Resource Monitoring During Performance Testing

Experience Report by Johann du Plessis

Introduction

During a recent review of performance testing projects I completed over the past 8 years, one of the goals was to compare the most successful projects with the least successful ones. The idea was to establish what made the successful projects so good and the other projects not so good. One of the items that repeatedly came up was monitoring. Monitoring came up during discussions of successes as well as discussions of the projects I classified as "not so great".

The lesson learnt from this is the importance of monitoring. The very good projects all had proper monitoring in place, while most of the not so great projects had very limited monitoring in place during test execution. This has changed the way I approach monitoring on projects and even the way I plan test scenarios and execution.

Planning for Monitoring

Although monitoring is always mentioned somewhere at the start of a project, most of the people involved don't realise the important role it plays in determining how successful or useful the performance testing will be. Performance testing is all about results and a successful test can be measured against the usefulness of the results. In my experience I've seen many tests done with long hours and lots of money invested in performance testing, but unfortunately many tests produce results that are not meaningful or useful.

So how do we ensure that the tests we run don't end up being meaningless because the results are meaningless? I believe it starts with the assessment and planning phases of a project. I include the importance of monitoring and the need for useful results in all discussions when a new project starts and tell everyone that the testing is planned around the results we want to get out of each test run. The important point here is to make sure all the necessary people are involved and make them understand that without monitoring there will be no useful results.

All of this seems very basic for the experienced performance tester, but planning your tests to give the right results with the right measurements can be quite a challenge. You have to know what will be monitored and which tool(s) will be used to achieve this. Including the right people to get the results is just as important.

Doing this early in the project gives you time to deal with issues like administrator access and/or people that don't want to or can't allow you to monitor certain resources on certain servers or in certain environments. I have one customer where I'm never allowed to monitor the network because it is seen as part of production. I often experience problems like this, where I can't monitor certain areas, mainly because the specific administrator doesn't allow any "outside" people access to resource monitors. Getting administrator access for Windows' Perfmon is a huge challenge and even impossible on some projects while it is a logical formality on others. I believe this is more of a challenge for consultants like me, than for performance testers with a permanent position within a company. The main goal is to emphasise the importance of monitoring right from the start and fight (in a very polite way) for rights to get access where needed. I find using examples from previous experiences work the best.

What to Monitor and Which Tools to Use

The most obvious rule is to make use of what is available. If you are using a commercial tool like LoadRunner you have an advantage because it includes so many monitors. Results can be correlated easily and overlaying graphs can be done quickly. Analysis is generally made a lot easier by the commercial tools. So if a commercial tool is available, use the monitoring it offers.

Knowledge of the monitoring requirements of the tool is essential. Do you need any administrator access or passwords? Do you need to install any agents on the server to be monitored? What measurements are available?

A very important note: I'm the performance testing expert on the project. I'm not the SQL, UNIX, Windows, Oracle, WebSphere, network, database or any other type of specialist. I don't know what all available measurements are for. Over time I've learnt a lot about these and when and how to use them, but my area of expertise is performance testing.

Talk to the relevant administrators and experts about monitoring their area. Ask them for input on metrics related to their area. Ask, learn and take note. Maybe you can give advice on the next project.

If you're not using a commercial tool or the license to monitor via the tool is not available, you move to the next best option. I believe this to be the monitoring tools available in the different system software. A lot of applications nowadays include good monitoring tools. This is also a good way to keep administrators and other relevant people involved with the testing. I let them gather the stats and pass the data on to me for analysis afterwards. For this I make use of spreadsheets. Most applications allow data to be exported to Excel. With Excel or any other spreadsheet graphs can be manipulated and very good results made available.

If results or data is only available in the format used as output by the application used, this have to do. Correlating and overlaying results might be more difficult and time consuming, but if this is done only for potential problem areas good results are still possible. I always add the lack of monitoring or monitoring tools as a risk to the assessment report and test plan. This prevents me form being blamed for bad results later on and can sometimes help to motivate the need to get monitoring in place before the next test run. I also put names of responsible people next to monitoring if I'm not doing it myself. This guarantees that I receive results from them after each test run.

Monitoring Makes Testing Useful

Case Study 1 – Minimum Monitoring Available

The first example I want to refer to was a web application in a Microsoft environment. I usually enjoy Microsoft environments because so many metrics are available for monitoring. The only challenge is administrator rights needed to monitor system and database resources. On this particular project I wasn't allowed any access and the various administrators and architects agreed to do the monitoring themselves. Although I greatly disagree with this, there are times where I have to continue testing like this.

Luckily this was a web application and I used LoadRunner for the testing. This meant that monitors for web resources like throughput, hits per second and connections were available. This is the case with most web performance testing tools and these resources should always be monitored. I always keep track of throughput and hits per second with response times. A lot can be seen about system behaviour by keeping an eye on these three metrics.

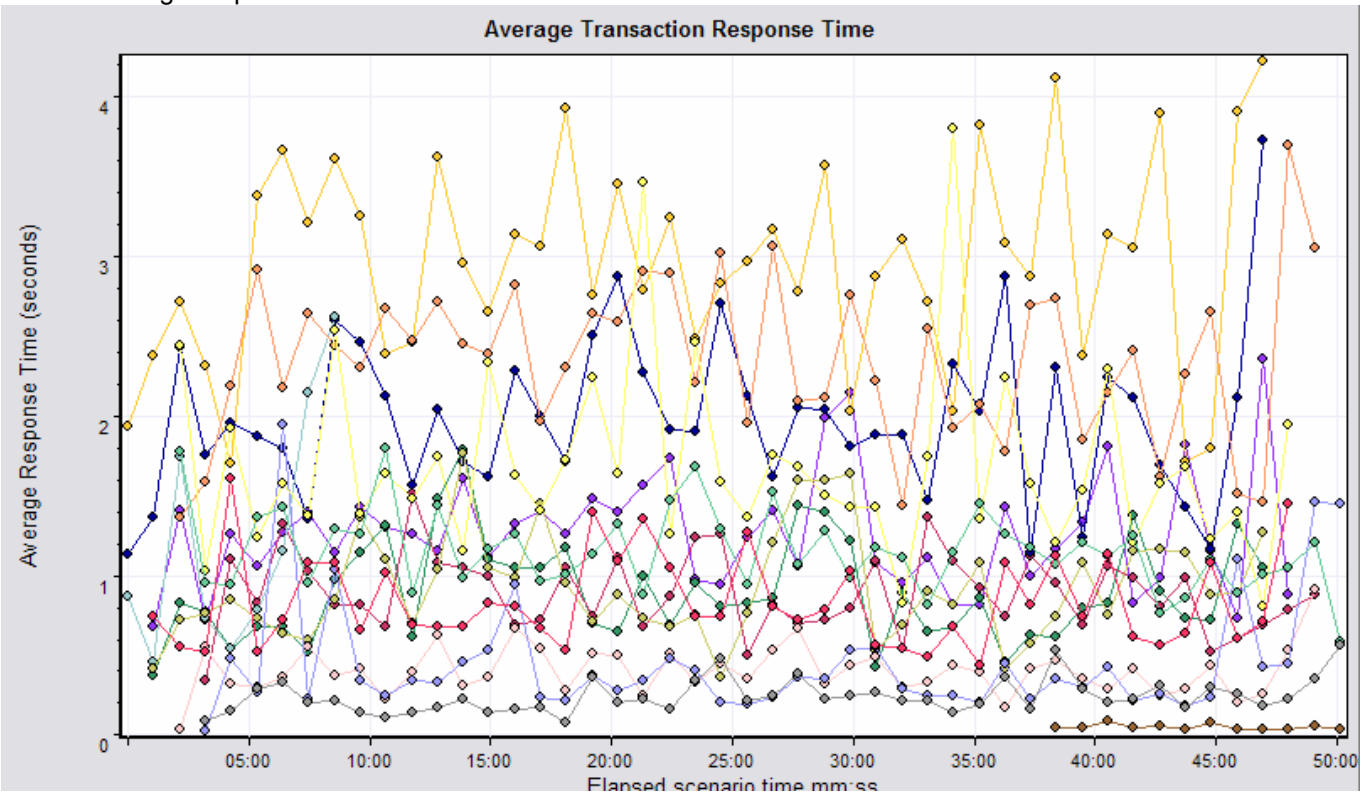
Load balancing was also very important on this project. Users were balanced between two servers through parameters linked to their user ids. I decided to monitor what I could for both servers separately to confirm load balancing and to add a bit of meat to my results. At least I could compare response times, throughput and hits per second for the two servers with no other monitoring available to me. I achieved this by running separate scripts for the users with the various user ids. The analysis tool from LoadRunner then allowed me to filter the results per host, giving me the separate results for each server.

