Effective Performance Testing

Part 6: Interpreting Scatter Charts

"The purpose of analytical displays of information is to assist thinking about evidence."

– Edward Tufte, PhD (2003)

(This item originally appeared on The Rational Developer Network, an online community for customers of IBM Rational Software. To find out more, including how you can get a free evaluation to the Rational Developer Network, please visit http://www.rational.com/services/rdn/find_out_more.jsp)

Scatter charts are by far the single most powerful visual evaluation tool at a performance engineer's disposal. Wise use of this simple-seeming type of chart will greatly shorten your search for those often-elusive bottlenecks. This article, the sixth in the "Beyond Performance Testing" series, is an expanded discussion of the scatter chart introduced in Part 6 of the "User Experience, Not Metrics" series.

So far, this is what we've covered in this series

Part 1: Introduction

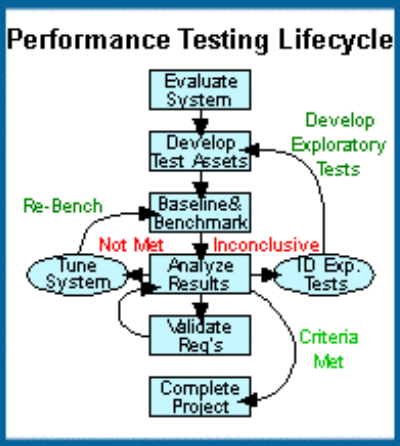
Part 2: A Performance Engineering Strategy

Part 3: How Fast Is Fast Enough?

Part 4: Accounting for User Abandonment

Part 5: Determining the Root Cause of Script Failures

This article is intended for individuals who either report on or analyze performance-related data. I want to preface it by noting that Edward Tufte's seminar and books are the foundation for my thoughts on graphically presenting information. If you aren't familiar with his work, I highly recommend that you visit his Web site. In the "Ask E.T." forum on his site, he states: "The purpose of analytical displays of information is to assist thinking about evidence." The scatter chart, especially the overlaid version I'll show you near the end of this article, is the best aid I know of to thinking about the evidence provided by a performance test.



Creating and Reading Scatter Charts — A Refresher

Let's start this discussion with a review of the basics. If you're already familiar with how to create and read a basic scatter chart, feel free to jump down to the next section, "Analyzing Basic Scatter Charts." If you'd like a quick refresher, keep reading.

Creating a Scatter Chart in TestManager

You can view a basic scatter chart right in the Rational® TestManager software quite simply. After any test execution, click the Resp vs. Time button. Once a chart is displayed, choose View > Settings from the menu bar to get the window shown in Figure 1.

