Performance Bottleneck Analysis of Web Applications with eASSIST

Tomohide Yamamoto†*, Yasuharu Yamada, and Tetsuya Ogata*

Abstract

This article introduces eASSIST, which enables us to monitor and evaluate the performance of each internal component of an e-Business application consisting of a WWW server application and Java components without needing the application's Java source code. It helps us to determine the application's performance bottlenecks easily.

1. Importance of bottleneck analysis of Java applications

Applications for electronic business (e-Business), such as online buying and selling on the worldwide web (WWW), have become more and more complicated. Moreover, they must have sufficiently high availability to process customers' orders and their transactions all the time. However, the application development cycle from planning to the start of service has become shorter and shorter. In particular, when the development and test phases are short, an application's performance will often not be tested sufficiently because the implementation for the application's required functions accounts for almost all of the time in these phases. Sometimes an increase in the number of customers using applications for online buying and selling, for example, will cause their performance to degrade or will create other problems that will adversely affect e-Business companies.

To avoid this, we should determine as many as possible of an application's performance bottlenecks as quickly as possible in the development and test phases and fix them before the service starts. Moreover, in the operation phase, we should observe the tendency of its performance and determine performance bottlenecks and fix them before the application's performance degrades much [1].

These days, many e-Business applications are written in Java [2]. Java is very useful for developing large applications of this type because it is not only a computer language, but also an open, integrated enterprise infrastructure software system and a platform for network computing. Therefore, bottleneck analysis of Java applications has become more important.

2. Conventional methods of bottleneck analysis

The simplest and most common method of bottleneck analysis is to insert monitoring code into an application's source code. This records various kinds of information useful for debugging and understanding its behavior during execution. This method is popularly known as "printf debug" in the application development and test cycle in the C programming language and similar programming languages. If you use methods like "printf debug" to determine an application's performance bottlenecks, you must edit its source code and manually insert monitoring code at each potential bottleneck. However, if you cannot guess where the bottlenecks will be, then this operation can become huge and hence slow.

Another method uses a kind of application called a "profiler", which can automate the editing. The profiler records the execution process of the target application and analyzes its behavior from the execution records. Generally, the profiler needs a specific exe-

† NTT Information Sharing Platform Laboratories
Musashino-shi, 180-8585 Japan
E-mail: yamamoto.tomohide@lab.ntt.co.jp

cution environment, and the target application runs much more slowly with the profiler than without it. So it is difficult to determine the application's bottlenecks with a profiler during actual operation or during the test phase simulating actual operation.

With the conventional methods, besides the problems mentioned above, performance bottleneck analysis must be done by application developers who are familiar with the application's specifications.

3. Solution for application performance management and monitoring: eASSIST

To solve these problems, eASSIST*1, developed by NTT Information Sharing Platform Laboratories, applies a performance evaluation function and a bottleneck analysis function to each component of Java applications on the Web application server. With eASSIST, even testers who are not familiar with the application's specifications can determine performance bottlenecks quickly and in detail using the various functions described below. **Figure 1** schematically shows the concept of a service using eASSIST.

3.1 Features of eASSIST

(1) It can measure the execution time and frequency of each of the Java components and their methods composing the application with quite a low overhead.

(2) Any Java component and its method can be freely selected as a target to observe on eASSIST's graphical user interface (GUI), and eASSIST modifies the target components and methods to insert monitoring code into them without needing their source code.

(3) It can record the SQL (structured query language) queries evaluated in JDBC (Java Database Connectivity).

(4) It can record and display the application host's information (CPU, hard disk drive, and memory consumption rates, etc.).

(5) Its bottleneck analyzer can display all Java components and their methods that have a long execution time. It can also display the sequences in which those components and methods are called.

(6) In measuring the execution time and frequency of Java components and their methods, eASSIST can set thresholds for each Java component and method. If a monitored value exceeds the threshold, eASSIST can display this infor-

*1 eASSIST is not a registered trademark, but only our code name.