Scripting and initial testing were run on the one server only and response times were good throughout scripting and baseline testing. However, average transaction response times were much slower during the test with load balancing to the two servers. The fact that I was able to show results for each server on its own proved crucial.

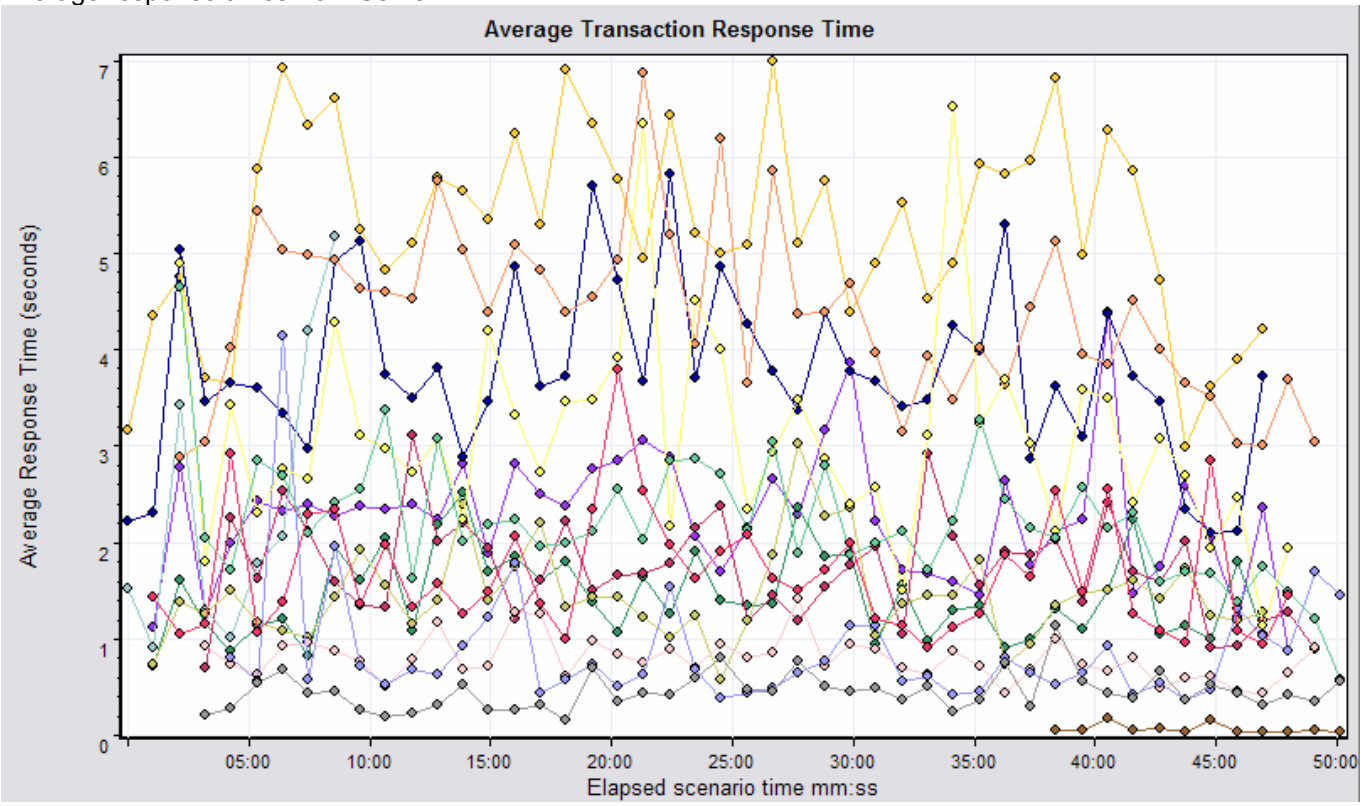
Lesson learnt: plan your tests so that you can produce separate results for each active server in the environment. Always include transaction response times in this planning. Confirm response times are the same from all application servers. Start taking note of response times when scripting starts.

The graphs below show the average response times for the test.

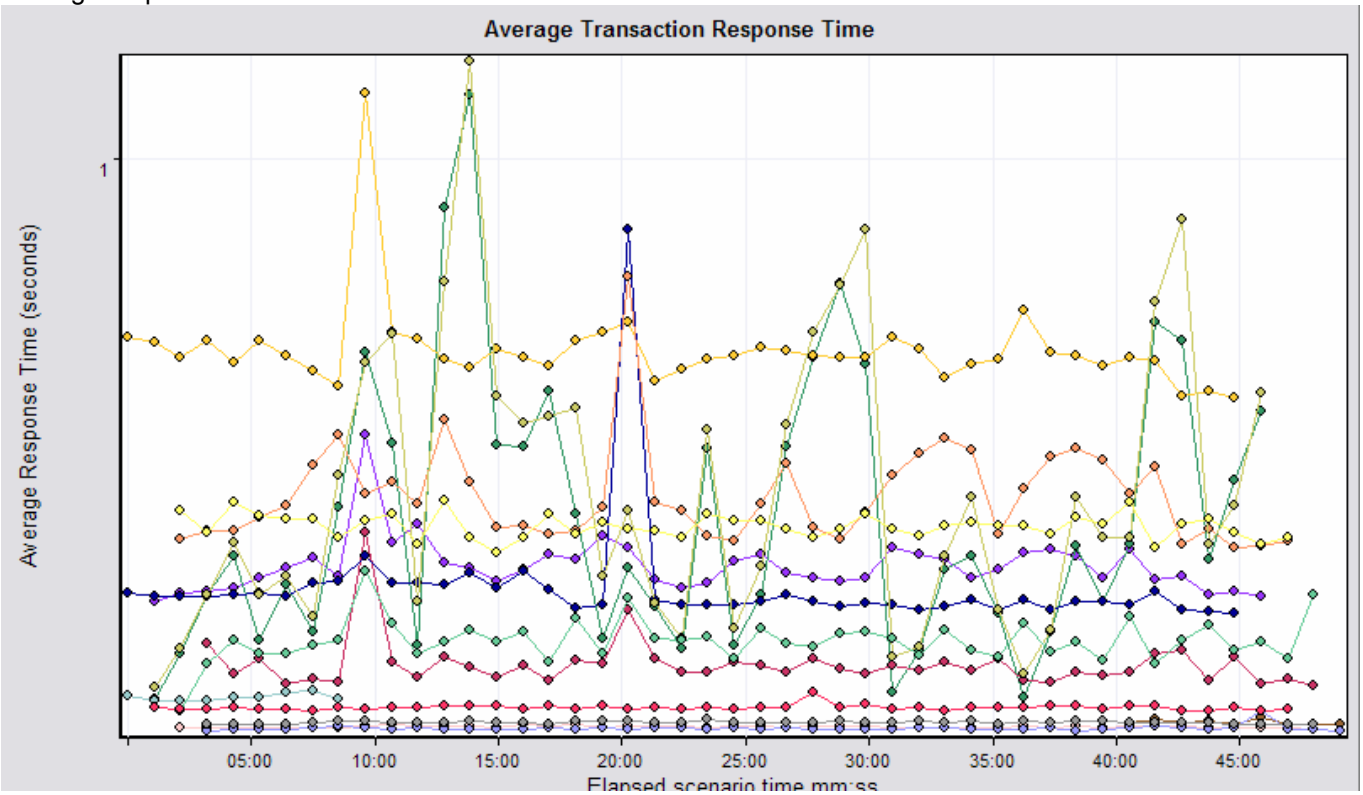
Overall average response times for the test:



