TECHNOLOGY WHITE PAPER

now from symantec

VERITAS

Application Performance Management

Introduction to Adaptive Instrumentation with VERITAS™ Indepth for J2EE

TABLE OF CONTENTS

ABOUT ADAPTIVE INSTRUMENTATION

Adaptive instrumentation is a patent pending technology that automatically discovers and isolates the major contributors to poor performance in J2EE applications while minimizing overhead.

This paper reviews the traditional approach to monitoring J2EE applications, lists problems associated with this approach and explains how Adaptive Instrumentation resolves these problems. A glossary of basic terms used throughout this document is included in Appendix A.

WHY ADAPTIVE INSTRUMENTATION?

Measuring and isolating performance bottlenecks in production Java 2 Platform Enterprise Edition (J2EE) applications requires an efficient data collection technology that provides detailed performance metrics with minimal overhead. Excessive overhead is usually the result of over-monitoring; therefore, deciding what components of the application should be monitored is a key issue. Traditionally, this has been a manual trial and error process where the application structure is examined and components that could potentially cause a bottleneck are added to the list of monitored components. If undetermined performance issues still exist after all components on the list are monitored, a manual analysis process needs to be employed until the root cause of the remaining performance problems is found. During this process, if too many components are monitored, the associated overhead will affect the system and some components will have to be removed from the list of monitored components to maintain reasonable overhead.

This process is tedious, time-consuming and requires a significant amount of developer experience. It is typically repeated during application testing as well as after the application is deployed in production since the test environment is often different from the target production environment.

Also, changes in an application's Java code, workload patterns and database behavior frequently introduce new performance problems. Any of these changes will require additional manual refinement to the process described above.

The question is: How can we improve this process? The answer is Adaptive Instrumentation.

DISCOVERING PERFORMANCE PROBLEMS

Most J2EE application performance monitors gather information on applications by placing probes in key locations throughout the application code. The probes gather performance data on specific components as the application executes. By gathering and analyzing the data collected by the different probes, an application performance monitor can provide accurate information on the overall behavior of the application, point to bottlenecks in the application and recommend necessary adjustments to alleviate those bottlenecks.

Inserting probes in the right places to balance the need for detail with acceptable overhead is a key factor in correctly measuring performance. For J2EE applications, the most elementary unit of work is a Java method. To determine the amount of time spent in each method, measurements are taken before and after the method is invoked. By looking at the difference in measurements (e.g., time, memory usage) before and after a method is invoked, we can get an understanding of the individual method's contribution to the overall performance of the application. We can then record the data in a performance database for further analysis.

The process of inserting the probes into the application code is called *Instrumentation.* Veritas *Indepth™ for J2EE* pioneered the use of a technique called *byte-code instrumentation* which allows insertion of probes into classes' and methods' binary code. This technique removes the need to access the application's source code and thereby allows instrumentation of any Java application. VERITAS byte-code instrumentation is done at run-time, as the classes are loaded.

WHICH METHODS SHOULD BE MONITORED?

A comprehensive approach applies instrumentation to all methods in the application so that maximum information is collected on the performance of that application. This approach, however, is limited to small applications in development environments since data collected by the various probes does not come for free. Taking measurements and recording them consumes expensive system resources and affects the executing applications. If all methods are instrumented, the application may suffer severe performance degradation. In some extreme cases, more time may be spent on recording performance data than on executing the application itself.

A dilemma arises between the need to maximize visibility into the application's performance and the need to minimize overhead that is caused by taking the measurements. As we increase the instrumentation level, more time is spent on measuring performance which results in a slower application. We need to strike a balance between the overhead caused by performance management and the visibility needed to identify and isolate performance bottlenecks.

HOW DID WE FIND THE RIGHT BALANCE IN THE PAST?

Most J2EE applications use well known interfaces that are part of the J2EE specification. Some of these interfaces are Java Server Pages (JSPs), Enterprise Java Beans (EJBs) and Java Database Connectivity (JDBC) drivers. Generally, byte-code instrumentation of these interfaces provides automatic identification of the response time contributions related to the use of relational databases, EJBs and Internet user interfaces. VERITAS *Indepth for J2EE* and similar J2EE performance monitors usually instrument all or most of the standard J2EE interfaces.

