JENNIFER's Special Feature

X-View

Version: JENNIFER v2.5 and Above

Date: 2009-04-30

Author: S.J. Kim

1 Why Monitor Service Response Time by Transaction?

Monitor the response time of all transactions individually and focus on the transactions that are having performance problem and analyze the profiles

Application Performance Management is a process of measuring the performance of application services, then analyzing the collected data in order to progressively improve the performance of said application. Application performance is measured by the observing response time of the services and amount of system resource used by the services in a specific time interval. In the center of APM is profiling data for the application services, which is the detailed log or record of what was processed in the application.

What is Service?

The term Service, if used loosely, can be synonymous to term “Application”. In the world of Java EE Application, Service is an application module that is prepared to receive and process HTTP request from an end-user. For example, if a user requested the website www.website.com, the Servlet or JSP that resides in the server is “Service” and the name of the service is the URL. On the other hand, the scheduler that runs in the background is not considered as service. Thus, service performance is typically measured by observing the number of hits per URL and response time associated with each of them.

Common problem with measuring performance is the quantity of services. In a typical enterprise-class application, there can be hundreds or thousands of service, each constantly and continuously processing user requests. Thus the biggest concern about service performance measurement is how to best collect, store, and express the data.

A standard approach is Grouping-by-Task. Service names or processes that completed service is grouped by tasks or jobs and number of performance services and average response time of services are calculated, and displayed in graphs and tables. However, Service grouping can often lead to dangerous trap of averaging. If one or few critical services are suffering from performance problems, the problem can be buried under data from averages of rest of the services.

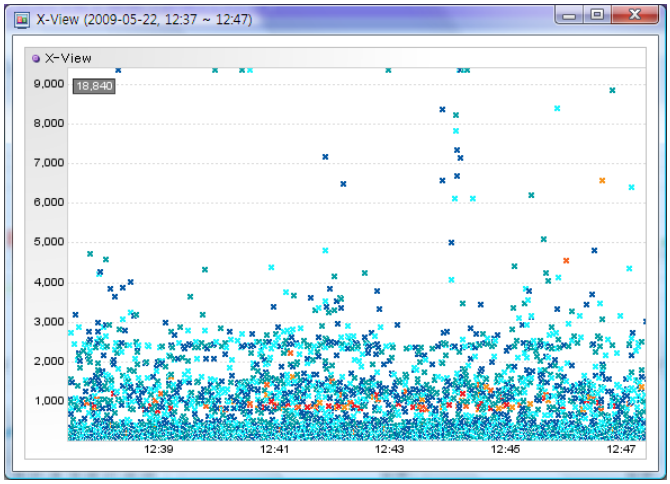
To combat this problem, some administrators started monitoring mission critical services individually and measure performance of remaining services in groups, but still there is remains problem with averages and also administrator much choose the specific services to monitor by predicting which services will have performance problem.

No matter how the services are grouped, the problem of averages will persist. Even within the same application, some service can be fast and some slow depending on special circumstance such as system status or logic problem

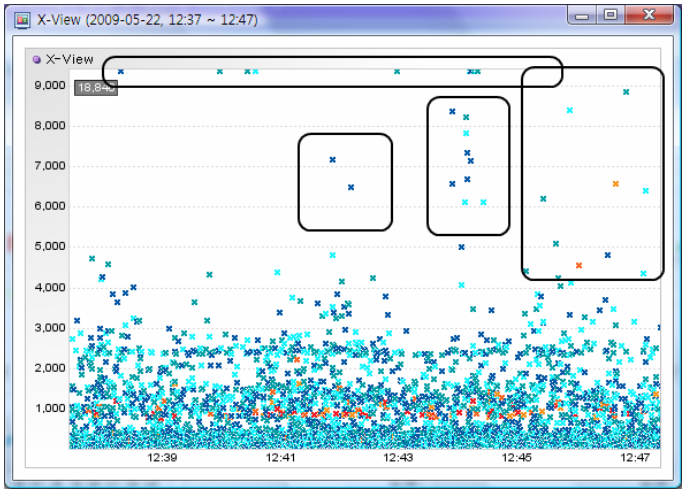
When performing simple reporting for SLA or status checking purposes, average value is sufficient, but when collecting and analyzing service data for the purpose of optimizing and improving application performance, all transactions must be monitored and measured individually.

If the specific transaction cannot be identified, the time and resource it takes to identify troubled service and finding root cause can be long and tedious.