Average response times from Server1



Average response times from Server2



Average transaction response times during the test were slower than the response times of previous tests that connected to Server2 only. The results show that the response times from Server1 are notably slower than response times from Server2. The first reaction was to blame load balancing with more users on the one server resulting in slower response times. However, the connections graph showed the same number of connections per server, meaning the same number of users per server with load balancing working fine. Throughput and hits per second were marginally less on the "slower" server, but with no other monitoring in place, it was very difficult to establish what the problem was.

In the end the problem was the NIC on the one server that was set to half duplex. A configuration problem, but I identified the fact that there was a problem with minimal monitoring available to me by making full use of what was available.

Lesson learnt: make maximum use of the metrics available during testing. Do so even more when limited monitoring is available.

Case Study 2 – Full Monitoring Available

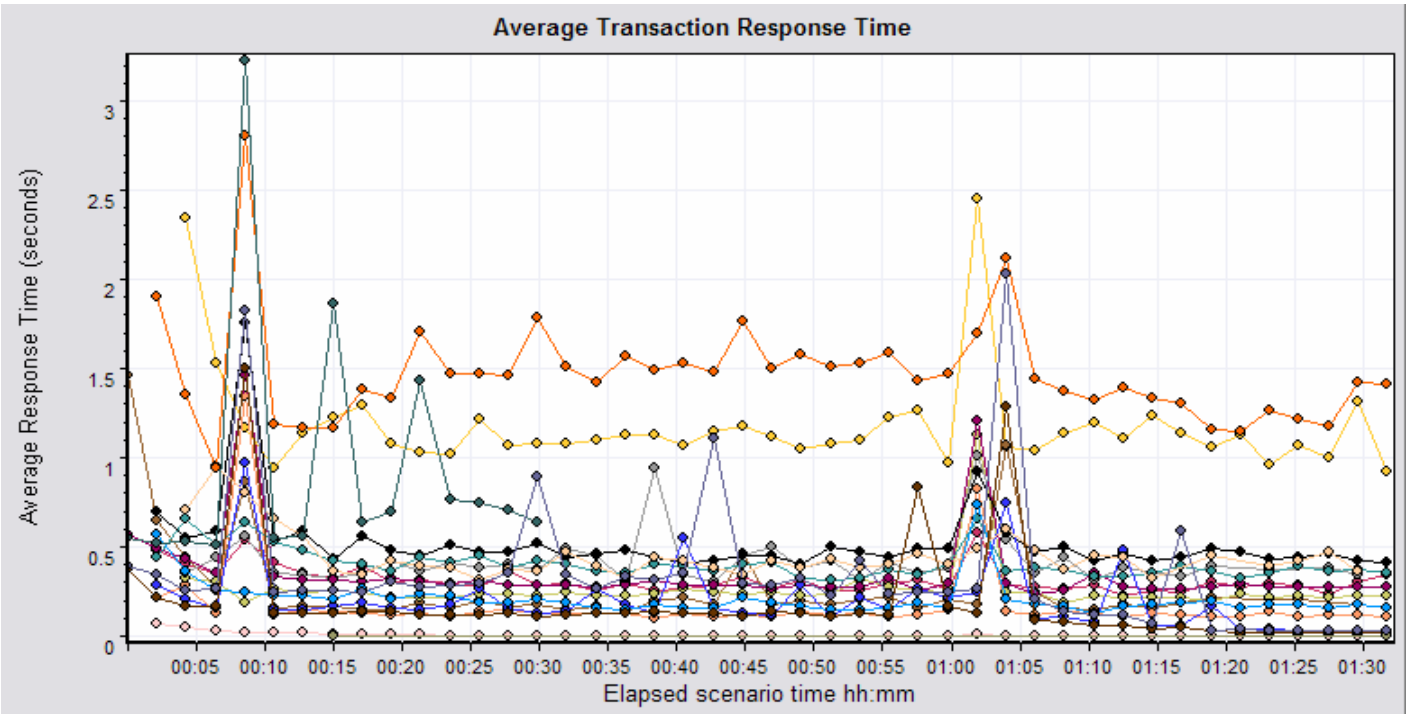
The second example shows the benefit of having proper monitoring in place during test execution. The test was run in production after a software upgrade. Although everything passed with results as expected, a memory utilization monitor showed a decline over time in available memory. In this system there is more than enough memory available and memory didn't run out during the load test, but would have if the test was run long enough.

What is very important with this example is that everything appeared fine during the test, with performance meeting all requirements. Everything was fine indeed, while there was still memory available. If the memory graph is taken out of the equation, the results look good with no errors or failures for the duration of the test. When the memory graph is added to the results, it shows there is a problem. After analysing the results, I made a recommendation that the test should be run again for a longer time. The customer followed my recommendation and we repeated the test two weeks later. This time I ran the test longer and only 11 minutes after the previous stop time the memory did indeed run out. This was great for me as a potential system crash in production was avoided.

