IDERA® | Precise Case Study

PRECISE FOR SQL SERVER

Computer Product and Services Enterprise Fixes Critical, Failing Transactions

PROBLEM BACKGROUND

A large enterprise computer product and services company in the United States of America spent more than one year to develop a significant new version of a custom application for the Microsoft .NET Framework and Microsoft SQL Server. The development team was ready to roll out the application to production.

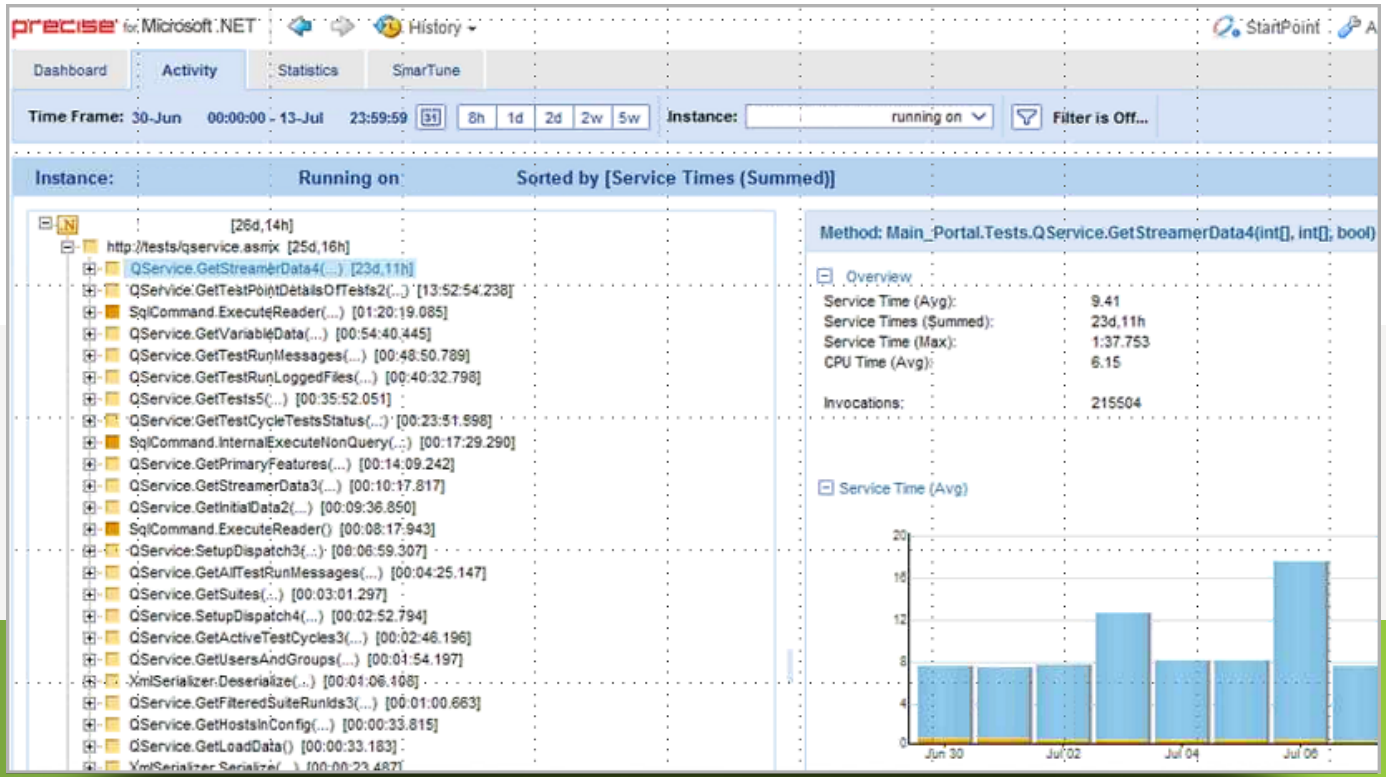
However, the team detected a significant performance problem where transactions are failing or timing out. It appeared that the CPU reached its maximum capacity. The initial response of the team was to allocate more resources in VMWare. Eventually, the team assumed that the application could not run on virtual machines but instead needed to deploy on physical machines.

The company set up a contest to try to improve the performance of the application. Fortunately, the team decided to use Precise to analyze the situation. The team planned to ask the application developers to provide different suggestions, to try out these suggestions, and to implement the best suggestion.