However, J2EE applications often spend significant amounts of time performing operations outside the scope of common J2EE interfaces, such as communicating with legacy applications, using third party packages, and executing logic that is unique to the application. To measure the performance of these components, we need the flexibility to apply instrumentation to a broader set of methods and classes. VERITAS Indepth for J2EE provides this capability through a feature called *Custom Instrumentation*.

CUSTOM INSTRUMENTATION

The custom instrumentation feature of VERITAS *Indepth for J2EE* allows users to monitor the performance and obtain the drilldown invocation context of any Java method. However, when customizing the list of instrumented methods, users must explicitly specify a method name, which means they must "know" the methods to instrument. Discovering the right methods to instrument is a tedious, iterative, and time-consuming process that requires detailed knowledge of the application.

Custom instrumentation involves adding specific methods to the instrumented methods list. Typically, the underlying J2EE application must be restarted whenever the list of methods to be instrumented changes. Restarting the application server interrupts running applications, thereby, impacting availability – not an optimal scenario for mission-critical web-based applications. This re-instrumentation process must be repeated until the root cause of a performance problem is isolated.

Overhead increases as additional methods are custom-instrumented, however; there is no way to quantify the amount of overhead associated with each additionally instrumented method. The only way to know that the overhead is too high is to over-instrument, watch the effect and then back out some or all of the new instrumentation. To back-out instrumentation – we need to decide which instrumented methods should be removed from the list, thereby reducing the overhead back to an acceptable level. This trial and error process imposes serious limits on the ability to monitor applications. It requires prior experience and understanding of the anticipated effect of instrumenting each method.

SUMMARY OF THE PROBLEMS WITH THE OLD APPROACH

As we have seen, performance management of J2EE applications is currently conducted in an environment that poses significant challenges to developers, performance analysts and others charged with the responsibility of providing smooth running applications with acceptable performance and service levels. We summarize these challenges here, before describing how Adaptive Instrumentation minimizes or even eliminates them:

• Limited visibility –	Default instrumentation does not necessarily cover the important application components. There is a need to manually discover bottlenecks and key components <u>using intimate knowledge of the application</u>
• Slow discovery -	An <u>iterative discovery process</u> is required by which you slowly reveal the *"hot spots"* in the application
• Manual adjustments -	The need to <u>manually add each discovered method</u> to the list of instrumented methods using *Custom Instrumentation*
• Impact on availability -	The need to <u>restart applications and/or application servers</u> for the new instrumentation to take effect, thereby, <u>interrupting availability</u>.
• Overhead assessment-	The inability to <u>understand the overhead associated with the instrumentation</u> of each additional method. In the absence of prior knowledge of problem areas in applications, IT organizations tend to "over instrument"; this trades off greater visibility at the expense of overhead

THE ADAPTIVE APPROACH

VERITAS Indepth for J2EE now employs Adaptive Instrumentation to address all of the problems described above. Adaptive Instrumentation uses advanced (patent pending) technologies to measure and collect a high-level view of an application's activity. It then uses this information to:

- Automatically choose the best candidate methods for instrumentation
- Filter the candidate method list to limit the overhead to a user-specified level
- Automatically generate a list of candidate methods for instrumentation
- Apply the instrumentation without needing to restart the application

The Adaptive Approach significantly reduces time and effort to isolate and resolve application performance issues; minimizes the need for detailed application knowledge; and avoids any negative impact on application availability.

HOW DOES ADAPTIVE INSTRUMENTATION WORK?

Adaptive Instrumentation is a three step solution:

1. **Survey**	Monitor all of the methods in the application with ultra-low overhead and record key information that allows Adaptive Instrumentation to evaluate the contribution of each monitored method to the overall performance.
2. **Analyze**	Analyze the information gathered on all the methods in the application and determine which methods should be monitored, given an overhead budget.
3. **Instrument**	Dynamically instrument the selected methods avoiding the need to restart the application

These three steps allow the Adaptive Instrumentation feature to choose which methods should be instrumented given a certain overhead budget. To help it make the right decisions in choosing which methods to instrument, we need to provide Adaptive Instrumentation with the amount of overhead (expressed as percentage of the application's *response time*) that we consider acceptable.

