Scapa® Test and Performance Platform
Remedy Comparison
Alderstone Consulting & Scapa Technologies
June 2015

Introduction

This performance engineering exercise was conducted in order to compare performance of Remedy versions running on standard architecture and seeked to:

- Benchmark performance on each version of Remedy based on a standard hardware and software setup.
- Identify performance bottlenecks (if any)

With the release of BMC Remedy 9.0, and the rewritten Java stack, a series of tests were carried out to make comparisons in performance between ITSM 9.0, 8.1 SP1 and 7.6.04. An identical approach was used against each system.

Approach

How Scapa works

A user transaction is captured at the HTTP layer by using a standard browser via an instrumented proxy. The proxy supports additional proxies, secure sockets and various forms of complex authentication, and records a trace of the HTTP interaction which, subject to additional processing, is replayed against the server. The HTTP replay facility again supports a broad range of encryption and authentication models and/or proxies and is extremely lightweight, so that very significant levels of usage can be simulated from a small client machine.

The Remedy web client speaks to the Remedy server by issuing commands embedded in the HTTP requests. These commands correspond quite closely to the underlying Remedy API calls, with API arguments serialized into and out of the HTTP stream.

Scapa exposes the serialized HTTP/Remedy API references in an XML-based capture/replay trace that forms a TestComponent. This representation of the TestComponent is edited through the Scapa GUI which has a number of automatic tools to support parameterization, equivalence processing and generalization (adding variance) at the HTTP layer (no programming is required).

At test time the XML representation is compiled into a highly performant script format.

Parameterization/equivalence processing is handled through available Scapa functionality which allows introducing parameters for all occurrences of certain literal values. The necessary script changes are handled automatically and the resulting parameter instance are editable through an easy-to-use GUI. To effectively generate a Remedy http test, Scapa traces input/output dependencies to ensure that the generalized parameters are valid with respect to the workflow in the system.

In addition to dedicated Remedy connectivity it has a number of key features:

- **Dynamic control of load:** The load on the servers is controlled and visualized through a single user interface via one or more sliders. The user interface is like a graphic equaliser, which can crank up the load and see where the system breaks, whilst watching performance data inside Scapa.

- **Control over business throughput as well as user count.** If it is important to know the number of transactions per second the system can support rather than the number of users, Scapa allows this to be controlled directly.
- **Predefined tests and scheduled tests execution.** If dynamic control is not required, tests can be pre-defined to run in certain ways, for example for regression testing. They can also be scheduled for out of hour's execution.
- **Distributed test execution.** Tests can be run from multiple client machines, to measure end-user performance in the various geographies over which the applications are being deployed. Test result analysis. All the data collected during a test is available for analysis with Scapa's statistical console.
- **Reporting:** Scapa provides a full-feature reporting capability with PDF, CSV and Web output, and a broad range of summarization and aggregation facilities, including percentile reporting and other statistical functions. Every test run is written to a SQLite databases.
- **Monitoring:** Scapa test execution and system data collection can be scheduled on a continuous basis to provide an ongoing monitoring capability for Remedy applications.

Infrastructure

Software

- Default builds of BMC Remedy ITSM 7.6.04 , 8.1 SP2 and 9.0.
- Using MS SQL Server 2008 deployed on same server
- Scapa Test and Performance Platform 3.3 Enterprise Edition – 250 VU license
- Browser used was Google Chrome

Hardware

Identical Amazon EC2 m3.xlarge instances were used for each server build. Default configuration for m3.xlarge is 4 CPUs (Intel Xeon E5-2670 v2), 15 GB Memory and 100 GB SSD. A fourth m3.large instance was used for the Scapa Test & Performance Platform installation

TestComponent

For each of the three environments two TestComponent were created, a read-only (Console) process, and a TestComponent that created an incident, updated and closed it. A detailed list of the steps taken in each process follows:

Console task

1. Login as standard user
2. Open Incident Console
3. Refresh Console view using Refresh icon
4. Change from the default view in the console to "View All" Refresh Console using the refresh icon
5. Logout

Create Task

1. Login as standard user
2. Open Create Incident form
3. Populate the form, using lookups for Customer Name, Company name, etc.
4. Add a unique text summary

5. Assign the incident
6. Change Status to “In Progress” and Save
7. Using the Search form, search on the unique test set in Step 4
8. Open the searched for incident
9. Update and close the incident
10. Logout

Both TestComponents were set with a minimum wait time (user think time) of 60 seconds.

TestCases

For each environment, a TestCase containing the two TestComponent relevant to that environment was built with the following population split:

- Console: 180 users
- Create: 80 users

A Scapa Control Sequence (test macro) was built for each environment, ensuring that all tests runs would run in an identical load pattern. The steps were as follows:

Step	Duration	Console Users	Create Users	Total Users
1	5 mins	1	1	2
2	5 mins	3	1	4
3	10 mins	30	10	40
4	10 mins	60	20	80
5	10 mins	120	40	160
6	20 mins	180	80	260

Results

Initial observations looking purely at response times showed that all three environments performed in a similar fashion. Response times were very close to the minimum 60 seconds for all load levels up to an including step 5 (max of 160 users). As the users for Step 6 were ramped in, some deviation on response times were noted.