INVESTIGATION

The tree for the Microsoft .NET Framework on the left side of the 'Activity' workspace for the problematic instance of SQL Server shows that the method 'QService.GetStreamerData4' had a summed service time of more than 24 days that corresponds to 91% of the total time of the web service 'http://tests.qservice.asmx'.

The 'Overview' area on the right side shows that the method 'QService.GetStreamerData4' for 'Main_Portal.Tests' had an average service time of almost 10 seconds of which it spent some 6 seconds or 65% on CPU time. Consequently, the method spent a massive 368 hours for a total of more than 215 thousand invocations on CPU time. That means that not only was the CPU of the virtual machine not sufficient, but also the CPU of the physical machine would not have been sufficient.

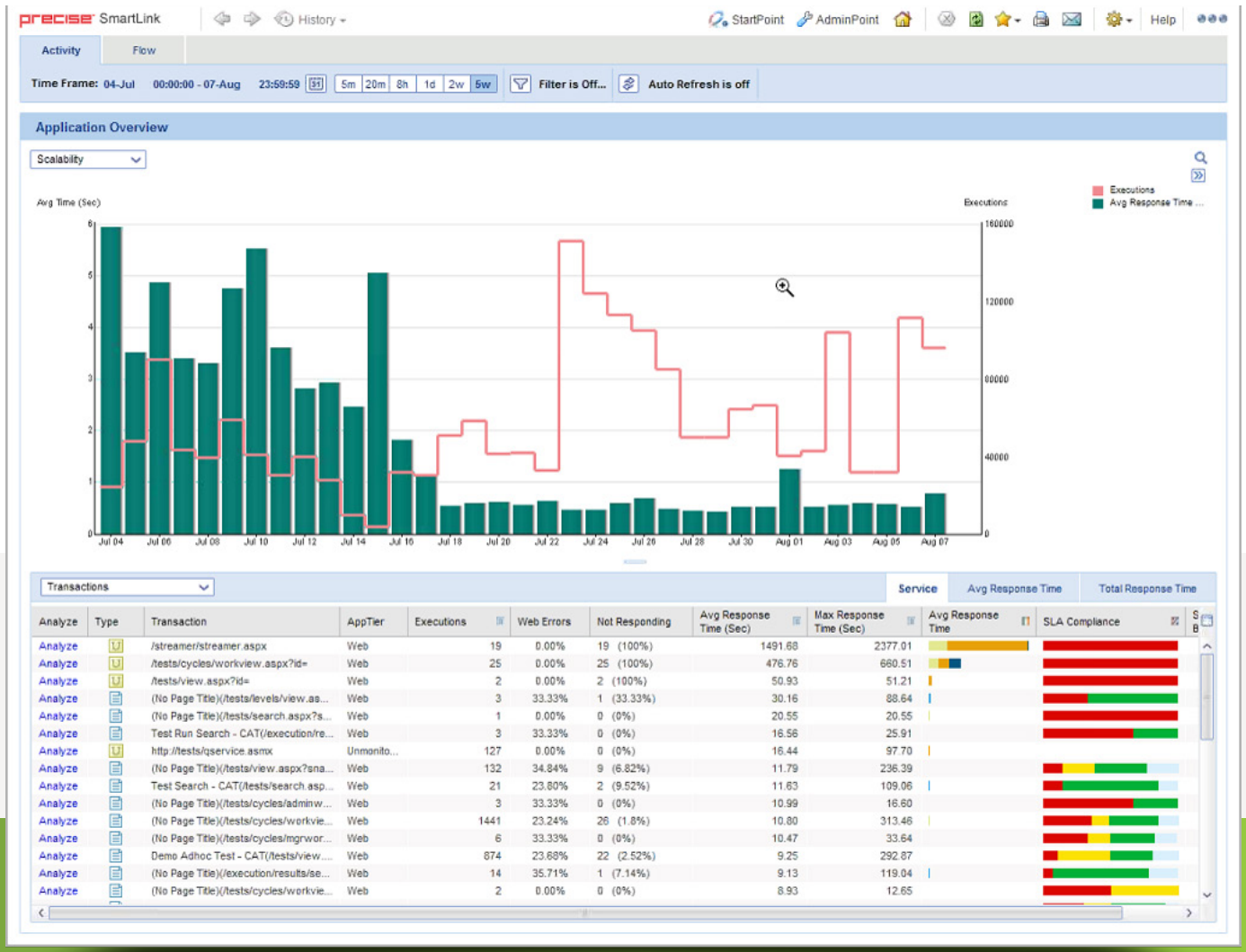


The 'Activity' workspace for the problematic instance of SQL Server.