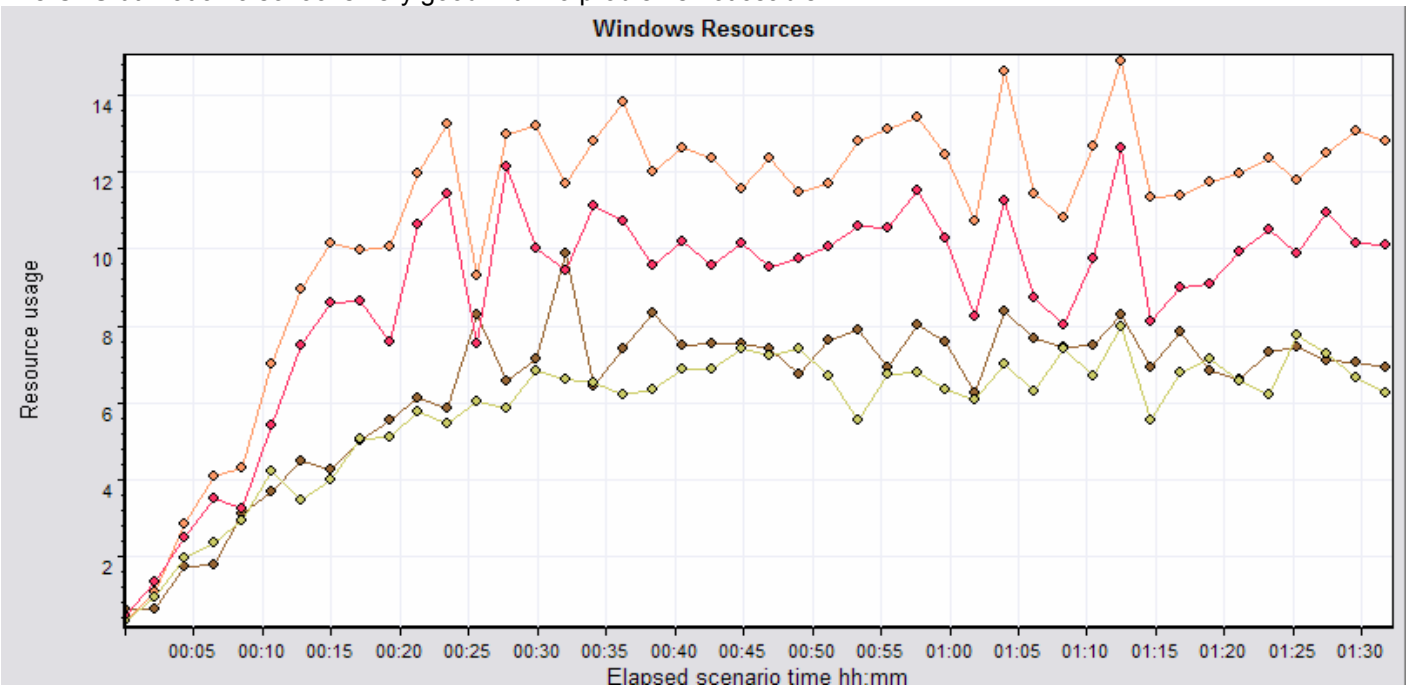
Lesson learnt: Review and analyse all available results carefully, even if everything looks fine. Make sure that everything is fine in all the available graphs and results. Don't assume nothing is wrong in a specific area because the graphs "look okay".

This example demonstrates how important monitoring is during test execution. If the memory wasn't monitored, the results would all have been fine and memory would have run out over time in production. Good response times alone are not enough evidence that the system will perform well. You need more results to be able to confirm a system will be fine in production.

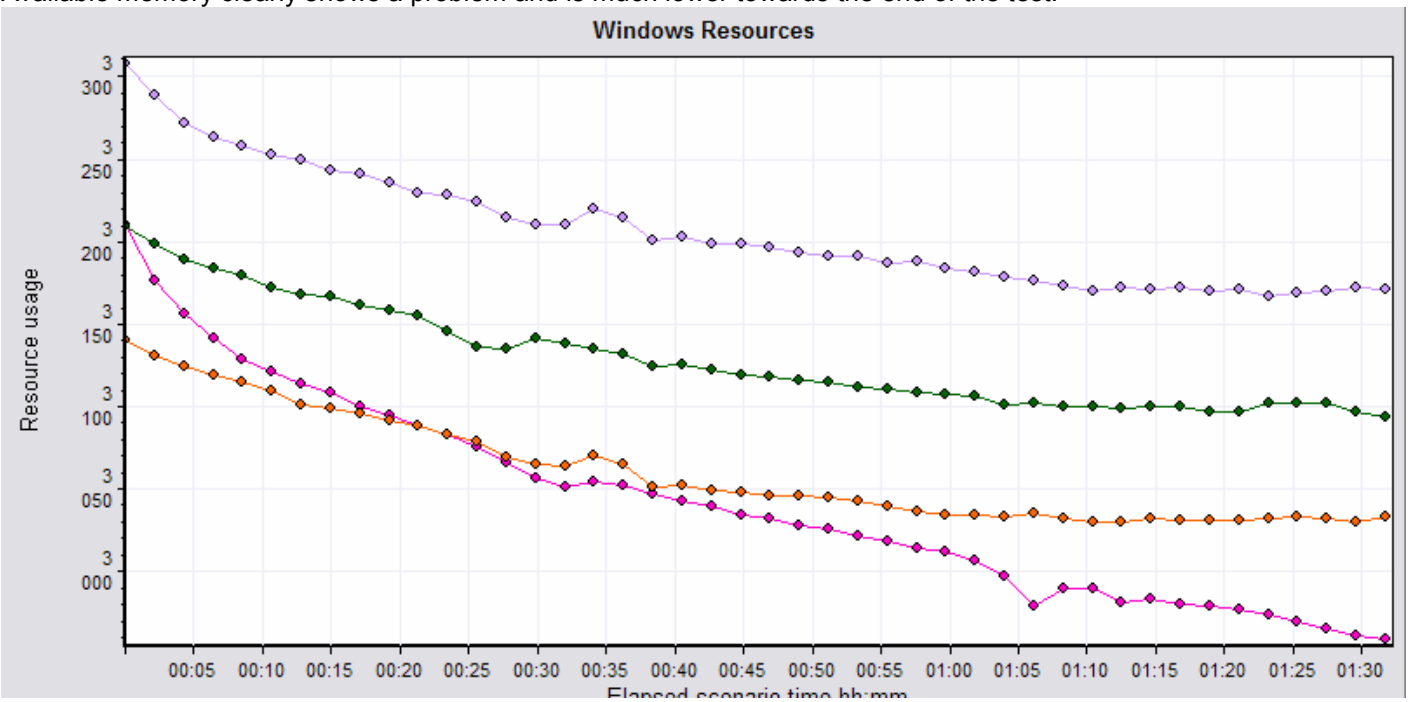
The following graph shows average response times for the test with no problems noticeable. The test was run for 90mins and response times look good with 300 users for the duration of the test. The requirement was all responses below 5 seconds with 300 users working on the system.



The CPU utilization also looks very good with no problems noticeable.



Available memory clearly shows a problem and is much lower towards the end of the test.