STEP 1 - TAKING THE SURVEY

The purpose of the survey is to map the contribution of each and every method in the executing application. During the survey, all invoked methods are measured for their contribution to overall performance. Since instrumenting all of the application methods may result in high overhead, the survey relies on the following principles:

- Only collect data that is absolutely essential to the evaluation process. This data includes the total *response time* and the *invocation count* for each method along with enough data to construct a call graph of the application logic.
- Run for a limited amount of time to reduce the impact of the survey's overhead.

The amount of time that the survey runs is determined by the user. It must be long enough to allow all important components of the application to execute as the survey cannot measure methods that were not invoked.

STEP 2 - ANALYZING THE SURVEY DATA

Once the survey ends, we have a list of all methods that were invoked in the application, the amount of time spent in each method and the number of times each method was invoked. Using the call graph collected by the survey along with the response times and invocation counts, the Adaptive Instrumentation technology now examines the results and decides which methods should be instrumented. A well defined set of principles guide Adaptive Instrumentation in choosing which methods to instrument. Some of these principles are:

- DO NOT INSTRUMENT methods with high invocation counts
- INSTRUMENT methods in descending *work time* order
- INSTRUMENT call tree paths throughout the application to isolate the largest contributors to response time in the context of the application logic
- Do not exceed the user-defined overhead budget

STEP 3 - INSTRUMENTING SELECTED METHODS

Method instrumentation is applied starting with the default instrumentation which includes the most common J2EE interfaces: Java Server Pages, Enterprise Java Beans and JDBC calls. We then instrument methods that were selected by the analysis step described above. Instrumentation of methods stops as soon as the allocated overhead budget is reached. With very small overhead budgets, it is possible to have only a small portion of the default instrumentation applied. The ability to control the amount of overhead caused by instrumentation is a unique feature of VERITAS Indepth for J2EE, known as *Over-Instrumentation Protection*.

