

Case Study

Tools to Solve Real Problems: How Pramati Server helps you quickly identify and resolve issues in your J2EE™ Application

Sampath M and Venu Gopal, Customer Support

In this paper, we discuss two cases of actual customer issues that were resolved through use of Pramati Server Diagnostic features. The examples illustrate the power the Server Management Console, which is not only useful during deployment or performance tuning period, but also during development and testing phases. Drilling down the analytics can give very useful information on the dynamics of runtime and also confidence that everything is working as expected. Customer names have been blanked out from the screen images to maintain confidentiality.

Case 1: Identifying Serialization in an Application

A large enterprise customer with a new eLearning application reported unusual response time during testing. On a modest load of 40 concurrent users response times for a particular operation in the application was noted to be about 11 seconds (**Figure 1**), which was unacceptable as field conditions demanded very large number of connected users. The cause of such high response times could not be identified till a detailed analysis of the diagnostic information was undertaken.

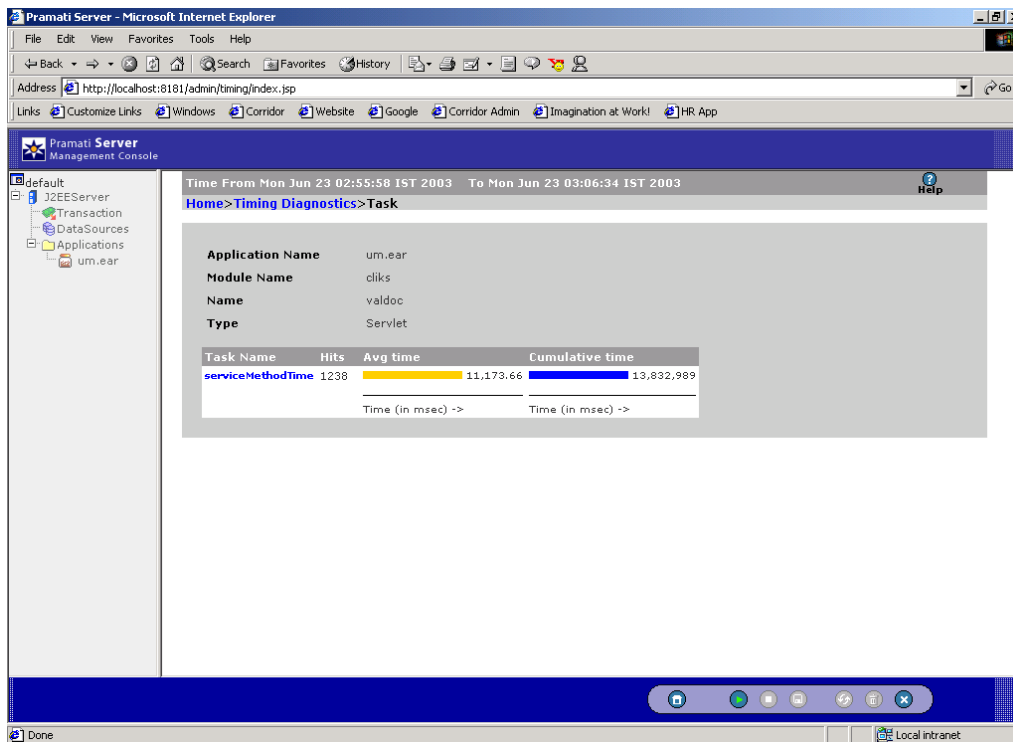


Figure 1

On first analysis there were no obvious problem in the database area as connection use time, wait time to obtain connections and reuse of prepared statements were quite good (**Figure 2**). It can be observed that the Prepared Statement is created only 38 times but reused 25,036 times during this performance run.