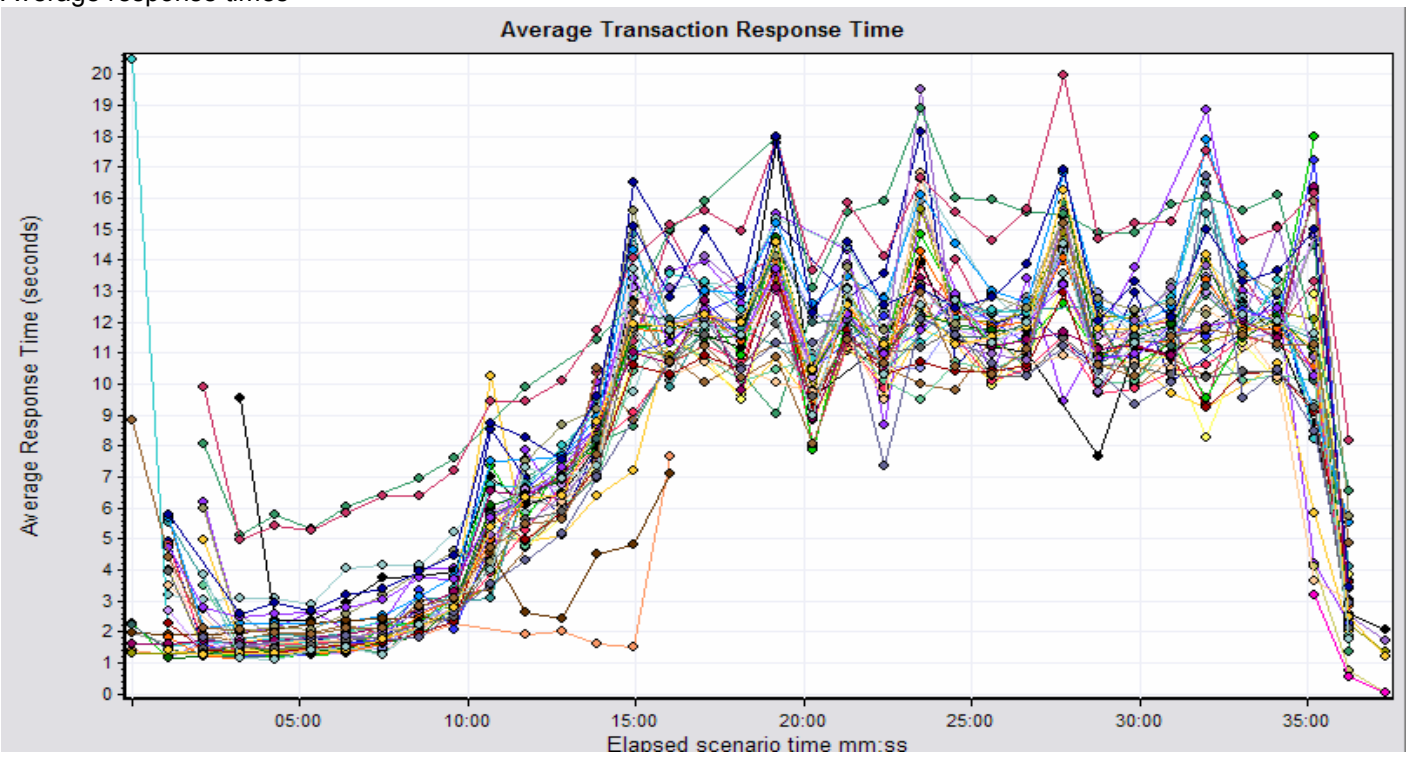


Case Study 3 – Full Monitoring Available

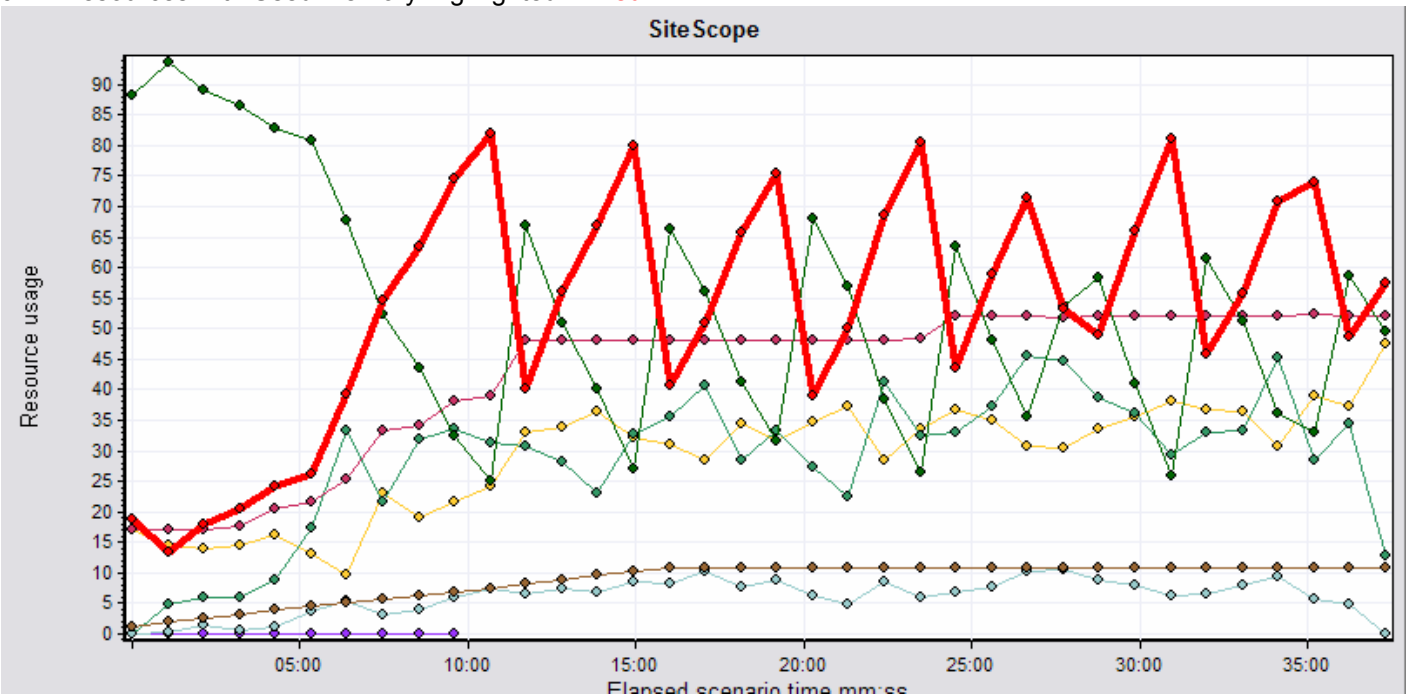
I use this example often to motivate the need for monitoring and the need for proper monitoring tools if adequate monitoring and/or tools are not available.

Two big problems were solved by analysing the results from the monitoring I had in place during the testing. The spiking in response times and the increase in response times about 10 minutes into the test. I found this example useful because results were used from two tools. It also demonstrates that spending time analysing all results after a test is a definite requirement for any performance tester.