8.1 was noted to have the best performance during the final segment of load, with minimal change on prior segments.

When the focus switched to completed transaction rates, all systems performed to minimal differences in all periods prior to peak load, with the Console tasks reaching a rate of 2 completed tasks per second and the Create tasks completing at .75/second. At peak load, the rates reached 3 and 1, but with large variance in the rates of 7.6.04 and 9.0.

	Steady State				Peak		
	per sec	per min	per hour		per sec	per min	per hour
81_Console	2.279	136.74	8204.4		2.963	177.78	10666.8
81_Create	0.77	46.2	2772		1.339	80.34	4820.4
76_Console	2.139	128.34	7700.4		2.447	146.82	8809.2
76_Create	0.757	45.42	2725.2		1.038	62.28	3736.8
90_Console	2.106	126.36	7581.6		2.415	144.9	8640
90_Create	0.749	44.94	2696.4		1.031	61.86	3711.6

7.6.04 response times:

In the following screen shot, the two navy lines show the two user populations increasing and running at steady states. The two black lines are corresponding response times. Note that they are both very consistent until the last increases in user count

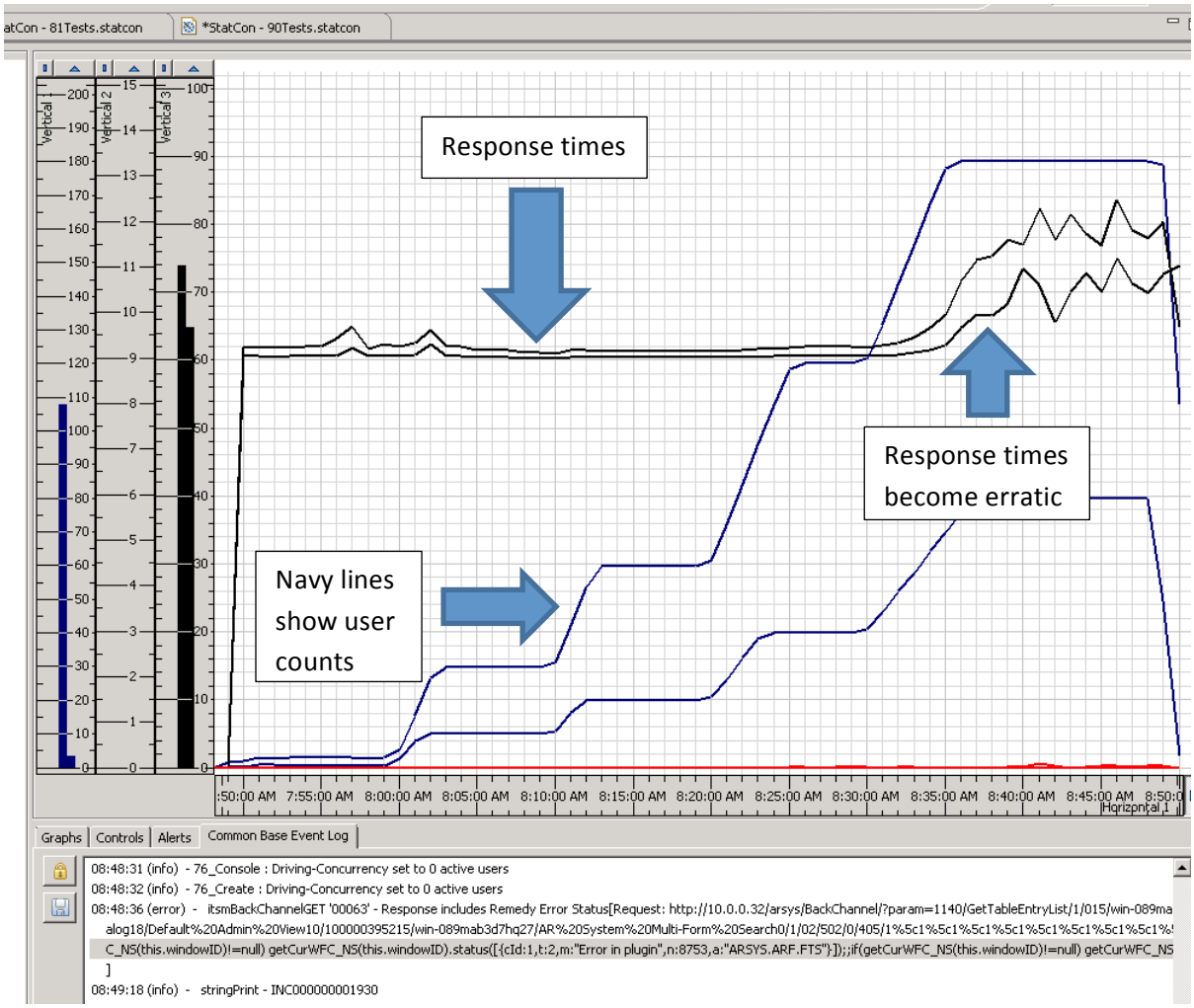


Table of the response times plotted against usercounts

Console UserCount	users	3	30	60	120	180
Console ResponseTime	secs	60.687	60.637	60.955	61.235	68.542
Create UserCount	users	1	10	20	40	80
Create ResponseTime	secs	61.567	61.818	62.025	62.693	75.179

The increase in response times in the final section of the test can be clearly seen. Some timeout and plugin error messages were noted at this stage (default timeout for any response across all tests was 30 seconds).