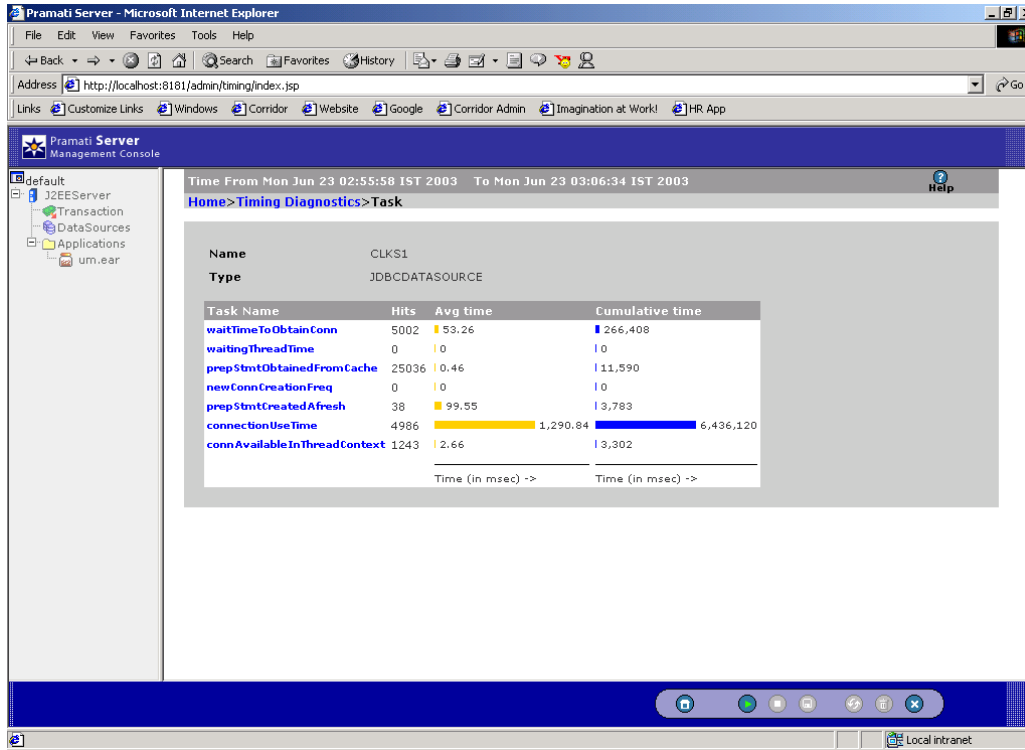


Figure 2

The distribution of response times clearly indicated that there was a possibility of execution threads queuing up for some common resource. The customer team was asked to investigate this Servlet (**Figure 3**) below for potential reasons, as the combined method calls inside consumed fairly reasonable amount on time.

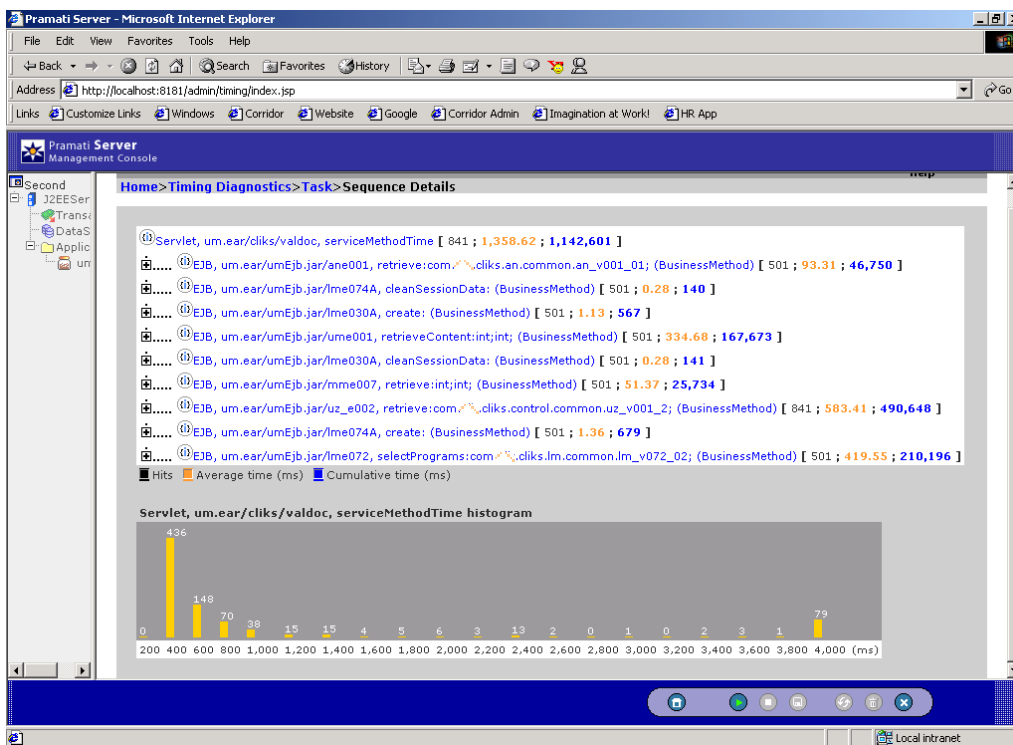


Figure 3

The thread dumps further revealed use of Hashtable a synchronized object, which was replaced by HashMap instead. This brought down the overall response time to 0.7 seconds for the Servlet, a 70% improvement.