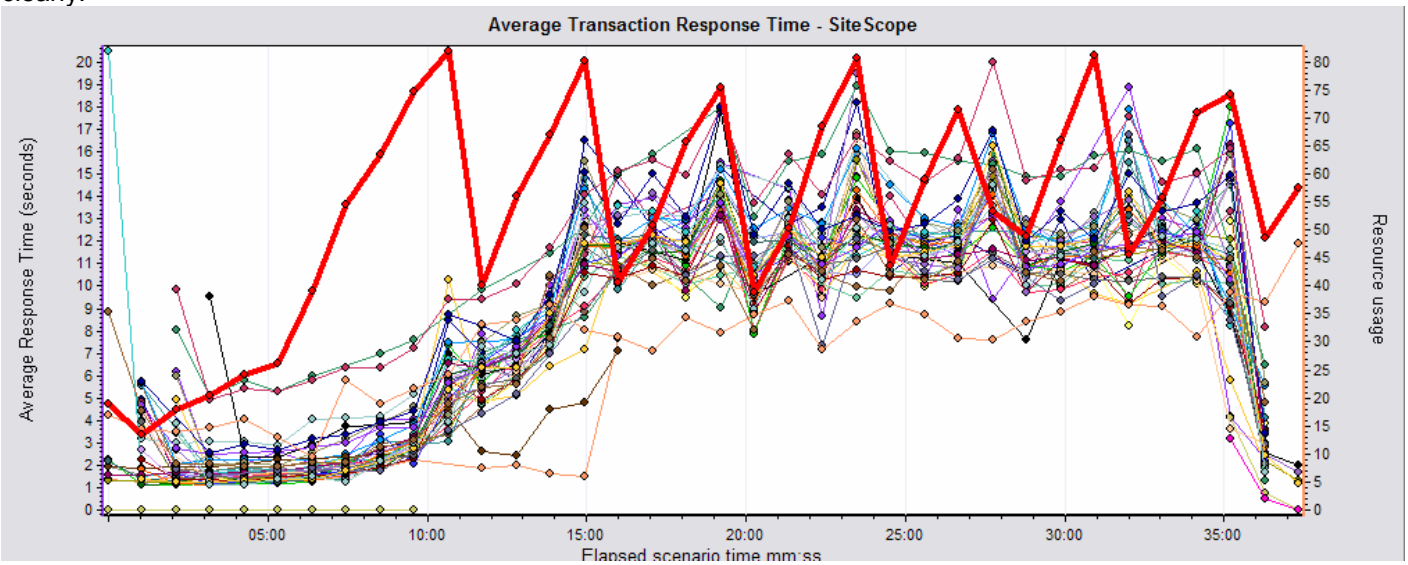
Average response times



JVM Resources with Used Memory Highlighted in Red



The correlation between the spikes in the two graphs is obvious and by overlaying the two graphs I could see this clearly.



This is a very good example to show the importance of resource monitoring and in this case the use of good tools to get useful results. Garbage collection for the JVM was investigated and changes were made to settings for garbage collection that eliminated the spikes in the response times.

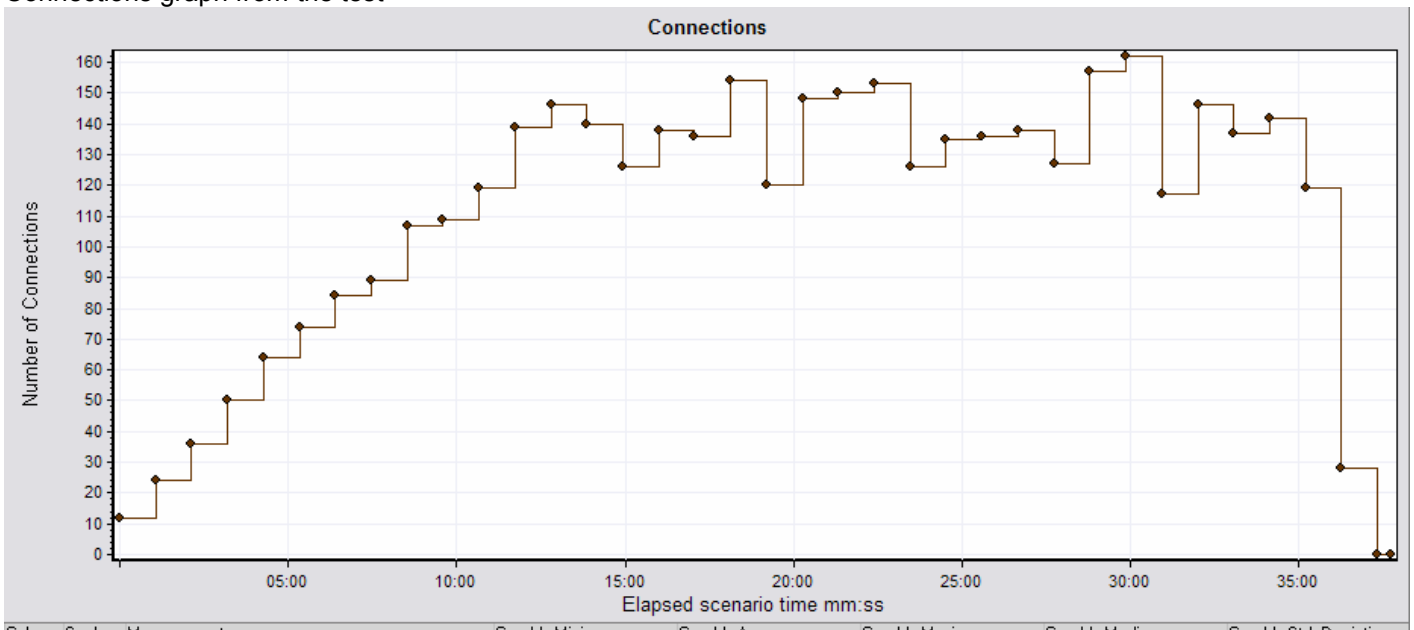
Lesson learnt: work through all results and look for graphs with similar trends or patterns. Compare spikes with spikes or look for resources that reach a maximum at the point where another graph shows a problem.

The second problem I looked at was the decline in performance from 10 minutes into the test onwards. This seemed to be directly linked to users ramping up, but I couldn't find any resource or other graph that showed a similar pattern. I worked through all the available results and also made sure I understood what each graph was showing me. As I was looking at the connections graph I read that the graph showed the number of open TCP/IP connections during the test. It also explained that two connections are opened per user for each Web server and if the number of connections

reaches a plateau, and the transaction response time increases sharply, adding connections would probably cause a dramatic improvement in performance.

Two things got my attention: I was running the test with 100 users and my connections graph showed a maximum of 162 connections and I certainly experienced a sharp increase in response times. On the graph it certainly looks as if the number of connections reached a plateau.