8.1 response times:

Again, the two navy lines show the two user populations increasing and running at steady states. The two black lines are corresponding response times. Note that this system does not experience the degradation of response times noted in 7.6, with just a very slight increase in response time for the final segment.

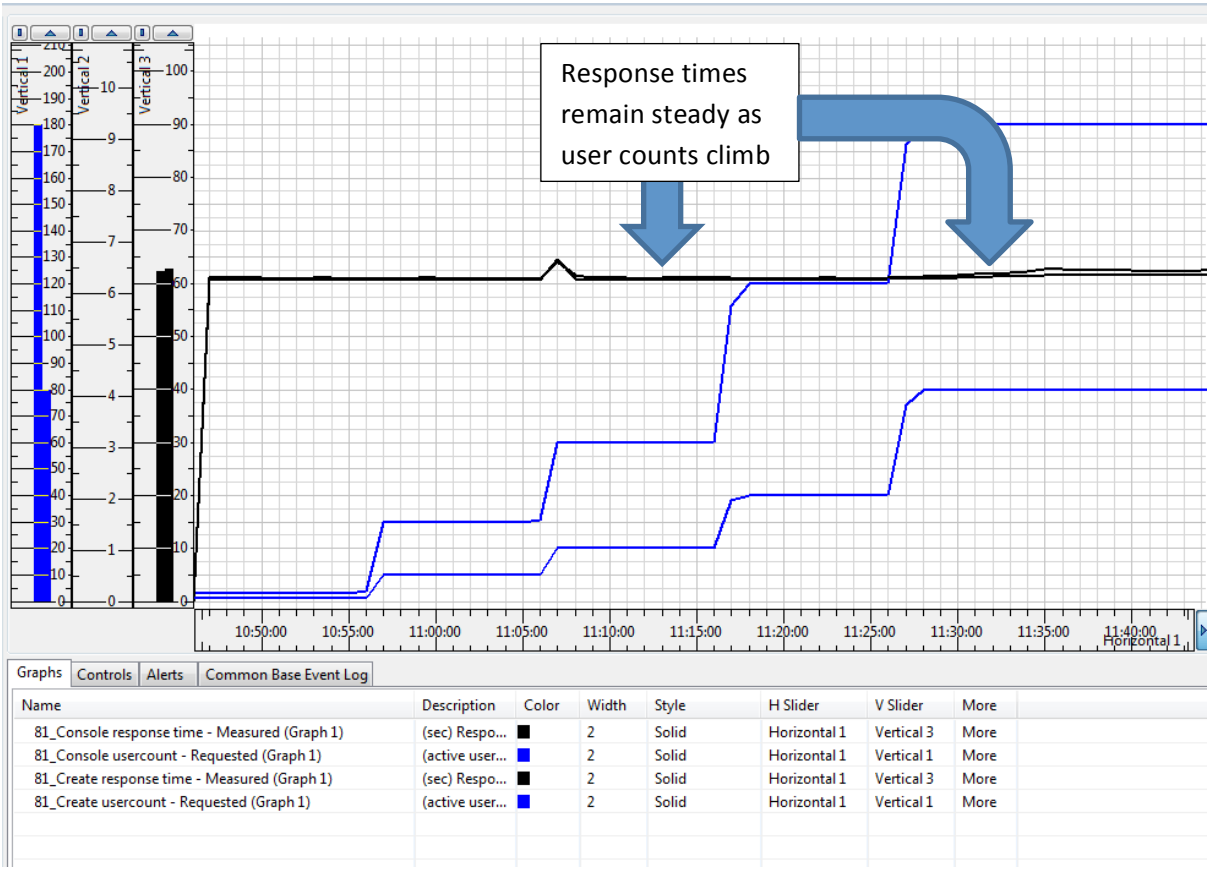


Table of the response times plotted against usercounts

Console UserCount	users	3	30	60	120	180
Console ResponseTime	secs	60.559	60.558	60.58	60.688	61.541
Create UserCount	users	1	10	20	40	80
Create ResponseTime	secs	60.948	60.935	60.965	60.983	62.424

9.0 response times:

In the following screen shot, the two navy lines show the two user populations increasing and running at steady states. The two black lines are corresponding response times. Note that they are both very consistent until the last increases in user count