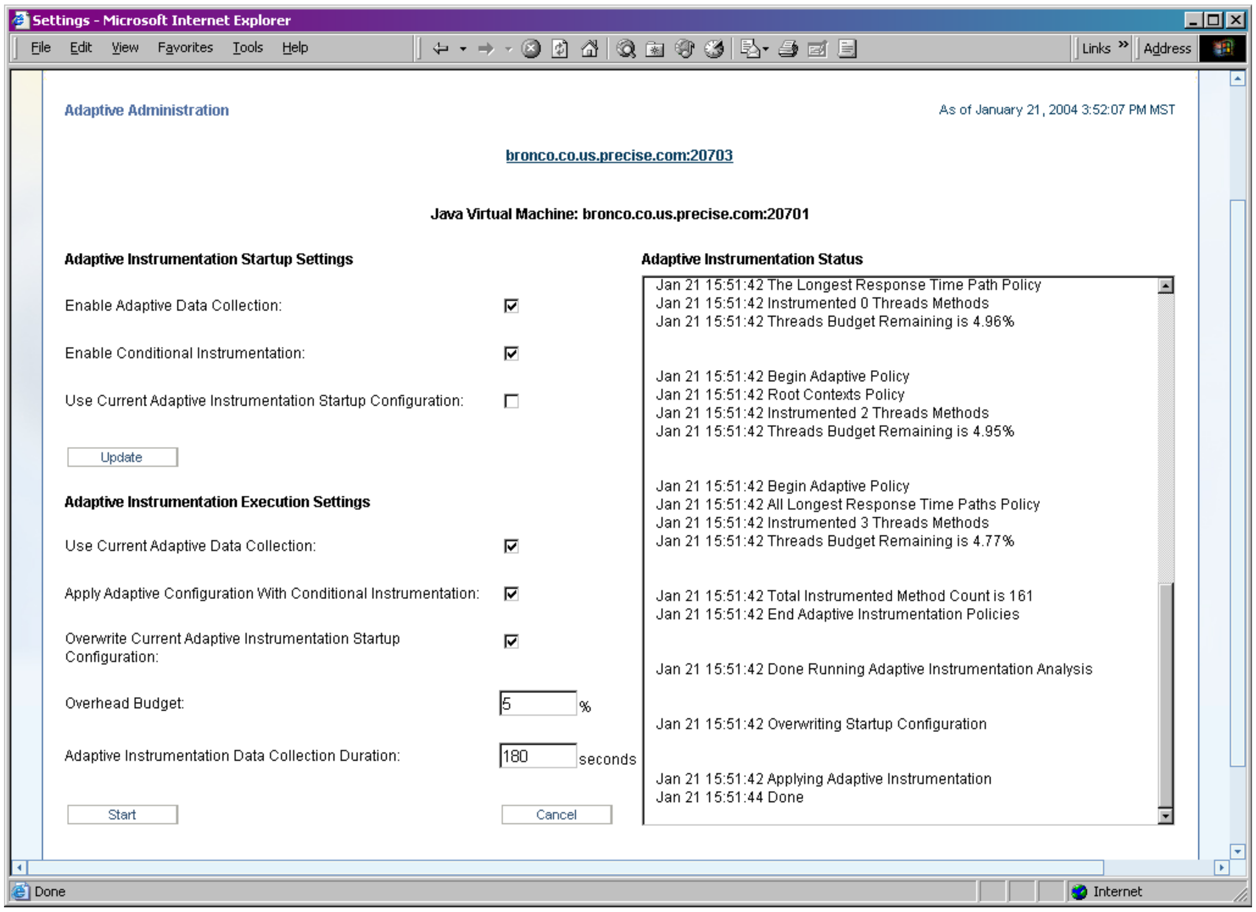
OVER-INSTRUMENTATION PROTECTION

Over-Instrumentation Protection allows users to specify a maximum acceptable overhead beyond which instrumentation should not be applied, regardless of whether it originates in the default instrumentation (e.g., JSPs) or in an Adaptive Instrumentation survey. This unique feature helps protect the application system from user errors that may result in unexpected overhead.

HOW TO RUN ADAPTIVE INSTRUMENTATION?

Adaptive Instrumentation is an integral part of VERITAS *Indepth for J2EE*. The user interface provides a specific configuration panel for Adaptive Instrumentation that allows users to control the instrumentation process. Using the controls on this panel users can:

- Specify the overhead budget – expressed as a percentage of overhead
- Specify the duration of the Adaptive Instrumentation survey
- Start/Stop an Adaptive Instrumentation Survey
- Dynamically Apply instrumentation


Settings - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links Address
Update
Start
Cancel
Done
Internet