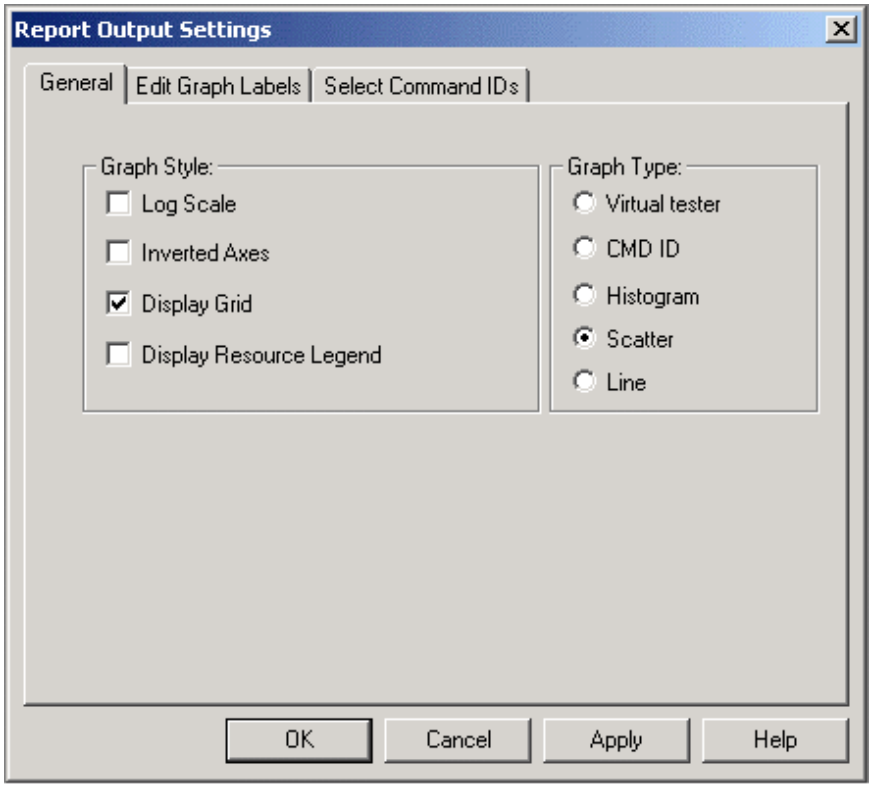


Figure 1: Selecting the scatter graph type

Click the Scatter radio button in the Graph Type list and then click OK. As always, I recommend trying out the different Graph Style options available in this window to see which versions of the chart work best for you. I often edit graph labels on the tab of the same name and select specific command IDs to view separately, but we'll discuss that further below.

Recreating a Scatter Chart in Excel

Details on recreating the basic scatter chart in Excel are given in Part 6 of the "User Experience, Not Metrics" series. Here's a summary of the steps:

- Copy the data from the first three rows (Cmd ID, Ending TS, and Response) of the table below the scatter chart in TestManager and paste that data into an Excel worksheet.
- Highlight columns B and C (Ending TS and Response), then choose Insert > Chart from the menu bar. On the Standard Types tab, in the "Chart types" list, select XY (Scatter), and click Finish.
- Customize the chart using the options available in Excel.

We'll be adding to this basic chart later in the article.

Reading a Scatter Chart

To refresh your memory about how to read a scatter chart, I'll describe what you see in Figure 2.

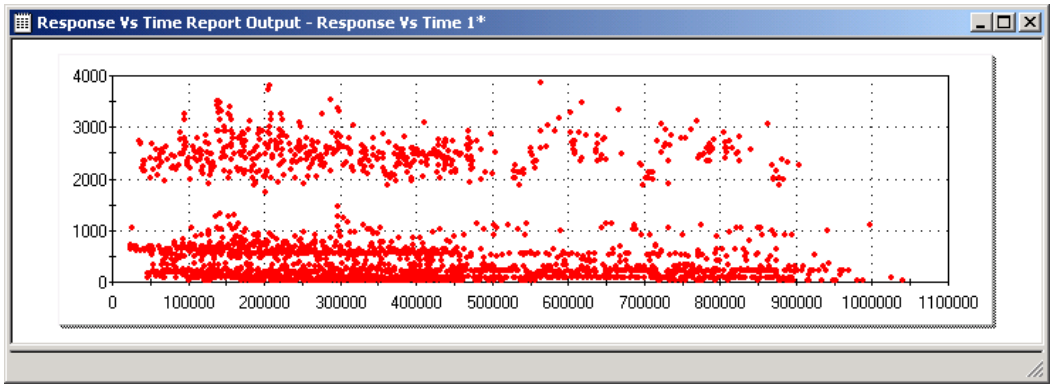


Figure 2: Basic scatter chart

Vertically on the left side is the response time in milliseconds. Across the bottom of the chart is the time since the start of the test run, also in milliseconds. Each red dot represents a command ID or timer measurement. Each measurement is placed on the chart at the intersection of the time since the beginning of the test execution that the timer completes, and the length of time that timer captured (that is, how long it took to receive a response).

In other words, in Figure 2, each red dot represents a page. The point where the dot lines up on the y-axis (vertical) is the number of milliseconds it took that page to load, and the point where the dot lines up on the x-axis (horizontal) is the number of milliseconds from the start of the entire test until that page load was complete. Thus, the dots appear sequentially in time from left to right.