The ‘Scalability’ area at the top of the ‘Application Overview’ area of the ‘Activity’ workspace shows that before the team implemented the changes on July 16, the method had a high average response time of some 4 seconds with a low number of executions of some 6,000 per day.

The bottom ‘Transactions’ data grid shows that the top transaction ‘/streamer/streamer.aspx’ had an average and maximum response time of some 1492 and 2377 seconds, respectively. All of its 19 executions failed.

After the team implemented the changes on July 16, the performance of the method switched to a lower average response time of some 0.5 seconds with a higher number of executions of some 6,000 to 12,000 per day.

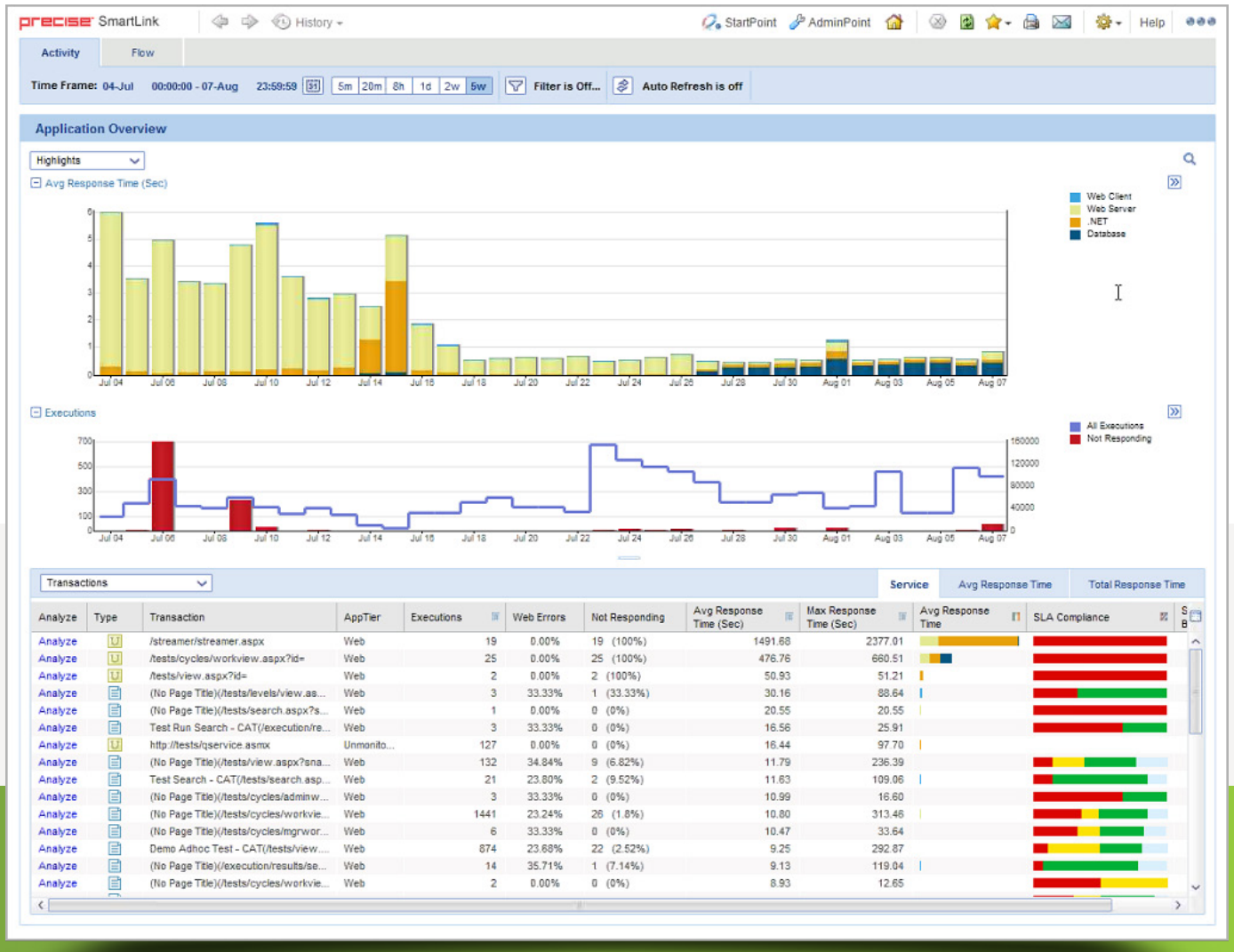


The ‘Scalability’ area at the top of the ‘Application Overview’ area of the ‘Activity’ workspace.