Adaptive Administration

As of January 21, 2004 3:52:07 PM MST

bronco.co.us.precise.com:20703

Java Virtual Machine: bronco.co.us.precise.com:20701

Adaptive Instrumentation Startup Settings

Enable Adaptive Data Collection: ✓

Enable Conditional Instrumentation: ✓

Use Current Adaptive Instrumentation Startup Configuration: ☐

Adaptive Instrumentation Execution Settings

Use Current Adaptive Data Collection: ✓

Apply Adaptive Configuration With Conditional Instrumentation: ✓

Overwrite Current Adaptive Instrumentation Startup Configuration: ✓

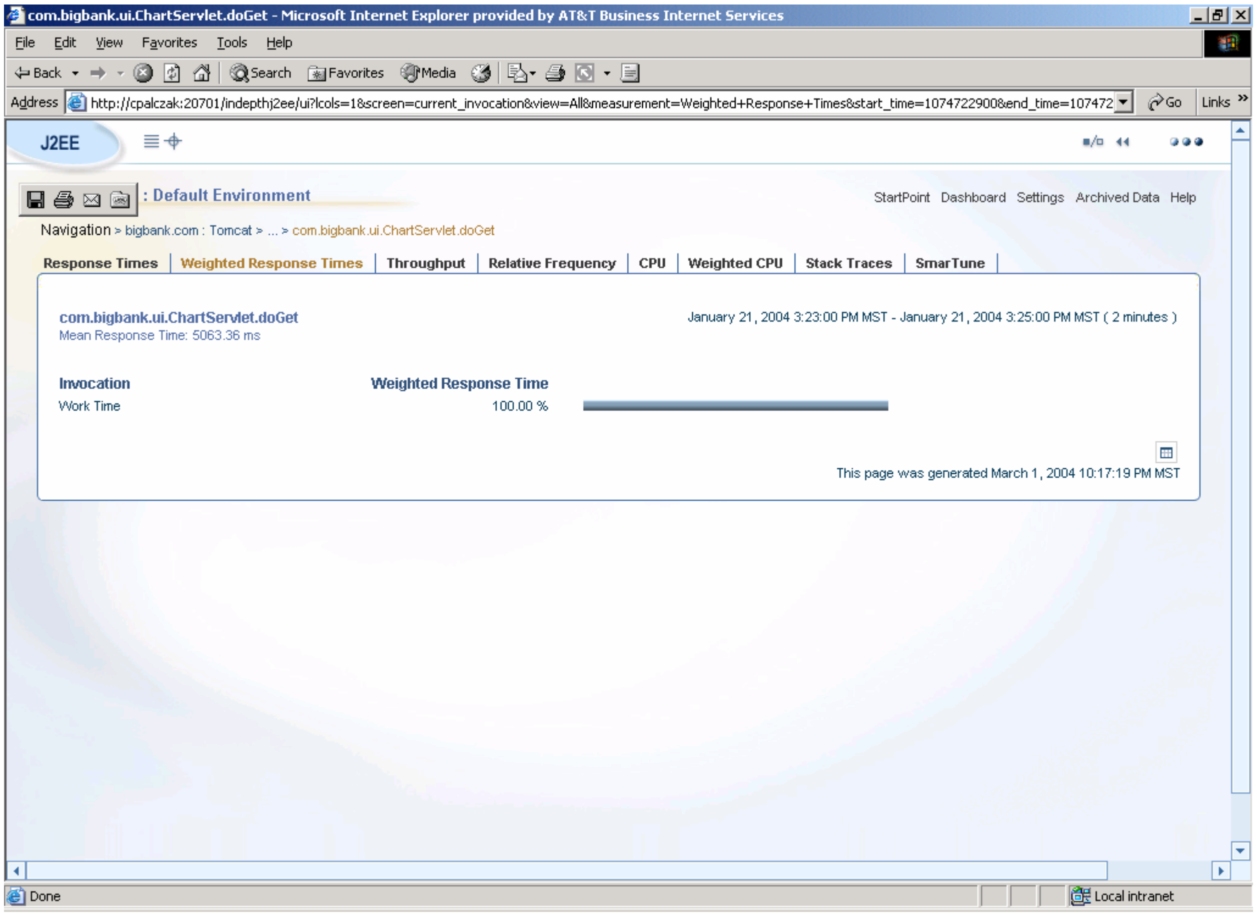
Overhead Budget: 5 %

Adaptive Instrumentation Data Collection Duration: 180 seconds

Adaptive Instrumentation Status

```
Jan 21 15:51:42 The Longest Response Time Path Policy
Jan 21 15:51:42 Instrumented 0 Threads Methods
Jan 21 15:51:42 Threads Budget Remaining is 4.96%


Jan 21 15:51:42 Begin Adaptive Policy
Jan 21 15:51:42 Root Contexts Policy
Jan 21 15:51:42 Instrumented 2 Threads Methods
Jan 21 15:51:42 Threads Budget Remaining is 4.95%


Jan 21 15:51:42 Begin Adaptive Policy
Jan 21 15:51:42 All Longest Response Time Paths Policy
Jan 21 15:51:42 Instrumented 3 Threads Methods
Jan 21 15:51:42 Threads Budget Remaining is 4.77%


Jan 21 15:51:42 Total Instrumented Method Count is 161
Jan 21 15:51:42 End Adaptive Instrumentation Policies


Jan 21 15:51:42 Done Running Adaptive Instrumentation Analysis


Jan 21 15:51:42 Overwriting Startup Configuration


Jan 21 15:51:42 Applying Adaptive Instrumentation
Jan 21 15:51:44 Done
```

Figure 1: Adaptive Instrumentation Control Panel



com.bigbank.ui.ChartServlet.doGet - Microsoft Internet Explorer provided by AT&T Business Internet Services
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cpalczak:20701/indepthj2ee/ui?lcols=1&screen=current_invocation&view=All&measurement=Weighted+Response+Times&start_time=1074722900&end_time=107472
Go Links
J2EE
: Default Environment
StartPoint Dashboard Settings Archived Data Help
Navigation > bigbank.com : Tomcat > ... > com.bigbank.ui.ChartServlet.doGet
Response Times | Weighted Response Times | Throughput | Relative Frequency | CPU | Weighted CPU | Stack Traces | SmarTune
Done
Local intranet