Analyzing Basic Scatter Charts

Analyzing any kind of scatter chart, basic or not, isn't a simple process. Nor is it easy to explain in an article. I do a half-day interactive seminar on creating and analyzing scatter charts, and that's just an introduction. It took my mentors about a year and a half to convince me of the value of these charts and teach me how to glean information from them. But what follows should be enough to convey the most important concepts: Use scatter charts to identify patterns in performance over time. Once you've identified a pattern visually, use all available resources to determine what caused the pattern.

I'll say a little bit about the analysis process and what you're looking for just to get you started. Then I'll give examples of some of the most common types of patterns I've encountered. But as you can imagine, the number of possible patterns we could run across in a scatter chart, and the number of possible causes for those patterns, is virtually unlimited. Again, the main thing for you to take away from this is that analyzing scatter charts is really about identifying patterns, determining what symptoms these represent, speculating about the causes of those symptoms, and finding a way to validate those speculations.

The Analysis Process

To have a scatter chart to analyze, we must first execute a performance test. While that may seem obvious, there are some important points to make about this initial step. To be able to analyze a scatter chart with any reasonable level of effectiveness, we must first not only execute a performance test but also have a thorough understanding of the test we're executing. While it may be true that I can look at a scatter chart from a test that I know nothing about and successfully guess what the cause of the visible pattern was, it's still just a guess, no matter how often I may guess correctly. That may count as good intuition, but it doesn't qualify as analysis.

Some of the things you need to be intimately familiar with to begin effective analysis of a scatter chart are listed below. This isn't an all-inclusive list, but as you'll see in the sections to follow, knowledge of each of these items can change the conclusions drawn about a particular chart dramatically.

- user arrival rate and distribution
- user delay times and distributions
- activities being performed and distribution of those activities
- single-user (and multiuser if available) performance benchmarks for each activity
- number and type of architectural tiers in the system under test

The next part of the analysis process is all about pattern detection. Detection of patterns (and technically antipatterns) is the cornerstone of all types of performance analysis, not just scatter chart analysis. In the case of scatter charts, there are several types of visual patterns that we want to evaluate. I'll demonstrate these with examples below. The types of patterns we'll be looking for include the following:

- patterns by time slice — Are response times different during the beginning, middle and/or end of the test execution?
- patterns by activity — Are the response times for a particular activity consistent throughout the test, or do they follow a pattern?
- patterns by groups of activities — What groups of activities have response times following similar patterns?
- combinations of the above

These visual patterns are representations of the symptoms we want to analyze. The fact that these patterns are identified visually is what gives the scatter chart its power. While we could scan through tables of thousands, or even millions, of datapoints or create mathematical algorithms to try to detect those patterns, the human eye/brain combination is orders of magnitude quicker and more accurate than any algorithm could be for this task.

Once a pattern or "point of interest" is identified visually, the next step is to isolate that pattern and remove the remaining "chart noise." In this context, chart noise includes all of the datapoints representing activities and time slices that contain no points of interest (that is, the ones that look like we expect them to). Removing the chart noise allows us to more clearly evaluate the pattern we're interested in.

The first thing I always do to minimize chart noise is to include only user-defined timers (that is, page load times represented by the timers I've inserted) and not every collected component response (that is, page component times represented by command IDs) in the scatter chart. During analysis, I'll often eliminate timers from the scatter chart that are extremely consistent and not adding value to my analysis. To do this in TestManager, simply return to the Report Output Settings window, go to the Select Command IDs tab, and ensure that only the command IDs and/or timers that you want to view are included in the list box on the right. See Figure 3.

If you export your data to Excel, you can even make each timer appear as a different color, as we discussed in Part 8 of the "User Experience, Not Metrics" series. This can also be useful in pattern detection, but it's a very time-consuming manual process.

That's enough theory. Now let's look at some examples of patterns you might encounter in a scatter chart and how to interpret what they mean.