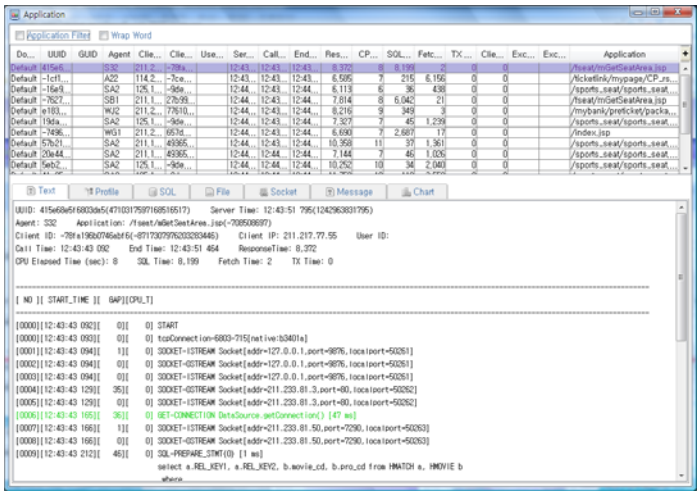
Problem with monitoring large quantity of service transaction data is how to organize and view them in a understandable way. Using a traditional line graph is only good for average data, thus useless in monitoring mass quantities. Only way to display them effectively is to use scatter-plot graphs. When expressed in scatter graph, administrator can see overall status of service transaction in one graph without loosing the granularity of the performance data. Take a look at the example of how performance data for individual transaction can be displayed in scatter-plot graph format.



The Y-axis represents the response time of individual transaction. X-axis represent the time when each transaction has finished being processed. Using this graphic format, administrator can intuitively perceive the status of all transaction individually and quickly identify specific service transactions that are suffering from performance problems.



The service transactions marked with boxes are predicted to be experiencing performance problem and need to be analyzed further.



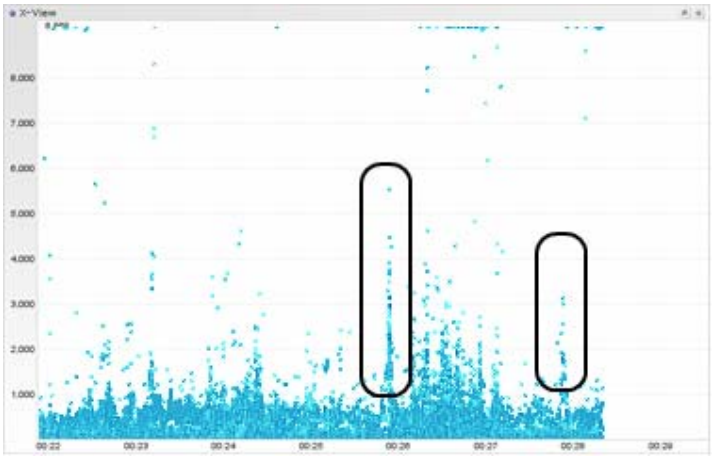
Once problematic transactions are identified, administrator observes the resource usage data and profiling data for each selected transaction to find out the root-cause of the performance problem and resolve the performance issue. Linking the transaction in the scatter graph and associated performance data such as resource usage and profiling data and critical for application performance management. A tool that can do this quickly and without high overhead to the monitored system is of great help in managing application performance.

2 Patterns in the Scatter Graph

Another advantage a scatter graph can bring is that the performance problems often show itself as patterns in the scatter graph. Different patterns in the graph often indicate specific performance problem and experienced administrator can often identify the performance problem simply by observing for specific patterns formed in the scatter graph.

2.1 Vertical or Spike pattern.

An application requires many different type of resource to run and the relationship between one resource and another is also very complicated. Typically, physical resources, such as CPU and Memory, are wrapped by software module and then wrapped resources are used by the application. When one or more of the resources that application needs is unavailable, the application is thrown into stat of “LOCK”. For example, a database lock occurs when the connection pool in the database is all occupied. When a LOCK occurs, all transaction that requires the said resource goes into “Waiting” state. One by one, transaction action take longer time to finish while waiting for the necessary resource to be available. Once the resource becomes available, all transaction will at once be processed, forming a vertical or spike-like pattern in the scatter graph.



When administrator find spike pattern in the scatter graph, he/she can look through the resource and profiling data of transaction in the spike and analyzing to identify which resource’s shortage was responsible for LOCK state. By increasing the available resource or adjusting how they are allocated, administer can quickly removed the cause of delayed service.