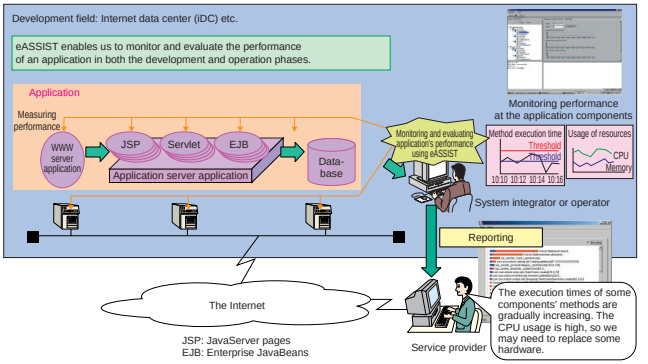

Fig. 1. Concept of service using eASSIST.

mation on its GUI and notify appropriate people by email and by SNMP (simple network management protocol).

The first two features are the most fundamental and important functions of eASSIST to determine the application's performance bottlenecks quickly and effectively. Feature (1) is designed not only to measure the execution time and frequency of each Java component and the component's method with quite a low overhead, but also, if eASSIST stops running, to ensure that the application's execution is not affected. This feature is important in the real operation phase. Features (3), (4), and (5) determine which Java components and their methods could be performance bottlenecks. Feature (6) lets eASSIST cooperate with other monitoring tools using SNMP in the real operating phase. It is useful for integrated observation of applications during actual operation.

The eASSIST clearly shows us which components or which of their methods consumed a lot of execution time and which were called frequently, after its bottleneck analyzer has examined eASSIST's various logs. Of course, it is not always true that the parts that are called many times or have a long execution time are the applications' bottlenecks, but they are candidates. To identify the bottlenecks, we need more thorough analysis because an application's performance degradation can be caused by not only the application's specifications but also by, for example, a lack of CPU power or memory in the application host. The eASSIST supports us in choosing candidates quickly and helps us to identify the serious application bottlenecks by supplying various kinds of information, for example host information, to analyze the bottlenecks in detail.

4. Example of using eASSIST for web application bottleneck analysis

Let us look at a simple example for the performance bottleneck analysis of a Web application with eASSIST.

First, investigate the Web application's execution code with eAnalyzer, one of eASSIST's tools. Then, based on the results, choose which Java components and methods to monitor and compose a registration file to use for registering components in eASSIST. **Figure 2** shows the Java components and their methods listed in eAnalyzer's GUI. Altenatively, you can choose Java components and methods to monitor by the categories defined by J2EE*2, such as JSP, Servlet, and Session Bean. This option is useful for monitoring specific kinds of components and methods in a Web application.

Next, launch the target Web application. Measuring code for eASSIST can be inserted into the Web application's execution code by eAnalyzer statically during the above investigation or dynamically while the application is running. In both cases, the application's source code is not needed. The insertion is done automatically by eAnalyzer.

Then, to measure both the external performance (for example turn-around time) of the Web application using a load testing tool and its internal performance (for example each component's execution time) by eASSIST, you perform the performance test with eASSIST and commercially available load testing tools. In measuring an application's performance with eASSIST, each component or its method can be monitored and displayed on eASSIST's GUI in real time. **Figure 3** shows Java components and their methods displayed in eASSIST's GUI. During this performance test, you may not be conscious of eASSIST, which runs in the background.

After the performance test, you start analyzing the application's performance bottlenecks after instructing the bottleneck analyzer to read the eASSIST's logs acquired in the performance test. Then you check what calls the execution sequence made using

*2 J2EE (Java2 platform, Enterprise Edition) is a specification for enterprise Java applications. JSP (JavaServer Pages), Servlet, Session Bean, and EJB (Enterprise JavaBeans) are terms defined by J2EE.

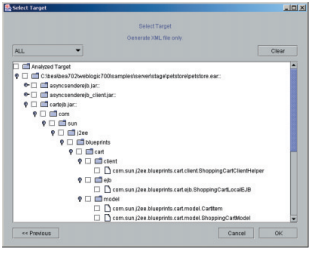

Fig. 2. Java components and their methods listed in eAnalyzer's GUI.