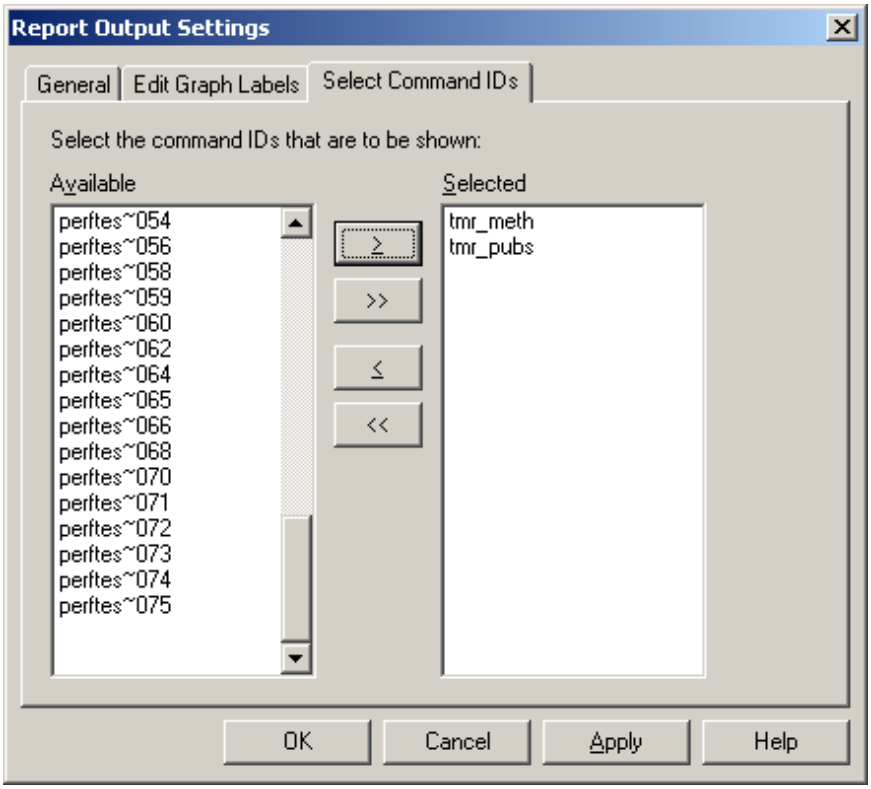


Figure 3: Selecting the timers and/or command IDs to be viewed

A "Good" Pattern

We'll start with what I call a "good" pattern. This is a good pattern not because the results have met any requirements or performance is good, but rather because there are no performance anomalies to explore. For example, look at the chart in Figure 4 and you'll see that the performance is consistent throughout the test. If you were to look at the performance report output table, you would expect to see very small standard deviations on response times.