Case 2: Optimization of Database Queries

This case is from an ISV building mission critical banking product to be deployed at Wall Street Banks. The application has a very tight specification on performance, as it performs certain time critical month-end processing for the bank and any time saved in this process is money saved.

The ISV reported that a particularly complex operation was taking up to 10 minutes on Pramati Server and had requested our assistance in exploring any optimization. On making a single threaded run with the Diagnostics on, we observed that one particular method was taking bulk of the time to execute. This was a key component in the architecture and was being reused in various other operations as well. The ISV team concurred that investigating this further would enhance performance in the entire application.

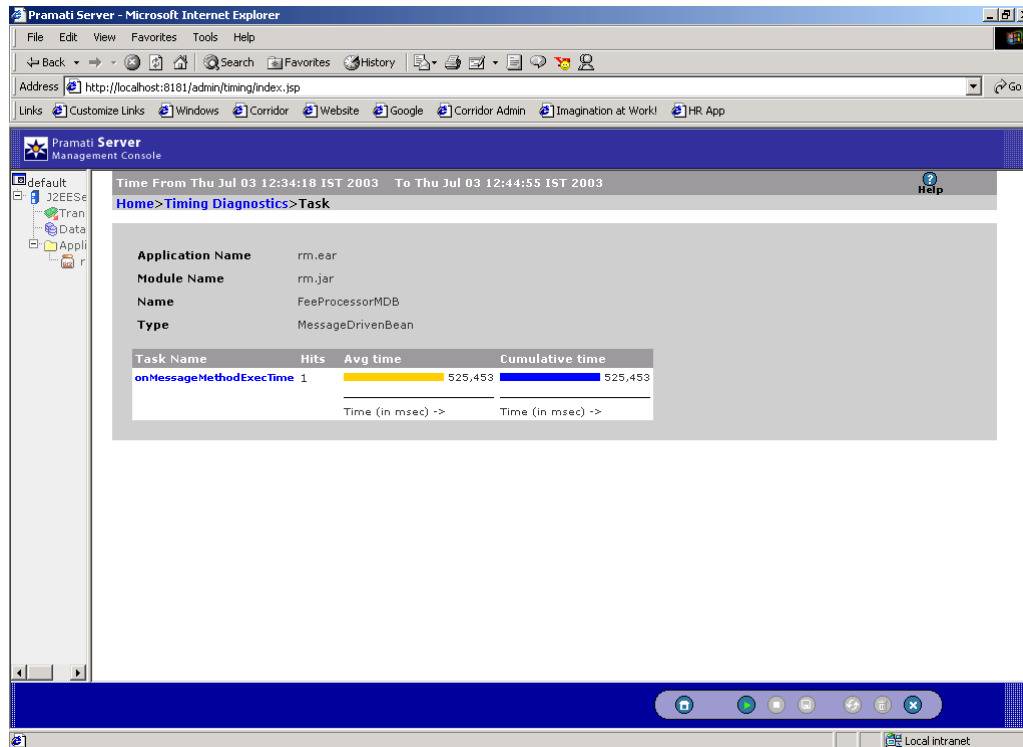


Figure 4

The problem was reported as long execution time of the onMessage method of an MDB (**Figure 4**). Further drill down reveals the actual method that is consuming enormous time (**Figure 5**).