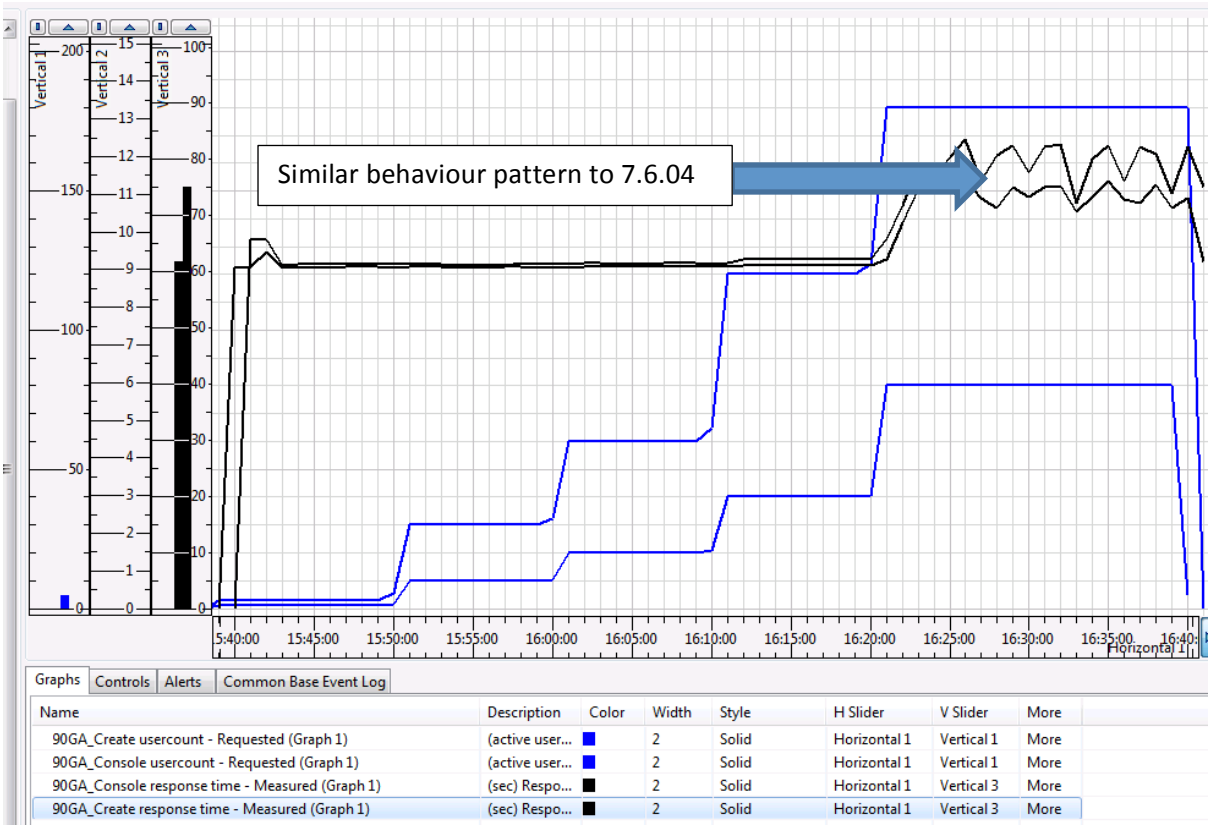


Table of the response times plotted against usercounts

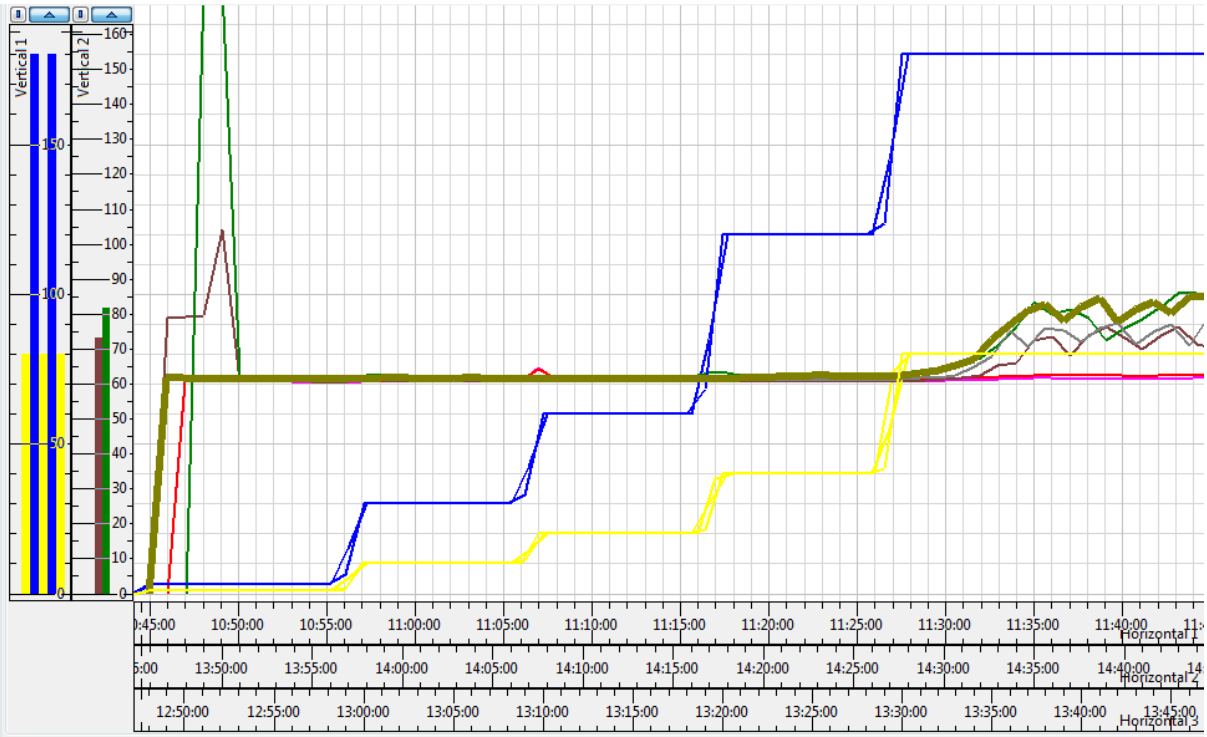
Console UserCount	users	3	30	60	120	180
Console ResponseTime	secs	60.8	60.77	60.875	61.148	73.397
Create UserCount	users	1	10	20	40	80
Create ResponseTime	secs	61.357	61.331	61.463	62.219	79.823