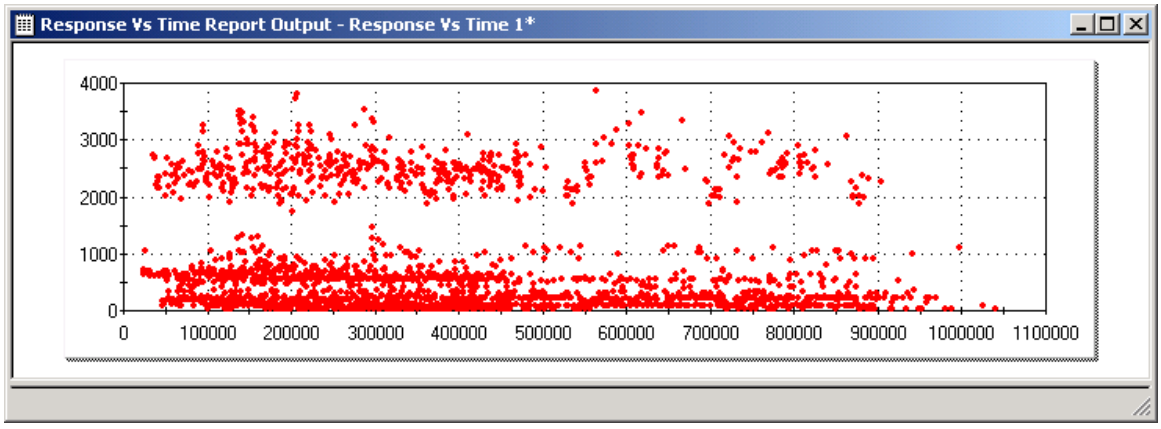


Figure 4: A "good" pattern

When you see a chart with a pattern like this one, you'll want to verify that your test run was essentially error free and then move on to comparing actual loads, response times, and component metrics with requirements and expectations and benchmark measurements. A good pattern actually provides little value in terms of analysis, but it provides a lot of value in validating your test.

A "Banding" Pattern

A "banding" pattern is one that displays obvious horizontal bands of response times. These bands almost always correspond to individual pages. In Figure 5, there are three obvious bands. The top band corresponds to users logging on to the system and displaying the home page. The middle band corresponds to the initial page that displays the areas where users enter and submit their user names and passwords. The bottom band corresponds to logging off from the home page. I was able to determine that by (1) knowing that those were the only three activities being performed in this test (a benchmark of the logon activity) and (2) using the scatter chart filtering options to display the timers one at a time.

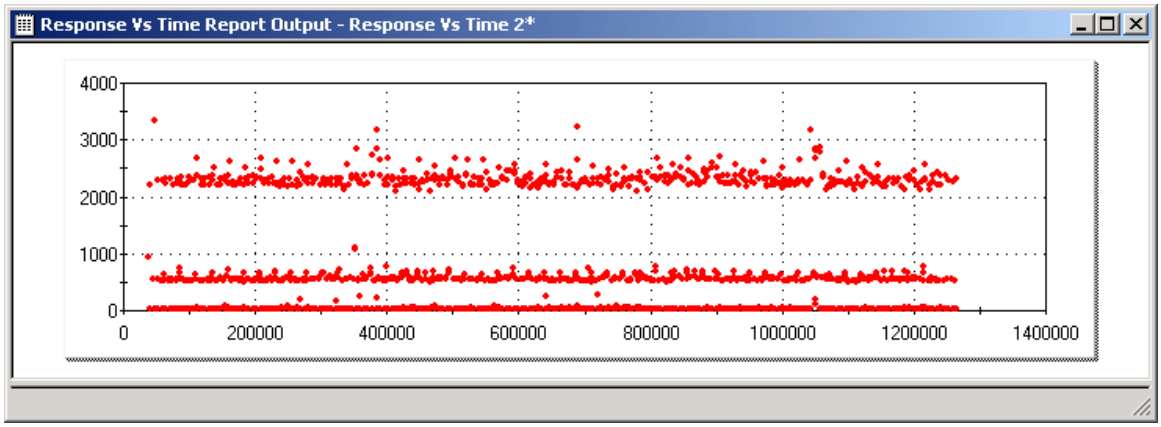


Figure 5: A "banding" pattern

In this case, a banding pattern was what I was hoping for, although the space between the bands was greater than desired, identifying an area for performance improvement. This scatter chart demonstrated that the logon function took about 2.5 seconds under this particular load (considered "very good" by the client). It also demonstrated that the home page took only about .5 seconds to load (also considered "very good" by the client). More important, though, it showed that the logon function took about 2 seconds longer than it took to load a static page. Since we knew that the logon function required the Web server to send a request to the authentication server to verify the user's credentials, we were able to speculate (and later confirm) that this round trip from Web server to authentication server took 2 seconds. This was determined to be "too long" by the client and became an area where tuning took place.

As you can imagine, instrumenting your environment to collect that "inter-tier" response time is often not insignificant (we'll discuss how to do that in detail in Part 9 of this series). Analyzing the scatter chart allowed us to be reasonably certain that doing this instrumentation would be a valuable exercise.

An "Outlier" Pattern

The "outlier" pattern is demonstrated in Figure 6. I defined outliers and discussed how to eliminate them from your analyses in Part 6 of "User Experience, Not Metrics," so I won't spend much time on the topic here, but there are some comments I believe should be added for completeness.