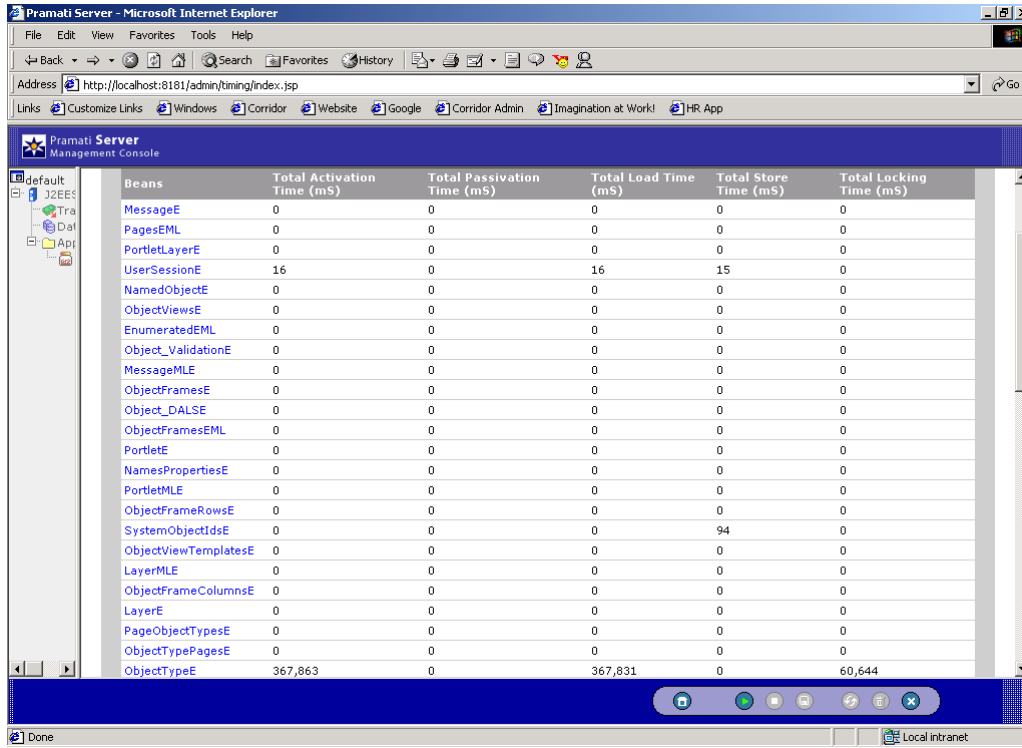


Figure 5

Further, it was observed that the load time of that component was a large proportion of the overall processing time. On investigating the load call, we found that for each execution of the overall operation, this bean was being loaded 522 times and on an average, the load was taking far more than it should given our experience (**Figure 6**).

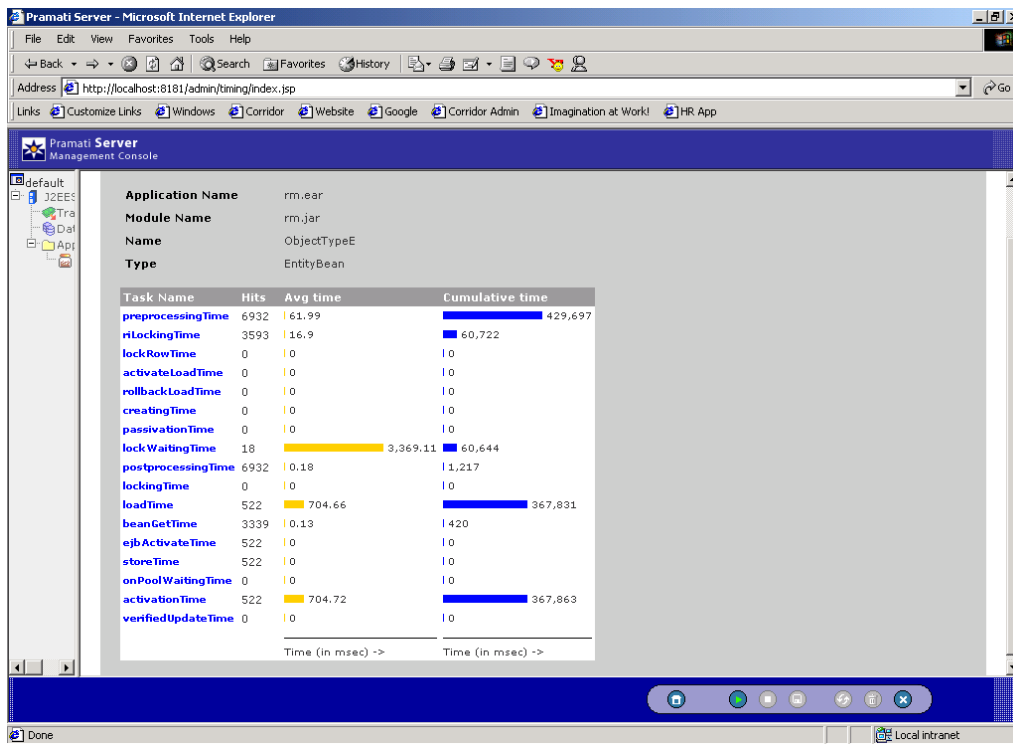


Figure 6

We narrowed the focus down to one DB query that was called 3132 times for the operation (**Figure 7**) and advised the customer to review the performance of the query as well as to examine if it could be eliminated altogether.