Response times in the peak load segment were noted to be poorer and show a higher level of variance than those experienced against 8.1.

Consolidated graphs

In this final screenshot, the graphs have been combined. The blue and yellow lines show the steady user counts climbing for the respective Console and Create tasks. The timelines have been manipulated so that the execution periods of each test have been transposed on the other. The other coloured coded lines are response times for the various tasks (key below).

This shows that each system performed almost identically well up to the final phase of user load – where we move from 160 total users to 260 total users, the exception being the 8.1 system which continued to perform as before.



Graphs | Controls | Alerts | Common Base Event Log

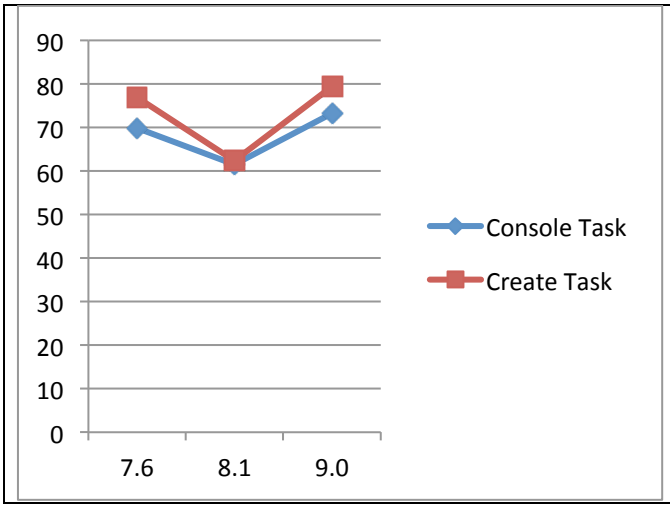
Name	Description	Color	Width	Style	H Slider	V Slider	More
81_Console response time - Measured (Graph 1)	(sec) Response...		2	Solid	Horizontal 1	Vertical 2	More
81_Create response time - Measured (Graph 1)	(sec) Response...		2	Solid	Horizontal 1	Vertical 2	More
81_Create usercount - Requested (Graph 1)	(active users) ...		2	Solid	Horizontal 1	Vertical 1	More
76_Console response time - Measured (Graph 1)	(sec) Response...		2	Solid	Horizontal 2	Vertical 2	More
76_Console usercount - Requested (Graph 1)	(active users) ...		2	Solid	Horizontal 2	Vertical 1	More
76_Create response time - Measured (Graph 1)	(sec) Response...		2	Solid	Horizontal 2	Vertical 2	More
76_Create usercount - Requested (Graph 1)	(active users) ...		2	Solid	Horizontal 2	Vertical 1	More
90_Console response time - Measured (Graph 1)	(sec) Response...		2	Solid	Horizontal 3	Vertical 2	More
90_Console usercount - Requested (Graph 1)	(active users) ...		2	Solid	Horizontal 3	Vertical 1	More
90_Create response time - Measured (Graph 1)	(sec) Response...		2	Solid	Horizontal 3	Vertical 2	More
90_Create usercount - Requested (Graph 1)	(active users) ...		2	Solid	Horizontal 3	Vertical 1	More

Each system scaled as expected to the period prior to peak load. This shows that users can confidently expect that a lightly loaded server would perform as expected while transactions and user count grow to a known maximum.