The ‘Highlights’ area at the top of the ‘Application Overview’ area of the ‘Activity’ workspace shows that before the team implemented the changes on July 16, most of the average response time of some 4 seconds were for the web server. The ‘Execution’ chart shows that before the team implemented the changes on July 16, there were two significant events where some 700 and some 250 executions were not responding.

After the team implemented the changes on July 16, the response time dropped to some 0.5 seconds but most of the average response time of some 4 seconds was still for the web server until July 26. After July 26, most of the average response time was for the database, with some also for the web server.

Overall, after July 26, the distribution of the average response time looks like a typical profile, with time spent in different tiers rather than only in the web server tier. The ‘Executions’ chart shows that after the team implemented the changes on July 16, less than 50 executions failed during several events.



The ‘Highlights’ area at the top of the ‘Application Overview’ area of the ‘Activity’ workspace.

FINDINGS

The company traced the performance problem to application code rather than the database using Precise. That is, the problem was due to demand and not due to supply. The company used Precise to determine which of the suggestions from the developers was the best and verified the results of implementing the best suggestion. It appeared that the initial assumption that the application cannot run on virtual machines but only on physical machines was incorrect.

SUMMARY

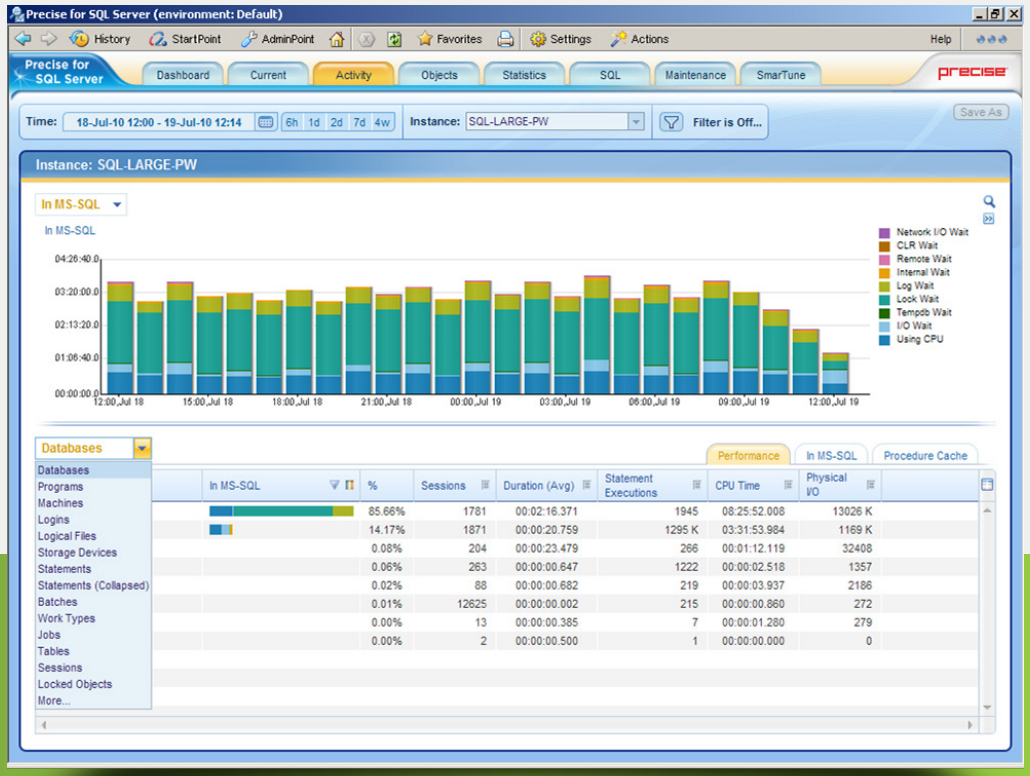
Every tier contributes to service time. It appeared that the application code created demand for resources showing how the efficiency and capacity of processing are two sides of the same coin. After the team implemented the changes, the application developers were grateful, and the team members received a bonus for meeting the go-live date, thanks to Precise.

PRECISE FOR SQL SERVER

ACCELERATE BUSINESS PERFORMANCE

- Database Performance Fuels Company Performance
- Multiple Platform Database Monitoring and Alerting
- Performance Management Database
- Root Cause Identification
- Tuning Recommendations
- What-if Analysis
- Capacity Planning

SEE IT IN ACTION



IDERA

IDERA.com