Holistic Performance Analysis of J2EE Applications

By Madhu Tanikella

In order to identify and resolve performance problems of enterprise Java Applications and reduce the time-to-market, performance analysis needs to be holistic and comprehensive across different tiers and application layers. Enterprise performance monitoring tools help to get and analyze the metrics of server resource usage, application code and Database SQLs in a unified view thereby making the bottleneck identification faster and easier.

In the current dynamic market conditions, IT organizations would like to develop and deploy their business-critical applications as quickly as possible to meet their business objectives and win over the competition with a specific focus on performance, scalability and availability requirements. In order to achieve the required performance and scalability of applications in the constrained timelines, the turn-around time to identify and fix the performance issues during development cycle should be minimal.

In order to carry out holistic performance analysis across the technology stack and application layers, there is a constant need to monitor the performance metrics across Web, Business and Database Tiers at Operating System, Middleware and Application layers such as Java Code and SQL code thereby reducing the time taken to identify and resolve performance bottlenecks. In this performance journey, enterprise performance monitoring tools that provide a single unified view of metrics in a dashboard, can be harnessed as a good aid for faster performance monitoring and analysis.

The objective of this white paper is to share my experience on performance analysis for a Java/J2EE enterprise application and illustrate the steps to quickly analyze performance metrics at OS level, Java Code level and Database SQL level and identify performance issues, with the help of an enterprise monitoring tool.

The illustrations and data points used in this white paper to support the performance analysis steps are taken from a performance assessment carried out on a Java/J2EE application for one of the leading financial firms in USA.
WHY COMPREHENSIVE PERFORMANCE ANALYSIS OF J2EE APPLICATIONS?

As any J2EE based enterprise application typically has multiple layers such as Web, Application, Database and EIS tiers, it is necessary to ensure that the components in each tier are performing well so as to achieve overall performance of the application. For instance, if the overall response time of a Business Process in a J2EE application is very high, it is required to find out the Layer in which the time spent is high so that the software component or code in that Layer needs to be focused for any optimization in terms of resource usage and processing.

Typically, when a business process in J2EE application is executed, it makes use of the OS resources such as CPU, Memory, Disk and Network, Application Server resources such as JVM Heap, Threads, and Application Resources such as Java Classes and Methods and Database Resources such as Memory Pools and SQL Queries. Hence granularity of performance metrics right from a problematic Java business process all the way to the poorly performing Component, Method and SQL Queries provide development teams with actionable results, thus reducing the cost and time required to optimize applications. Figure.1 provides the list of performance metrics that need to be monitored, analyzed and correlated for holistic and comprehensive performance analysis of any enterprise J2EE application.

![Tier-wise Performance Metrics](image)

**Figure.1. Tier-wise Performance Metrics**

**LEVERAGING ENTERPRISE CLASS PERFORMANCE MONITORING TOOLS**

Enterprise monitoring tools, in general, can be perceived as a combination of a code profiling tool and a monitoring solution that captures performance data related to application code as well as resource utilization using a ‘Probe’ or ‘Agent’ installed on the Server that should be analyzed. Performance data collected from the probe/agent can be configured to persist to a database that can be used for
offline analysis. For instance, an enterprise class monitoring tool can persist data for different time periods, such as a day/month/3 months and so on depending up on the configuration chosen by the administrators.

In general, enterprise class monitoring tools provide a mapping between Method Calls and different Application Layers. This feature makes identifying a bottleneck layer of an application easy and quicker - just by clicking on a server request, the time spent in different Layers will be listed. This helps the users of the tool not to spend much time in custom instrumentation rather straightaway focus on performance analysis.

**HIGH LEVEL DEPLOYMENT ARCHITECTURE OF THE J2EE SYSTEM**

Before moving into the detailed step by step performance analysis of J2EE application, I would like to provide a brief overview of the deployment architecture.

As shown in Figure.2, the application comprises J2EE based webservices hosted on IBM WebSphere Application Server (WAS), that interacts with IBM’s Information Content management software hosted on WAS – it extracts and manages the content from multiple external systems using Java Adapters for different protocols. The application in itself does not have any Web UI however it is integrated with a portal that is external to it.

![Figure.2. Deployment Architecture – J2EE Application](image)
HOW TO IDENTIFY PERFORMANCE ISSUES OF J2EE APPLICATIONS ACROSS LAYERS?

In this section, the detailed steps of analyzing the performance metrics of J2EE application at different levels (as mentioned in previous sections) is illustrated with snapshots and data points captured.

As mentioned earlier, performance of a Java Business Process can be drilled-down to different layers such as JSP/Servlets, EJB, JDBC and SQL queries.