An analysis of the final segment (peak load) for each run shows the following:

- 8.1 produced the best (lowest) response times
- 7.6.04 timings were slightly higher
- 9.0 timings were highest (worse) of all
- The variance was higher in the Create task, which has a higher number of transactions in it as a whole

It should be noted that the tests were all run on a m3.xlarge Amazon EC2 instances. Using instances with more compute power may well move the envelope of peak load further up the usercount and transaction rate graphs, allowing different results to be noted.



Task	Time
7.6 Console	69.86
8.1 Console	61.50
9.0 Console	73.4
7.6 Create	76.87
8.1 Create	62.37
9.0 Create	79.82

Subsequence Timers

As noted earlier, each component had timers wrapped round pieces of the overall transaction – these are compared here.

Two sampling period are shown – a lightly loaded part of the scenario (30 Console users and 10 Create Users), and Peak Load (180 Console Users, 80 Create Users).

The values shown are the end user response times for various types of activity. These were defined during the TestComponent recording process. User wait, or think time, has been eliminated from the timings shown below.

		Lightly Loaded			At Peak Load		
		7.6	8.1	9.0	7.6	8.1	9.0
Console	ChangeView	0.155	0.09	.095	1.13	0.256	0.3903
	Console	0.101	0.05	.079	1.565	0.207	0.35
	Login	0.151	0.141	0.41	5.703	0.501	11.154
	Refresh1	0.058	0.079	0.08	0.458	0.218	0.34
	Refresh2	0.111	0.11	0.109	1.018	0.272	0.4

Create	AssignToMe	0.078	0.077	0.09	0.49	0.147	0.332
	NewInc	0.126	0.083	0.07	1.504	0.187	0.3474
	SaveInc	0.339	0.152	0.1476	2.997	0.496	0.9082
	SearchCustomer	0.094	0.06	0.089	1.227	0.139	0.27
	SearchInc	0.392	0.106	0.25	3.523	0.315	2.17

A brief outline of each timer:

TestComponent	Timer Name	Description
Console	ChangeView	The time it takes to change the console view, changed by selecting a different value from the default
	Console	Initial time to open the console
	Login	System time to complete a user login
	Refresh 1	Timing the response of a simple view refresh
	Refresh 2	As above
Create	AssignToMe	Time to Assign created incident to the test user, clicking the Assign to Me link
	NewInc	Time to open the Create New Incideent form
	SaveInc	Time for initial save of the created incident
	SearchCustomer	Time to open the Search Customer view and a list of default results appear
	SearchInc	Time to search for the created Incident by its own Incident number

Conclusions

Side by side testing of this reference Remedy ITSM systems allows us to make the following statements:

- All systems supported 160 simultaneous users without system component failure or functional issue
- 8.1 System supported 260 simultaneous users without drastic performance degradation for most of the user actions.
- The performance of the 7.6.04 and 9.0 systems degraded in the peak load period, with Login times for both systems increasing in the greatest proportion.

- Transaction throughput metrics suggested an optimum performance point at just under 5000 new incidents per hour (and additional 10000 console monitor transactions as a background load)
- At around a half of that load (3500 incidents per hour) user experience was very similar to what can be expected from the system when being accessed by a single user for all of the platforms.
- Most noticeable degradation of response times is associated with Logins and initial form openings and searches. As the tests necessarily log in and out for each iteration, this impact may well be lost on a real user.
- These results are only applicable to m3.xlarge Amazon EC2 instances. No inferral regarding usercounts or transactions rates on instances with different copmute power should be made without testing.

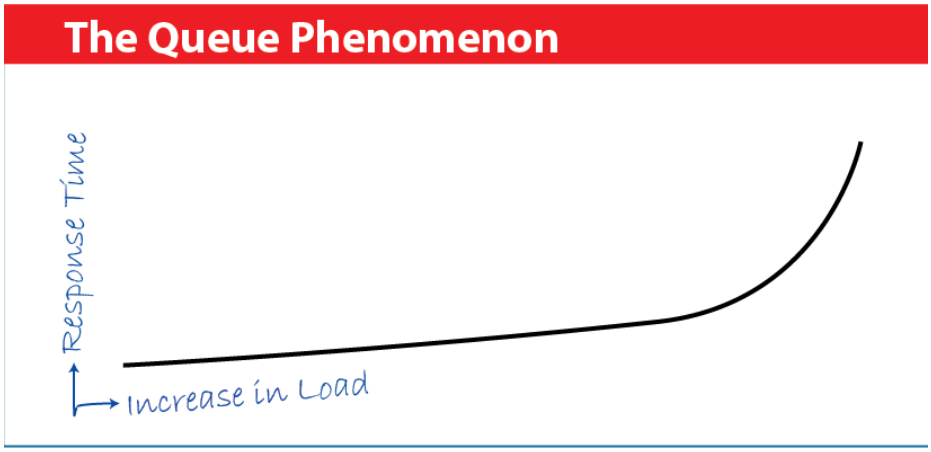
APPENDIX A : The Edge of capacity

The edge of capacity is sometimes referred to as the 'sweet spot' or pinch point and is the point where a system is at its maximum throughput, whilst maintaining an acceptable response time.