As you can see, Figure 6 clearly has two datapoints that are candidates to be eliminated as outliers. But before we eliminate these datapoints, we should make some effort to determine why the response times were so far outside of the pattern for the test. The first thing we should do is reexecute the test under as close to identical conditions as possible to see if the datapoints appear consistently. If the candidate outliers are reproducible, it's likely that they aren't true outliers.

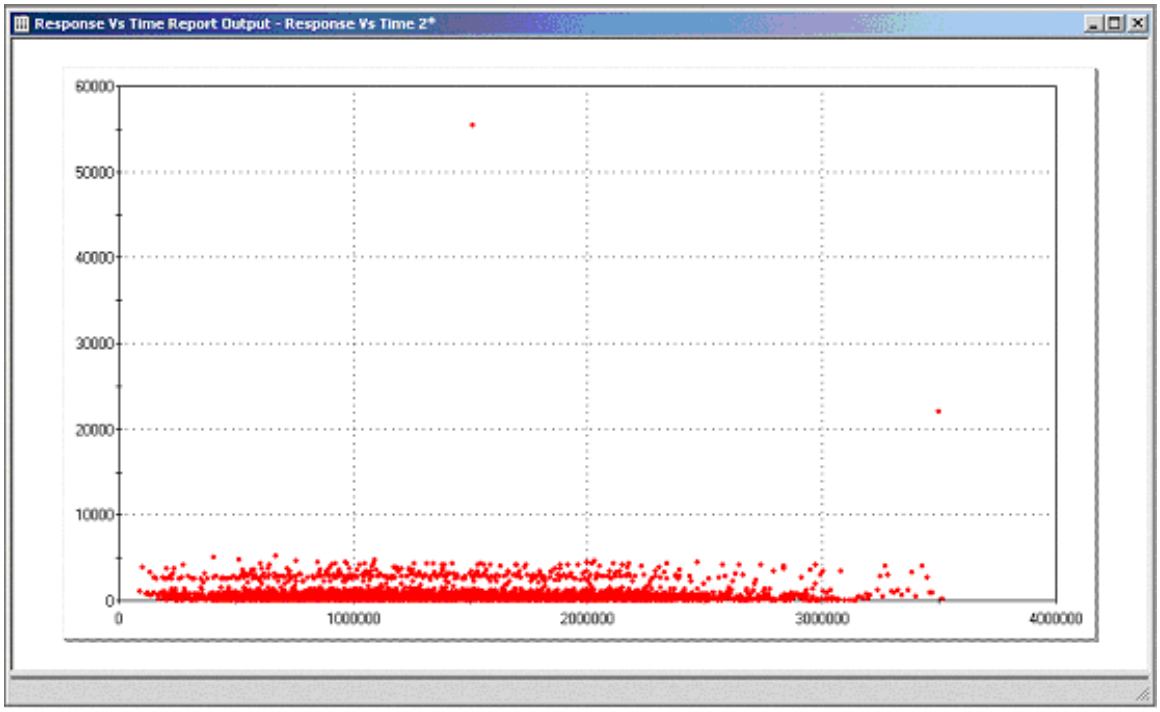


Figure 6: An "outlier" pattern

The reason I mention this is that Figure 6 actually represents the same test as Figure 5, but with a heavier load. When I executed the test several times, I found that I regularly got one or two outliers, but they were often in different places during the test run. Upon further analysis, I found that I had one set of incorrect user credentials in my datapool. This candidate outlier was actually performing correctly in response to the improper data I had entered. In this case, analyzing the scatter chart showed us several things:

- We had to double-check our datapool.
- Authentication failure took significantly longer than successful authentication.
- Failing to authenticate a second time (with the same user name) took less time than failing the first time (that is, the credentials for that user name were being cached on the server side even when authentication failed, which was counter to the stated requirements in this case).

In this case, my initial supposition about those two points being statistical outliers was incorrect, but my research to validate that supposition led to findings I may not have come across otherwise.