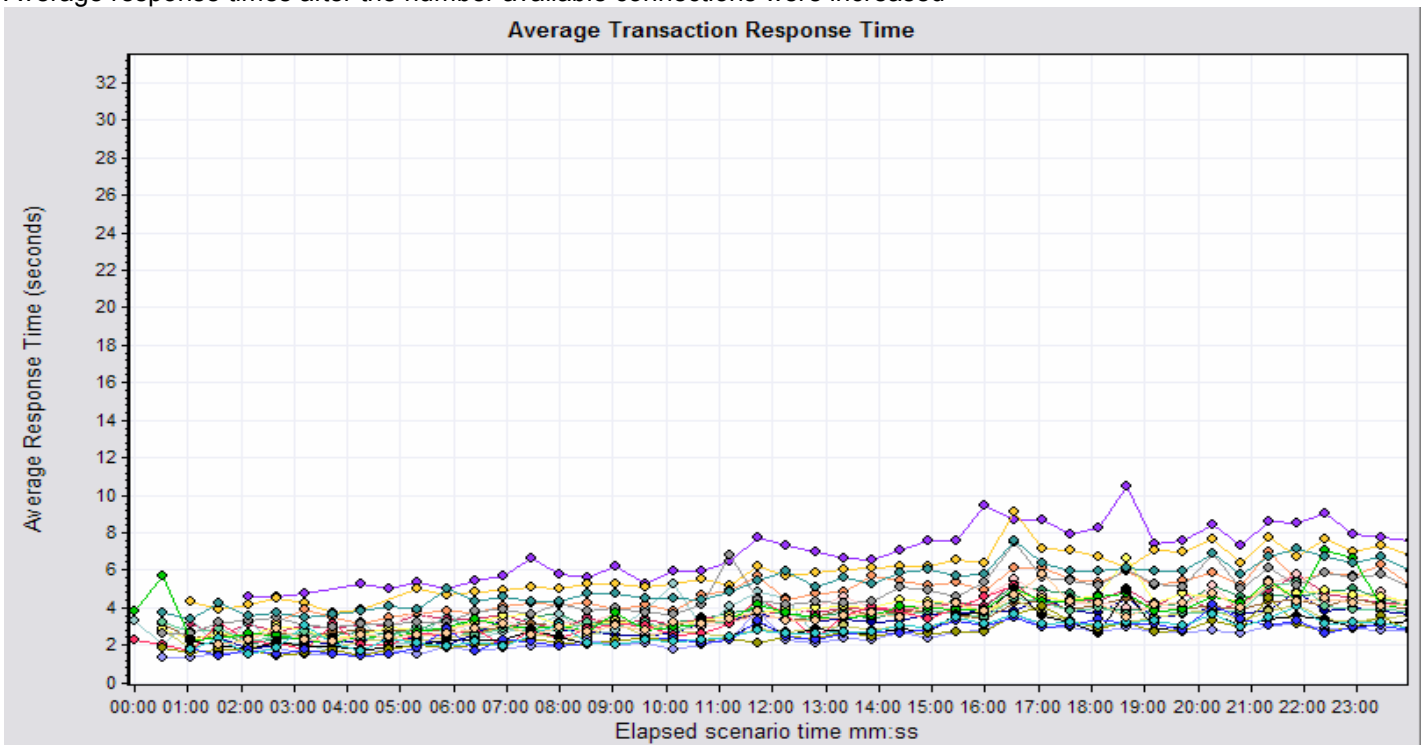
Connections graph from the test



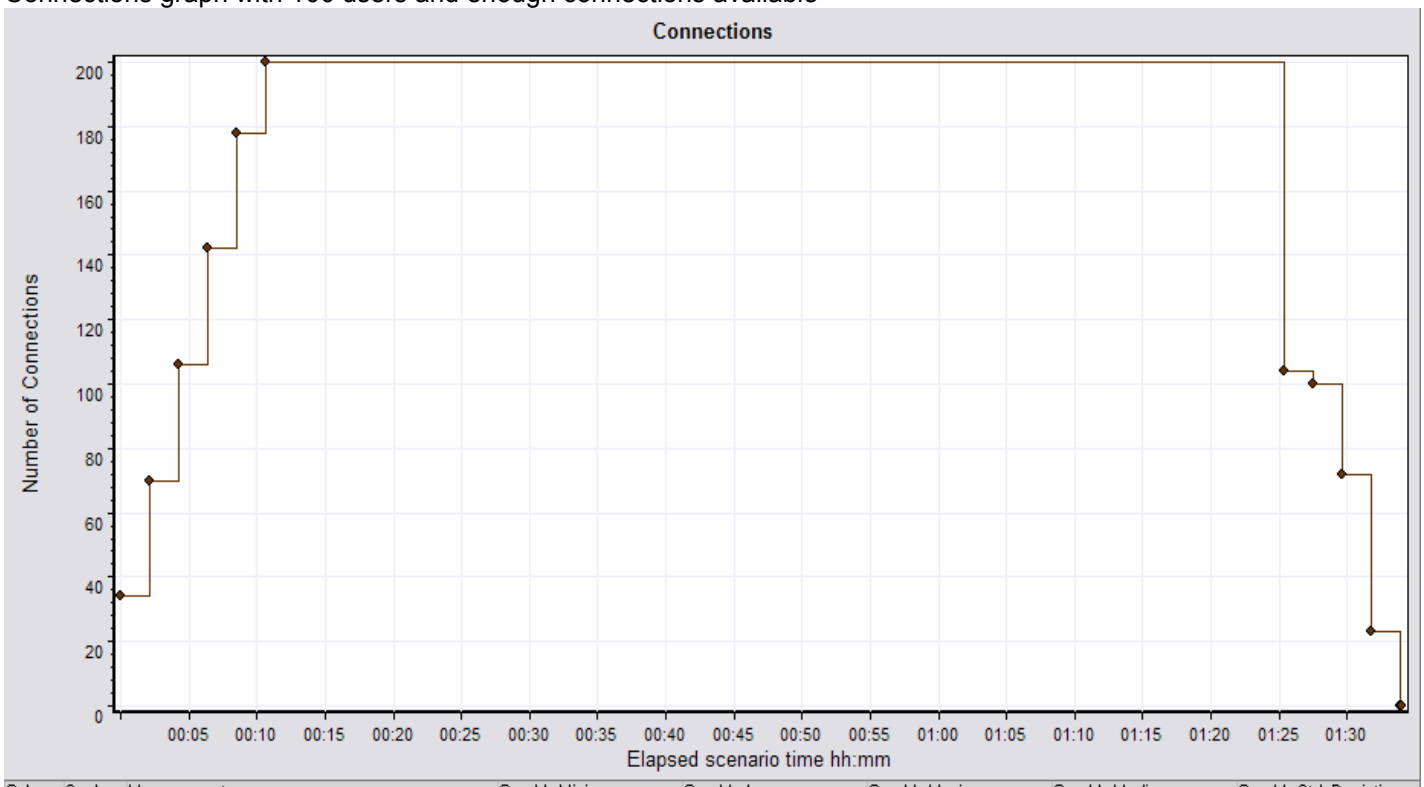
Color	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviation
	1	Connections	0	107.784	162	126	47.603

I mentioned this to the system architect and after investigating he confirmed that the maximum connections available were 162. This was increased to 210 and I ran the test again. The results were fantastic.

Average response times after the number available connections were increased



Connections graph with 100 users and enough connections available



Color	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviation
	1	Connections	0	85.75	200	100	62.87

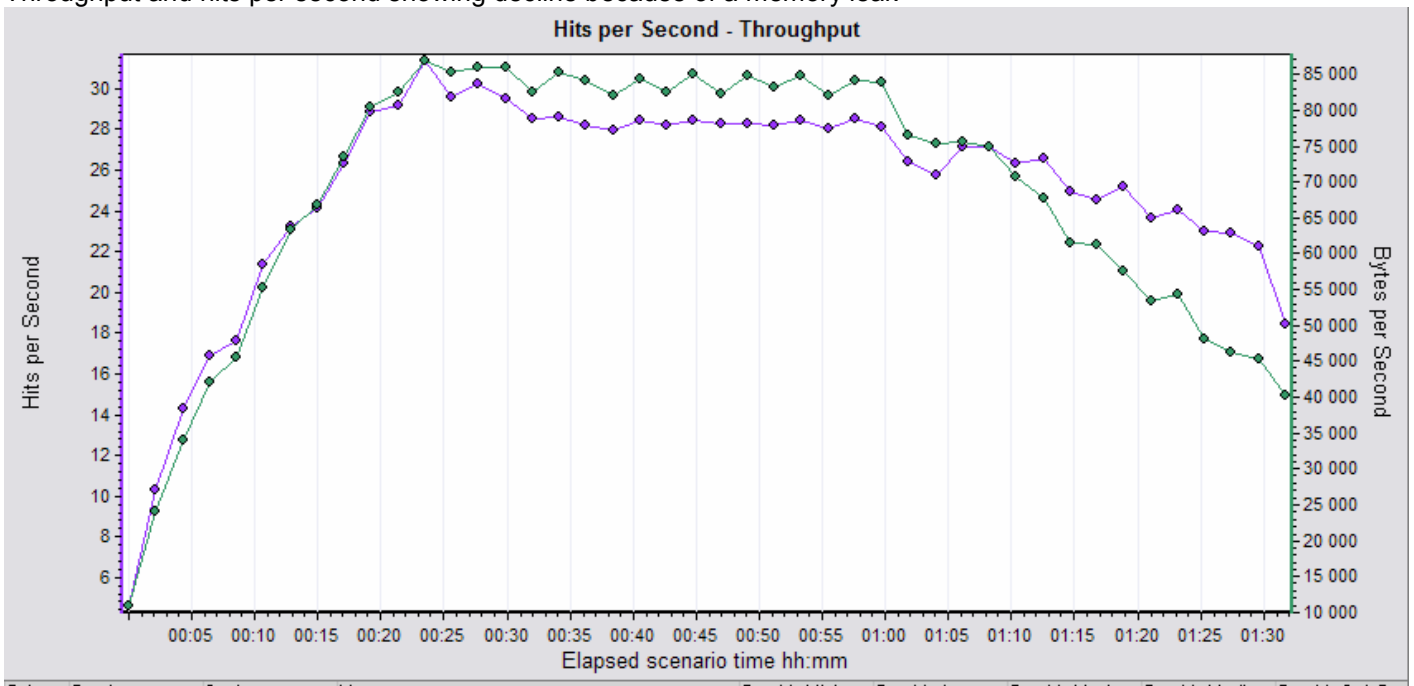
Lesson learnt: Understand what each graph or measurement is showing. Share what you find with the appropriate people.