The different performance metric views can be broadly categorized into the following which are required to nail down to the application component or code that is the cause of the performance issue:

HOSTS METRICS - This refers to Physical Server machine and Operating System on which WebServer/Application Server/Database Server/EIS are hosted and the performance metrics of interest are the utilization metrics of the Server Resources - CPU, Memory, Disk and Network.

![Figure.3. Host Metrics](Source: HP Diagnostics)

Figure.3. illustrates that average CPU Used of the server machine is around 30% and the Physical Memory Used (Avg.) averages to around 37% for the given duration. Besides, this view can be customized to include detailed OS level performance counters such as Context Switches, Disk Reads/Writes, and Network Reads/Writes as highlighted in Figure.4 below.

<table>
<thead>
<tr>
<th>Status</th>
<th>Color</th>
<th>Host</th>
<th>CPU % Over Threshold</th>
<th>CPU</th>
<th>Context Switches/S</th>
<th>Disk Bytes/S</th>
<th>Network Bytes/S</th>
<th>Page In/S</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Data</td>
<td></td>
<td>hostscoreactin</td>
<td>m/a</td>
<td>17%</td>
<td>1,122</td>
<td>155.6 KB</td>
<td>15.7 KB</td>
<td>0</td>
<td>37%</td>
</tr>
</tbody>
</table>
MIDDLEWARE METRICS – This represents the performance metrics of middleware such as Application Servers in general that will be hosted on a given OS of a physical server – the 2nd common layer in the technology stack. In the context of this paper, it refers to IBM WebSphere Application Server’s performance metrics that are captured using java probes configured for the required Application Server instances – depending on the deployment architecture of the application, the # of java probes configured is equal to the # of WAS instances. Figure 5 below shows that the physical server host1.nsroot.net.in has 4 WAS instances running and hence 4 different Probes are configured.
The above view shows the performance metrics of IBM WAS, such as JVM Heap Used, Thread Pools, JDBC Pools, EJB, Avg. Request Latency and Normalized CPU Usage (the % CPU used by the java process averaged across the CPUs). This will provide a bigger picture about the WAS instance or the Java process performance characteristics for single or concurrent user load tests. This view in turn is the entry point to further drill down to see the # of requests coming into the Server and the corresponding Average Latency.

Figure 5. illustrates that CPU usage of all the 4 WAS instances together is in the range of 20-25% indicating that the overall CPU Usage on the server is approximately close to the CPU usage of all the 4 WAS instances together.

Figure 6. shows other WAS metrics that can be chosen in Probes View to further understand the instance behavior.
SERVER REQUEST METRICS – These metrics further enable the performance architects/analysts to identify the top ‘time consuming’ server requests and the corresponding # of invocations.

Normally an Enterprise class monitoring tool provides the details of server requests metrics for a given duration as shown in Fig.5. (for instance, WEBSERVICES_PROBE1).

Figure 7. below shows the Server Request View during a 200 concurrent user load test for the application stated above. The most important and frequently used columns of this view are Server Request, Latency (Avg.), Count and Exceptions. It shows that request1() and request2() method calls are topping the list – whereas, other method calls are pertained to IBM WAS maintenance activities that run in background and hence they need not be considered.
CALL PROFILE DETAILS – In order to find out the java method that is the cause of the performance bottleneck for a Java Business Process, the call profile details of the corresponding server request need to be analyzed.

In Figure.7. triangle and circle are highlighted that represent Latency Average and Latency Maximum respectively for request2(). By clicking on either of them, the call stack for the server request will be displayed in detail as shown in Figure.8. below.
In Figure 8, the java method calls that have high exclusive method time i.e., time spent in the method excluding its calling methods, for `request1()` are highlighted. Out of 12 min 43 sec of total latency, the exclusive latency of 2 hotspot methods took around 11 min 6 sec and 1 min 28 sec respectively. Converting it to relative percentage time spent, it comes to around 87.3% and 11.6% respectively, indicating that these are the 2 performance hotspots for `request1()` that should be focused for tuning and optimization.

After analyzing the functionality and the java code of the hotspot methods, it is identified that obtaining a connection in IBM’s Content Management is an issue in terms of response time (i.e., `ConnectionPool.getUser()`), and it is fixed by getting a patch from IBM (as it is fixed in the product’s next version).

**SQL STATEMENT METRICS** – Besides the performance details of java code and its method calls, insights into the time spent and # of invocations of SQL Queries and Stored Procedures executed on Database is also of valuable information in comprehensive performance analysis of J2EE applications.

Enterprise performance monitoring tools provide performance details about database SQLs, They typically show the SQLs that are executed from the java code, the method from where it is called and the # of invocations – they capture this information from the arguments of the JDBC query execution method calls as these tools, in general, use a java probe configured on Application Server instance. Hence, this information can be viewed along with the Server Request metrics thereby providing a unified view across the different application layers.