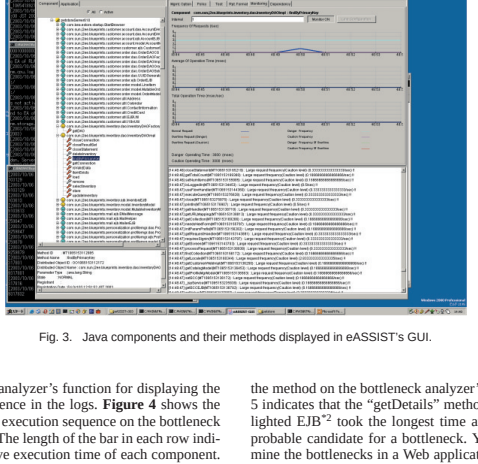

Fig. 3. Java components and their methods displayed in eASSIST's GUI.

the bottleneck analyzer's function for displaying the execution sequence in the logs. **Figure 4** shows the top level of the execution sequence on the bottleneck analyzer GUI. The length of the bar in each row indicates the relative execution time of each component. Figure 4 indicates that the highlighted Servlet, whose method name is "doGet", took the longest time. **Figure 5** shows the subordinate sequences executed in the method on the bottleneck analyzer's GUI. Figure 5 indicates that the "getDetails" method in the highlighted EJB*2 took the longest time and that it is a probable candidate for a bottleneck. You can determine the bottlenecks in a Web application more precisely in this way, by recursively investigating subordinate Java components that account for most of the execution time of Java components that could be the

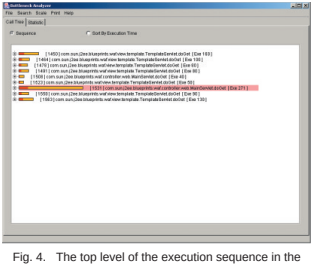

Fig. 4. The top level of the execution sequence in the bottleneck analyzer's GUI.

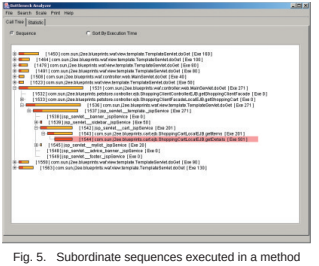

Fig. 5. Subordinate sequences executed in a method in the bottleneck analyzer's GUI.

bottlenecks. After determining the bottlenecks, you examine the source code of the bottleneck candidates one by one and fix them to eliminate the bottlenecks.

In conclusion, eASSIST helps us to determine Web application bottlenecks quickly and should enable us to improve the quality of Web applications quickly, which is appropriate for a service with high availability.

5. Future development

Because eASSIST does not automate the whole process of performance bottleneck analysis and we must utilize our experience of bottleneck analysis, we consider that it is important to accumulate performance bottleneck analysis know-how and to incorporate this into eASSIST as expert knowledge.

References

[1] H. Tanaka, Y. Yamada, A. Gotou, and H. Ishii, "Providing high quality e-Business applications," NTT Technical Journal, Vol. 13, No. 2, pp. 44-48, 2001 (in Japanese).
[2] URL: http://java.sun.com/

Tomohide Yamamoto
Engineer, Application Platform SE Project, NTT Information Sharing Platform Laboratories.
He received the B.E. degree in mechano-informatics and M.E. degree in information engineering from the University of Tokyo, Tokyo in 1993 and 1995, respectively. He joined NTT Laboratories in 1995. He had been researching protocols and developing systems for charging for digital contents safely and correctly for several years. He is now developing various Java-based tools related to enterprise application management and Web services.

Yasuharu Yamada
Research Engineer, Application Platform SE Project, NTT Information Sharing Platform Laboratories.
He received the B.E. and M.E. degrees in electronic-mechanical engineering from Nagoya University, Nagoya in 1992 and 1994, respectively, and joined NTT Laboratories in 1994. He has been developing network management systems using SNMP and researching TINA-based network management systems for several years. He is now developing various Java-based tools related to enterprise application management and Web services.

Tetsuya Ogata
Engineer, Application Platform SE Project, NTT Information Sharing Platform Laboratories.
He received the B.S. degree in physics from Yokohama City University, Yokohama in 1994. He joined NTT in 1994. He is developing various Java-based tools related to enterprise application management and Web services.