As load increases, a higher transactional load will be placed on a system (higher TPS). Eventually the system in question will not be able to process the workload efficiently, and some sort of queuing will occur. As the queue increases, the system will take longer to process each request and there will be an increase in response time. The edge is the point where the transaction request slows and the transaction time increases as the requested load is increased.

In other words the edge of capacity is the point at which the system is a maximal throughput capacity.

The Queuing Phenomenon



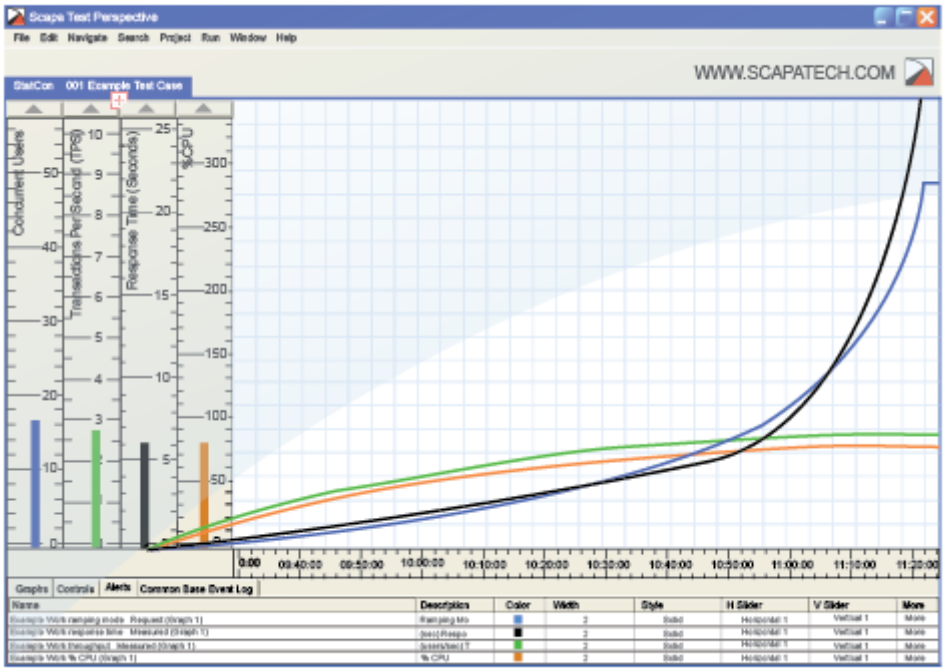
Queuing phenomenon encompasses all activities of our lives. We have to wait in line whenever the number of servers or the service rate of the server does not match the rate at which the customers arrive in the queue.

For Example:

- Waiting in line for an elevator
- Waiting in line to use an ATM
- Waiting in line at a checkout

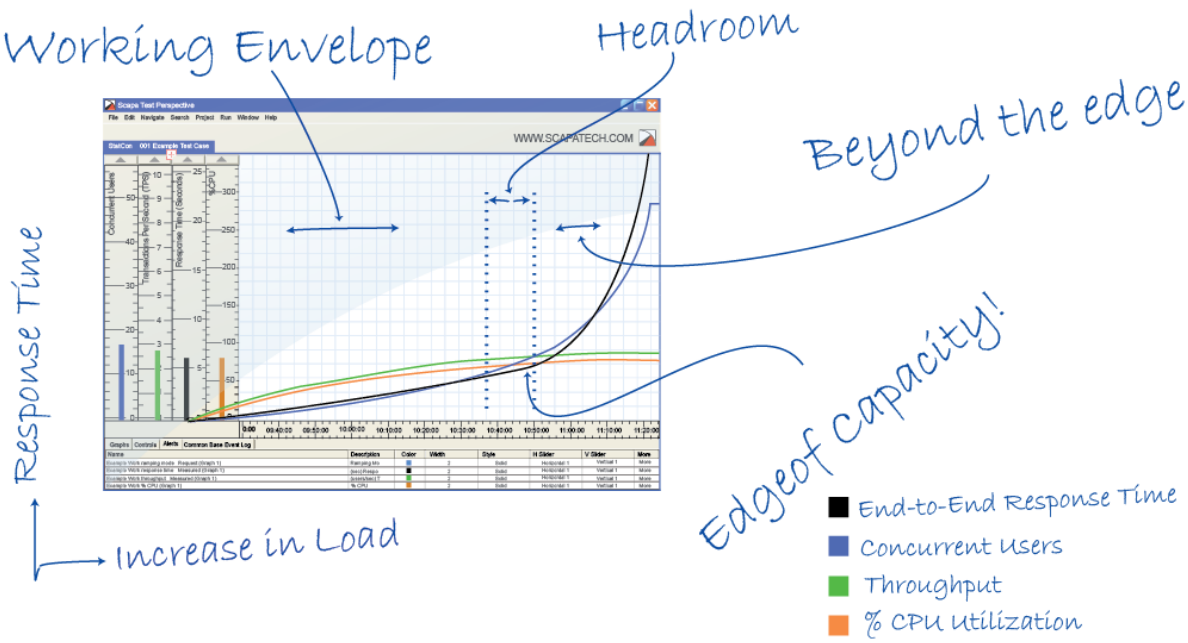
As the queue increases, the system will take longer to process each request and there will be an increase in response time.

