Metrics browser so they're both visible.

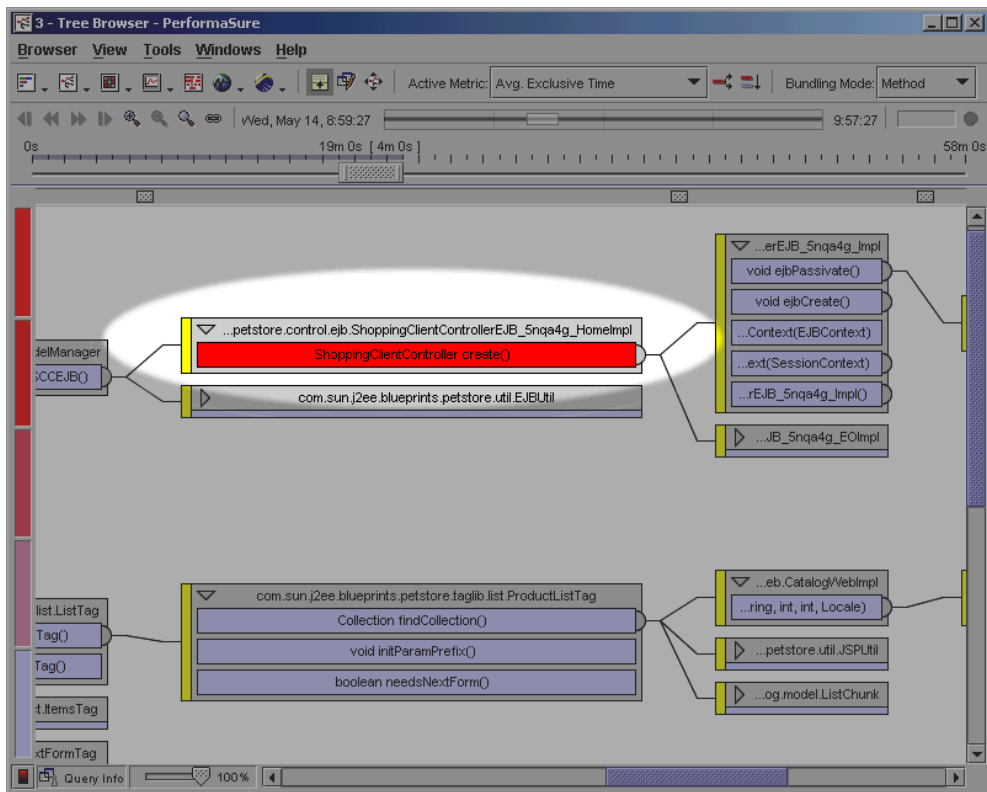
Now we want to zoom in on the timeframe in question and tie both windows together using the Zonar and the Link Time controls. From the Metrics browser, locate the Zonar (under the second toolbar at the top of the window). The Zonar lets you zoom the view to only show information for a particular time period. You can pick any point along the run, and select any duration of time. Click and drag the Zonar selection bar to the 19 minute mark as shown below:



Notice that the metrics graphs are redrawn to show the new time constraints, that is, the point at which passivation exceeds our threshold. Now let's update the Request Tree view to the same time constraint. To do this, click the **Link Time Control...** button and specify "Link All" from the control dialog. Now the Request Tree view is linked, showing the time set up for the Metrics browser. Moving or sizing the Zonar in one window affects the other. Resize the end of the Zonar to 4 minutes duration. Now in the Request Tree view we are guaranteed to be looking at performance hotspots for the time passivation became problematic.



In the Request Tree view, change the Active Metric to Avg. Exclusive Time and use the **Expand Critical Paths** button to hide unrelated nodes. There's only one hotspot in the call graph. It's in the EJB `create` method. This was the bottleneck at this time period.



A bit of research into how the EJB `create` method works reveals that this method is responsible for performing EJB passivation, when necessary.

So we've come to the definite root cause of this performance issue. While the metrics pointed towards passivation, it was only by tying back to the transaction execution path (something only PerformaSure can do) that we could definitively state that this was in fact the root cause of the problem.

For More Information

For additional information on PerformaSure:

- Visit <http://java.quest.com/performasure>
- See *Honing in on J2EE System Performance, An Executive White Paper* Aberdeen Group
- See *PerformaSure User's Guide*

For information on companies that are successfully using PerformaSure or additional information, please contact:

Rini Gahir
PerformaSure Product Manager
Quest Software
(416) 643-4849
rini.gahir@quest.com

Erin Jones
Public Relations Director
Quest Software
(949) 754-8032
erin.jones@quest.com

About Quest Software

Quest Software is a leading provider of application management solutions. Quest provides customers with Application Confidence(sm) by delivering reliable software products to develop, deploy, manage and maintain enterprise applications without expensive downtime or business interruption. Quest products increase the performance and uptime of business-critical applications and enable IT professionals to achieve more with fewer

resources. Headquartered in Irvine, Calif., the company has offices worldwide and more than 18,000 global customers. For more information, visit www.quest.com.

Java/J2EE Products Available from Quest Software

Foglight for J2EE — *The Application Monitoring Solution*

Foglight is a 24x7 distributed application-monitoring solution that provides end-to-end monitoring of every component affecting application performance. J2EE cartridges monitor the health of the application server environment to detect problems before end users are impacted.

Visit <http://www.quest.com/foglight>.

Spotlight on WebLogic Server — *Real-time Application Server Performance Viewer*

Spotlight on WebLogic Server provides real-time performance information on the application, service and resource usage within WebLogic Server. Color-coded alarms enable the WebLogic administrator to quickly determine which server or application is experiencing a performance problem. Once a problem is exposed, expert advice provides insight into the problem and how to resolve it.

Visit <http://java.quest.com/spotlight>.

JProbe — *Premier performance tuning toolkit for Java*

Designed for both client- and server-side environments, JProbe is a comprehensive, award-winning toolkit for diagnosing code errors and inefficiencies. JProbe paints graphical pictures of everything from memory usage to calling relationships, helping developers to understand precisely what is causing problems in Java applications, servlets, JSPs and EJBs – right down to the offending line of source code.

Visit <http://java.quest.com/jprobe>.

JClass — *The only Java server and client components you need*

JClass is a comprehensive set of Java components that enables developers to add sophisticated user interfaces and dynamic content to rich- and thin-client applications. Whether GUI components for Swing-based development or server-side components for servlet- and JSP-based presentation interfaces, JClass accelerates the development of professional business applications.

Visit <http://java.quest.com/jclass>.

DeployDirector — *Java provisioning and management*

DeployDirector is a Java application provisioning and management solution that helps IT organizations to deploy, manage and update rich clients. DeployDirector maximizes uptime of core systems while dramatically reducing time to repair.

Visit <http://java.quest.com/deploydirector>.