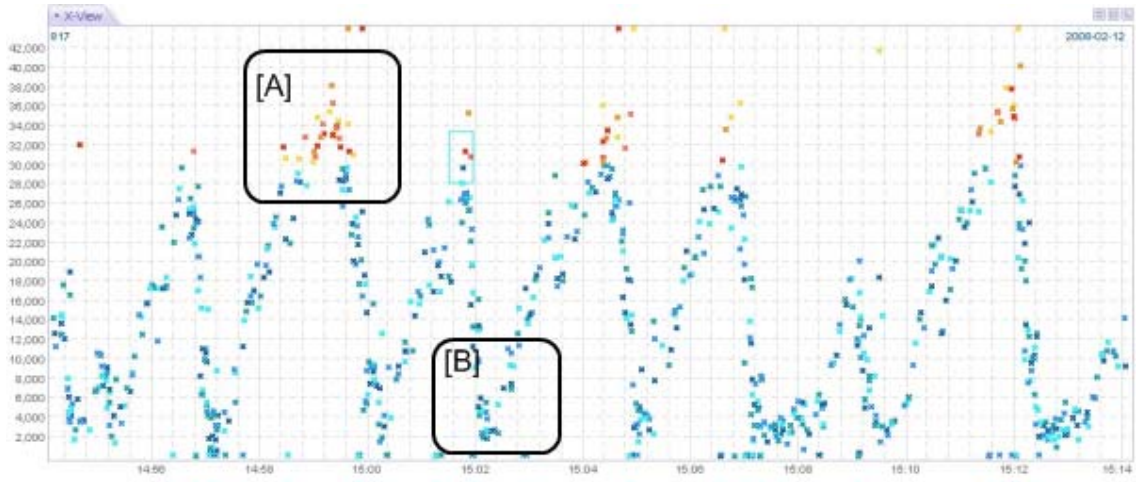
If the transactions in the spike pattern all originated form one application server, the shortage of resource likely cam from internal source. If they came from multiple different application servers, then the shortage likely came from external source unless different servers suffer from exactly name resource problem (which is less probable).

If transactions in the pattern were originated from same service, the shortage is in one of the resources specifically used by that service. If many different services are identified, then the resource shared in common is the culprit.

2.2 Wave Pattern

Not all resource shortage may result in a LOCK state.

Often, resource shortage may result in phenomenon where the response time of services fluctuates repeatedly. See the below picture. As you can see, shortage in resource has resulting in a wave like pattern in the scatter graph.

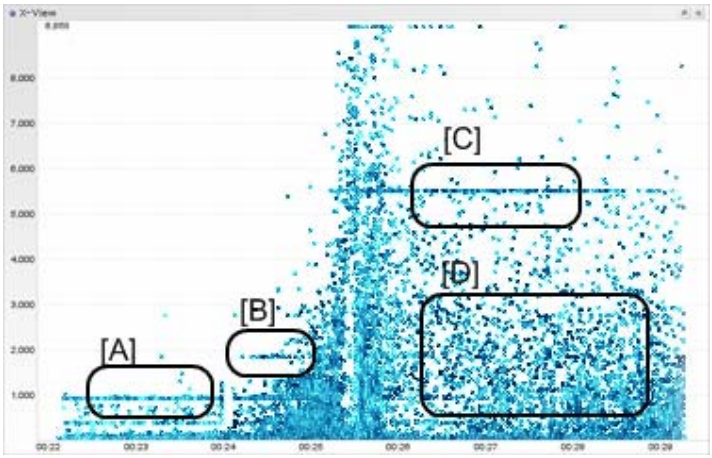


By analyzing the transaction in both peaks of the wave, administrator can identify the common resource that is causing fluctuation in response time.

For most standard OS monitoring tools, resources are often measure as percentage used, from 0% to 100% from the available pool. Problem here is that this way of measuring only shows supply of resource but not demand. For example, a given application's throughput requires 300% of CPU utilization that a server can provide. Only when that server increases its CPU resource by 3x, the application can run properly without experiencing performance problem. According to the OS monitoring, CPU utilization will always show 100%, but it cannot show how much more is needed in order to fulfill the application's resource requirement.

2.3 Horizontal or Flat-line Pattern.

Administrator may put a timeout setting for any given processes to prevent the LOCK or delay in service. This timeout setting is useful, but ill-configured timeout setting can cause trouble when they are set too short.



These ill-configured timeout setting can be seen in a response time scatter graph, via flat-line pattern. Take a look at the above picture. You can see 3 different instances of patterns. Line A has formed after initial timeout setting set at 10 seconds. Then line B forms, these are same transaction that formed line A restarted, then timed out after another seconds. Line C is formed by another group of transactions which timed out at 5.5 seconds. Once the resource problem is resolved, all transaction returns to normal state shown in area D.

As we have seen in this article, resolving performance problem is often game of finding commonality between failing services. Scatter graph provides administrator a easy and powerful way to organize, perceive, and manage performance problem in a mass of service transactions.