com.bigbank.ui.ChartServlet.doGet
Mean Response Time: 5063.36 ms

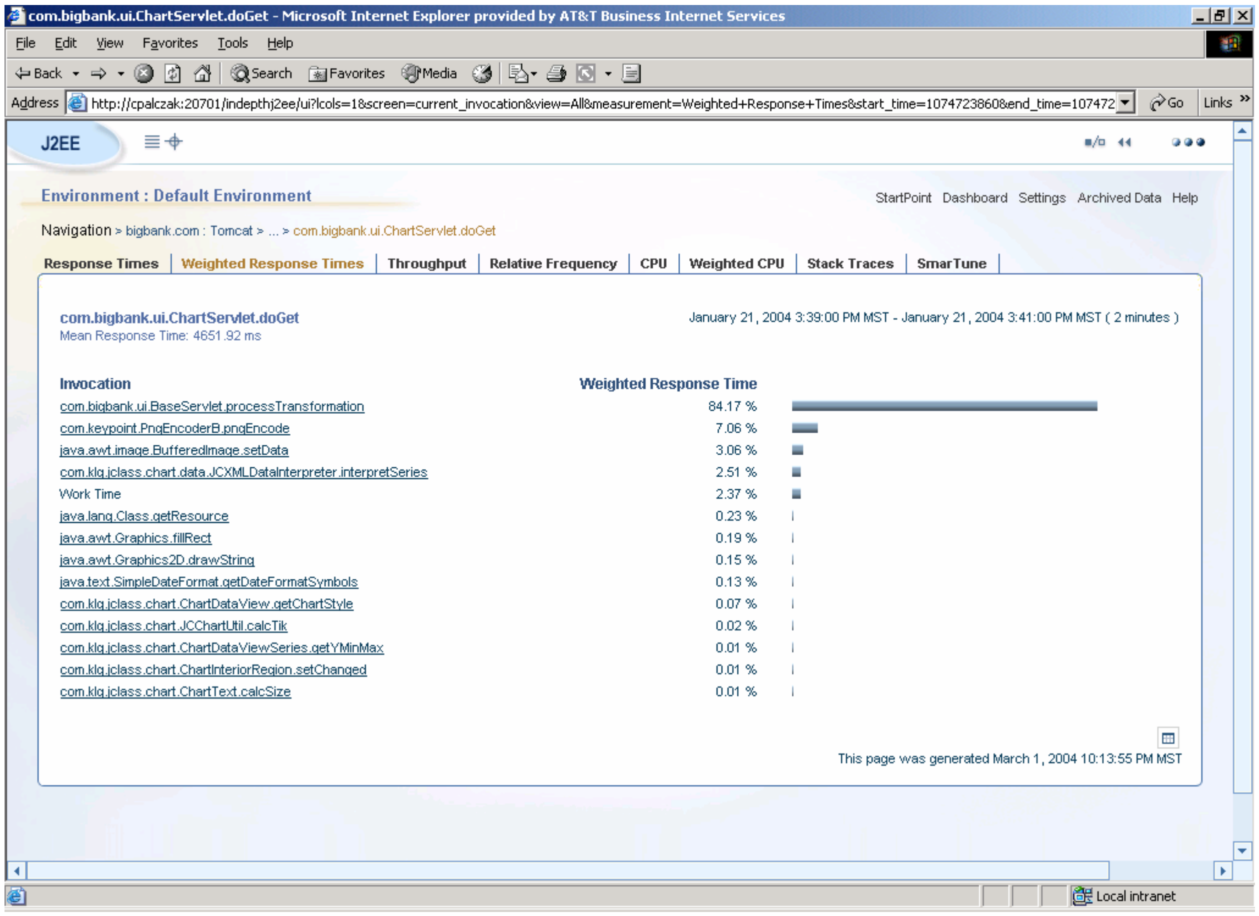
January 21, 2004 3:23:00 PM MST - January 21, 2004 3:25:00 PM MST (2 minutes)

Invocation	Weighted Response Time
Work Time	100.00 %

This page was generated March 1, 2004 10:17:19 PM MST

Figure 2: Before Adaptive Instrumentation – No specific method information available

Before Adaptive Instrumentation, total application overhead is rolled up into a single component called Work Time. Not enough information is available to determine the specific methods that are executed.


com.bigbank.ui.ChartServlet.doGet - Microsoft Internet Explorer provided by AT&T Business Internet Services
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cpalczak:20701/indepthj2ee/ui?lcols=1&screen=current_invocation&view=All&measurement=Weighted+Response+Times&start_time=1074723860&end_time=107472
Go Links
J2EE
StartPoint Dashboard Settings Archived Data Help
Local intranet