Figure 9 shows the SQL Statements that get executed from the Server Requests and their execution time. For instance, if Call Profile of a server request shows that JDBC call is taking most of its time, the corresponding SQL Query can be obtained from this view.
However, in the application under consideration, no SQL queries are found to be expensive or attributing in high percent to the overall response time of the business transactions.

Thus, a single unified view of the performance data and comprehensive analysis across Web Layer (i.e., JSP and Servlets), Business Layer (i.e., EJBs and POJOs) and Data Access Layer (i.e., JDBC layer) gives a holistic view of the system’s performance and also reduces the time to identify and resolve performance bottlenecks effectively. Enterprise monitoring tools normally facilitate this process by providing an easy-to-use and intuitive dashboard.

**ANALYZING LOAD BALANCING ISSUES USING MIDDLEWARE METRICS**

As the deployment architecture of the application makes use of BIG IP Load Balancer, it is essential to ensure that requests are evenly distributed between the 2 WAS nodes. The number of server requests hitting the each WAS node provides insights about any load balancing issues of the system.

This is illustrated using the snapshots captured on 2 physical servers for a 200 user load test and comparing the number of server requests.
Figure 10 and Figure 11 show that the # of server requests for both request1() and request2() together on Node1 and Node2 respectively are 260 and 20. This clearly indicates that the LoadBalancer (BIG IP, in current context) is not distributing the requests equally between the 2 server nodes. The BIG IP configuration is reviewed and modified later to ensure that the requests are evenly distributed.
OVERHEAD OF ENTERPRISE CLASS MONITORING TOOLS ON RESPONSE TIME

An effort is made to study the overhead on overall server response time when an enterprise monitoring tool is enabled in the current environment. A single user load for a mix of 5 java business transactions is simulated for 15 min duration and response time data is captured with the tool ON and OFF.

The objective of this exercise is to make a mention that enterprise class monitoring tools used for holistic and comprehensive performance analysis to get a unified view of performance metrics across layers, come with certain permissible overhead on the server response time as they use probes that are attached to JVM.

Table 2. below shows the tool overhead averaged across 5 business transactions with exactly same test scenarios and test data.

<table>
<thead>
<tr>
<th>#</th>
<th>Business Transaction</th>
<th>MONITORING TOOL OFF</th>
<th>MONITORING TOOL ON</th>
<th>% Overhead in Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Avg Response Time</td>
<td>Txns Passed</td>
<td>Avg Response Time</td>
</tr>
<tr>
<td>1</td>
<td>Transaction1</td>
<td>25.023</td>
<td>59</td>
<td>28.164</td>
</tr>
<tr>
<td>2</td>
<td>Transaction2</td>
<td>24.908</td>
<td>71</td>
<td>27.896</td>
</tr>
<tr>
<td>3</td>
<td>Transaction3</td>
<td>22.038</td>
<td>66</td>
<td>25.822</td>
</tr>
<tr>
<td>4</td>
<td>Transaction4</td>
<td>17.032</td>
<td>81</td>
<td>20.991</td>
</tr>
<tr>
<td>5</td>
<td>Transaction5</td>
<td>16.941</td>
<td>82</td>
<td>21.22</td>
</tr>
<tr>
<td></td>
<td>Average % Overhead on Server Response Time</td>
<td>15%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It indicates that the overhead is around 15% due to use of monitoring tool which is well acceptable in test environments where the primary objective is to find out code level performance bottlenecks - however, this overhead should be considered as indicative and it might vary depending on the enterprise monitoring tool and the application nature (such as CPU intensive, MEMORY intensive etc.) and hence an exclusive exercise to find out the tool’s overhead is suggested for a different application and different enterprise monitoring tool.

CONCLUSION

The paper details out the analysis steps to identify and analyze application performance bottlenecks of a Java/J2EE application across different application layers with illustrations from one of the enterprise class monitoring tool. A holistic and comprehensive analysis of performance metrics across different application layers reduces the time to identify and resolve the performance issues.
REFERENCES
1. “HP Diagnostics software Solution brief”
   https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-15-25%5E761_4000_100

AUTHOR’S PROFILE
Madhu Tanikella is a Technology Architect working with Systems Integration group in Infosys. He has around 9 years of experience in Performance Engineering & Performance Tuning of client-server and web-based software systems developed using Java/J2EE, .Net and Oracle technologies. He has specialized expertise in the areas of NFR Validation, WorkLoad Modeling, Performance Tuning, Performance Modeling and Capacity Planning of distributed software systems. He can be contacted at madhubabu_tanikella@infosys.com