Important Measurements for Identifying Problems

Looking at the different cases discussed in this paper, it is clear that response times are always very important. CPU Utilization and memory are two more metrics that should always be included. Throughput and hits per second are just as important. I've identified two memory leaks by looking at throughput and hits per second graphs during load tests. A steady decline over time is usually an indication that all is not well.

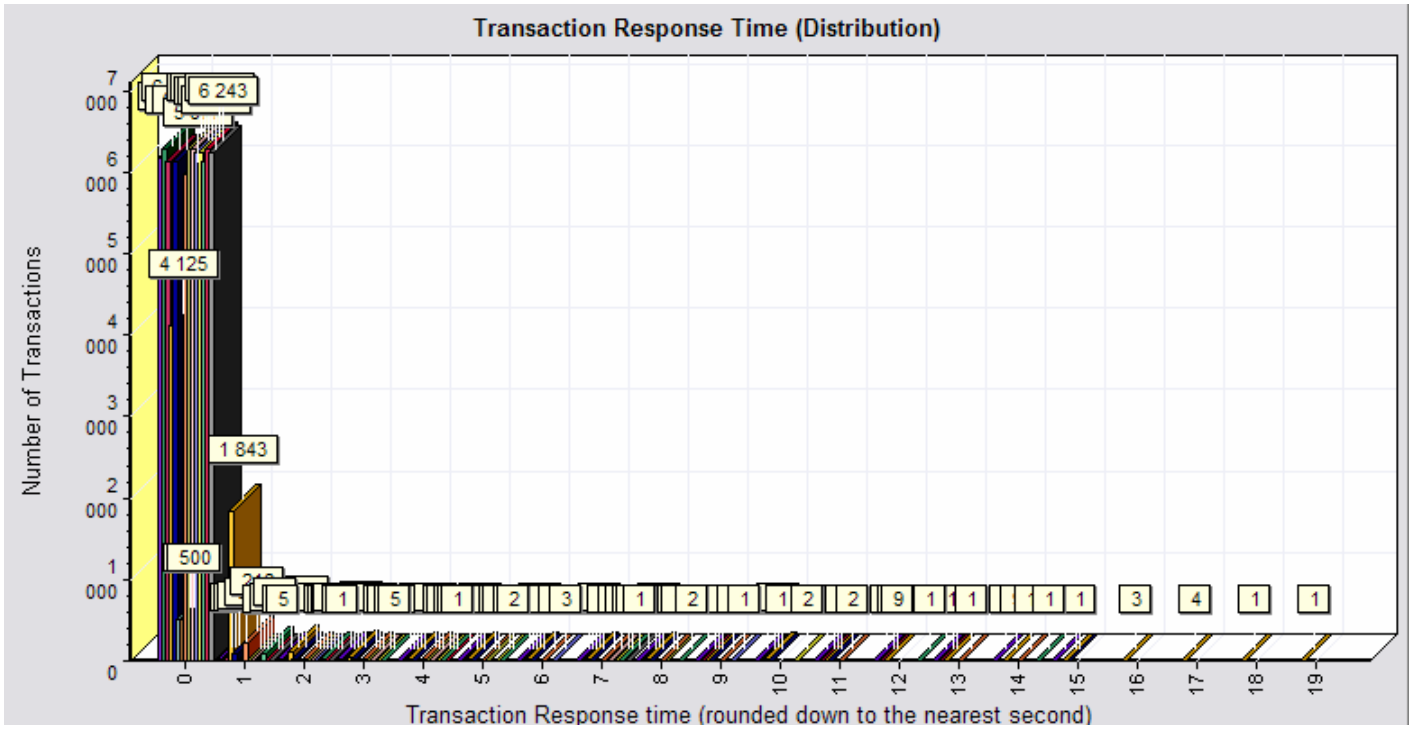
The graphs below show the steady decline in throughput and hits per second when a memory leak is present. I've experienced a memory leak without this decline, but in most case this will be present. In fact, a lot of problems can first be noticed by looking at the throughput and hits per second during a test. The decline is not limited to memory only and nearly any resource that reaches a limit will show, and can first be spotted, by the decline in throughput and hits per second.

Throughput and hits per second showing decline because of a memory leak



Color	Graph	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximun	Graph's Median	Graph's Std. Dev
	Hits per Second	1	Hits	4.617	25.046	31.367	27.102	5.348
	Throughput	1	Throughput	10885.563	68236.701	86900.023	75558.594	18980.902

Another very useful graph is the transaction response time distribution graph. The graph is not available in all tools, but I use it often to show where the bulk of the response times were. The graph is handy if maximum responses are slow and can be used to show that the bulk of transactions were good and only a few had slow response times. An example of the transaction response time distribution graph is shown below.



Summary

There are so many metrics available for monitoring that I don't try to know or master them all. I rely on the architect, administrator or expert on site to guide me when I have to decide which measurements to monitor during a performance test. I always ask for specific monitoring requirements and this usually gets the relevant person involved to help me with the decision. Everything can't be monitored, but I make sure that monitoring is in place to cover the whole infrastructure.

The table below contains the monitors I use most often. These are the monitors I use if no specific requirements for monitoring exist.

Web resource monitors Hits per second Throughput Connections HTTP responses per second
Transaction monitors Transaction response times Transaction response times under load Transaction response time distribution
Windows resource monitors CPU utilization Memory utilization
UNIX resource monitors Average load CPU utilization System mode CPU utilization User mode CPU utilization
Microsoft IIS monitors Bytes sent Bytes received Get requests Post requests Maximum connections Current connections
WebSphere monitors Used memory Free memory Connection pool size Connection wait time Connection percent maxed Connection percent used Rolled back Timeouts Current requests Session size
SQL monitors Cache hit ratio User connections
Oracle monitors Current open cursors Current logons

Contact Details

Johann du Plessis

Technical Test Consultant

Micro to Mainframe

johannd@mtom.co.za

+27 76 601 4840

www.mtom.co.za

www.performancetesting.co.za