Environment : Default Environment

Navigation > bigbank.com : Tomcat > ... > com.bigbank.ui.ChartServlet.doGet

Response Times | Weighted Response Times | Throughput | Relative Frequency | CPU | Weighted CPU | Stack Traces | SmarTune

com.bigbank.ui.ChartServlet.doGet
Mean Response Time: 4651.92 ms

January 21, 2004 3:39:00 PM MST - January 21, 2004 3:41:00 PM MST (2 minutes)

Invocation	Weighted Response Time
com.bigbank.ui.BaseServlet.processTransformation	84.17 %
com.keypoint.PngEncoderB.pngEncode	7.06 %
java.awt.image.BufferedImage.setData	3.06 %
com.klg.jclass.chart.data.JCXMLDataInterpreter.interpretSeries	2.51 %
Work Time	2.37 %
java.lang.Class.getResource	0.23 %
java.awt.Graphics.fillRect	0.19 %
java.awt.Graphics2D.drawString	0.15 %
java.text.SimpleDateFormat.getDateFormatSymbols	0.13 %
com.klg.jclass.chart.ChartDataView.getChartStyle	0.07 %
com.klg.jclass.chart.JCChartUtil.calcTik	0.02 %
com.klg.jclass.chart.ChartDataViewSeries.getYMinMax	0.01 %
com.klg.jclass.chart.ChartInteriorRegion.setChanged	0.01 %
com.klg.jclass.chart.ChartText.calcSize	0.01 %

This page was generated March 1, 2004 10:13:55 PM MST

Figure 3: After Adaptive Instrumentation – methods have been exposed

After Adaptive Instrumentation, visibility is greatly improved. The generic Work Time contribution has been reduced from 100% to less than 3%. The most significant executing methods have been exposed and the contribution of each method to overall performance is revealed.

USING MULTIPLE SURVEYS

The ability to run an automatic survey of an executing application is very powerful and paves the way for newer and more sophisticated ways of using instrumentation. Some possibilities are:

- Fine-Tuning - Experiment with different overhead budgets to further fine tune the level of instrumentation. Users can run multiple surveys with different budgets and find the best balance between detail level and amount of overhead.

- Multiple Instrumentation Sets - Use different instrumentation sets for different times since applications may behave differently at different times. For example, there may be a difference between the behavior of an application during the week and its behavior during weekends or even differences between morning and afternoon peaks within the same day. Users can use Adaptive Instrumentation to run multiple surveys and apply different instrumentation sets at different times.

- Pre-Survey - Run survey in a pre-production environment and apply the results to the production environment. In a staging environment, users can run the Adaptive survey in the pre-production system with different budgets until they are satisfied with the instrumentation set. The results are then copied to the production environment and readily used there.

SUMMARY

Adaptive Instrumentation is a new patent-pending state-of-the-art technology that allows developers, administrators and performance analysts to maximize their visibility into applications while minimizing application overhead by automatically finding the right balance between the two. It allows you to:

- Automatically discover and isolate the “hot spots” in a J2EE application’s business logic context
- Instrument methods without restarting applications
- Protect yourself against over-instrumenting applications

Adaptive instrumentation is a key component of VERITAS Indepth for J2EE, a comprehensive performance management solution for J2EE applications.

APPENDIX A -- GLOSSARY

Byte Code	The 'machine language' interpreted by the **JVM.**
Byte-Code Instrumentation	**Byte code,** variables, and methods inserted into a Java class to collect information on or modify program behavior.
Hot Spots	Methods in an application that are the highest contributors to the overall performance of the application
Invocation Count	The number of times a method was invoked by other methods.
JVM	The Java Virtual Machine, a program which executes **byte code**. Sometimes referred to as VM.
Overhead	The amount of time spent on gathering and recording performance data versus executing business logic.
Over-Instrumentation Protection	The ability to automatically disable instrumentation that exceeds a given overhead budget.
Response Time	The total amount of time spent inside a method or an application.
Work Time	The amount of time spent by a method or an application executing, excluding calls it made to other methods/services.

VERITAS Software Corporation
Corporate Headquarters
350 Ellis Street
Mountain View, CA 94043
650-527-8000 or 866-837-4827

For additional information about VERITAS Software, its products, or the location of an office near you, please call our corporate headquarters or visit our Web site at www.veritas.com.