A computer system is comprised of many queues. Some are complex, first in first out, first in last out etc., but the phenomenon remains.

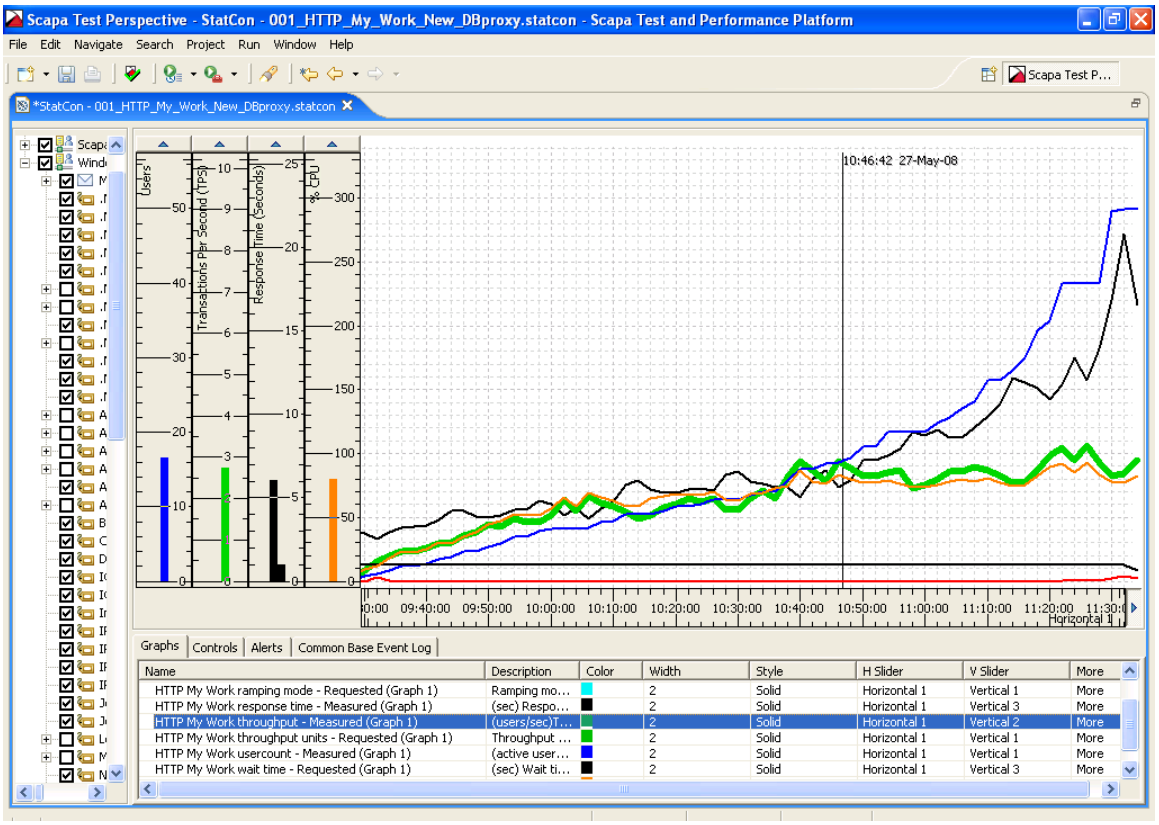


Scapa TTP has been designed so that load can be increased over time, during a single test cycle, so that the edge of capacity can be easily identified, within the one test cycle. Load can be increase and decreased during the test run in order to dial in on the edge. The edge can be clearly identified in the above screenshot. The compress timeline feature of Scapa makes it easy to see the results for the whole test run.

Terminology



The *edge-of-capacity* can be easily identified in the following screenshot. The horizontal timeline has been compressed so that the whole test run can be viewed.



The screenshot above displays a graphical signature that is indicative of a system that scales in proportion to load. This is a classic example of how a system should scale.

The edge-of-capacity is evident – at approximately 10:46am and the end user experience (response time shown in black) increased only slightly as the load request increases.

Increasing the user concurrency (blue line) periodically was used to place additional load on the system. The transactional throughput (green line) reached a maximum of 2.9 (approximate) user transactions/activities per second.

As we increase the load beyond the edge, we can see that the end user response time increases (this is expected as the system is already running at maximum throughput - the system simply cannot perform any more transactional throughput, therefore the response time will increase).

We can also see the corresponding amount of %CPU utilization correlated with the end user experience and transactional throughput. We can see that the average %CPU utilization also scales in proportion to load and is therefore predictable. This is not necessarily true for all systems.