A "Caching" Pattern

Another common pattern for a scatter chart is what I refer to as the "caching" pattern. The test that resulted in the scatter chart in Figure 7 was consistent from start to finish, but the chart makes it appear as if something very different from the remainder of the test occurred at the beginning of the test.

At the start of the test, we see very clustered, (relatively) slow response times even though we have well-distributed requests. Running this test a second time resulted in a "good" pattern. The explanation for this is quite simple. After the server had been rebooted, the server-side cache had been cleared and none of the JSP pages had been compiled. As a result, all the users to request a page before it was compiled or cached had to wait for that activity to be completed before the page could be served. This is quite common and should be accounted for in at least some of your performance testing if your servers will ever be rebooted, or have their cache cleared, in production without some type of precaching and/or precompiling mechanism in place.

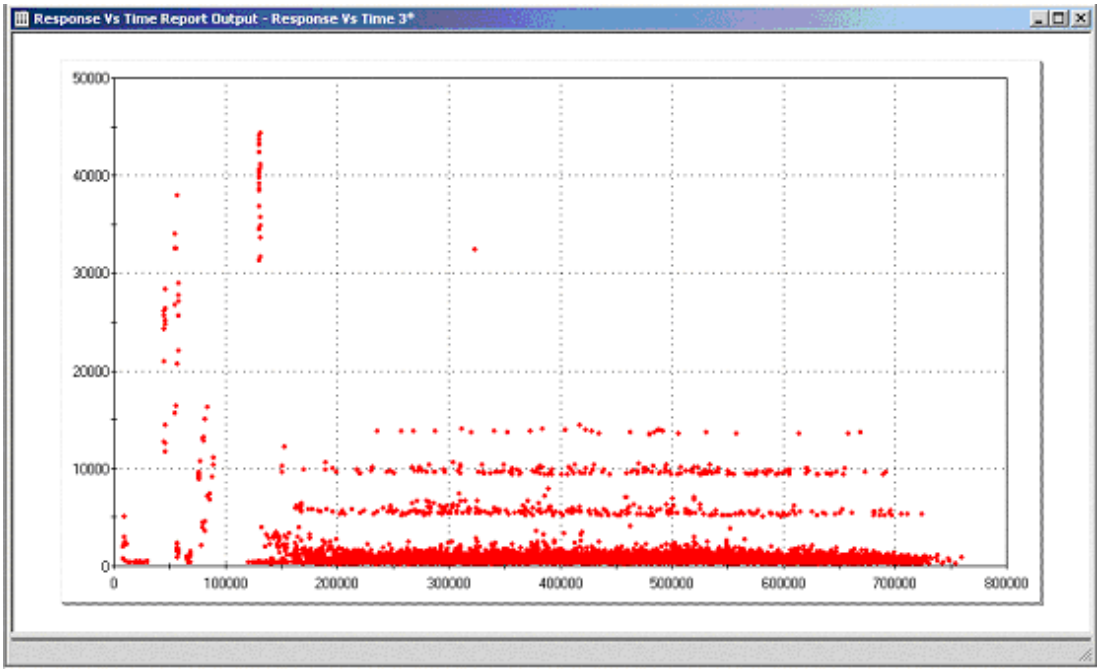


Figure 7: A "caching" pattern

A "Classic Slowdown" Pattern

Figure 8 depicts what I call a "classic slowdown" pattern. This chart shows a steady-state workload and response time at the start of the run, with response times getting increasingly poor until they start to improve again. In this particular case, it turns out that every activity that accessed the application server started to slow down. What had happened was that the application server became overutilized (both CPU and memory) under peak load. The response times began to improve as more users completed their transactions and exited the application (that is, as the total load decreased). This is typical behavior. While oftentimes your test won't run long enough at lower loads at the back end to see the subsequent increase in responsiveness, it's typically there. In the "Creating and Reading Overlaid Scatter Charts" section we'll demonstrate how to correlate CPU and memory utilization with response vs. time scatter charts.

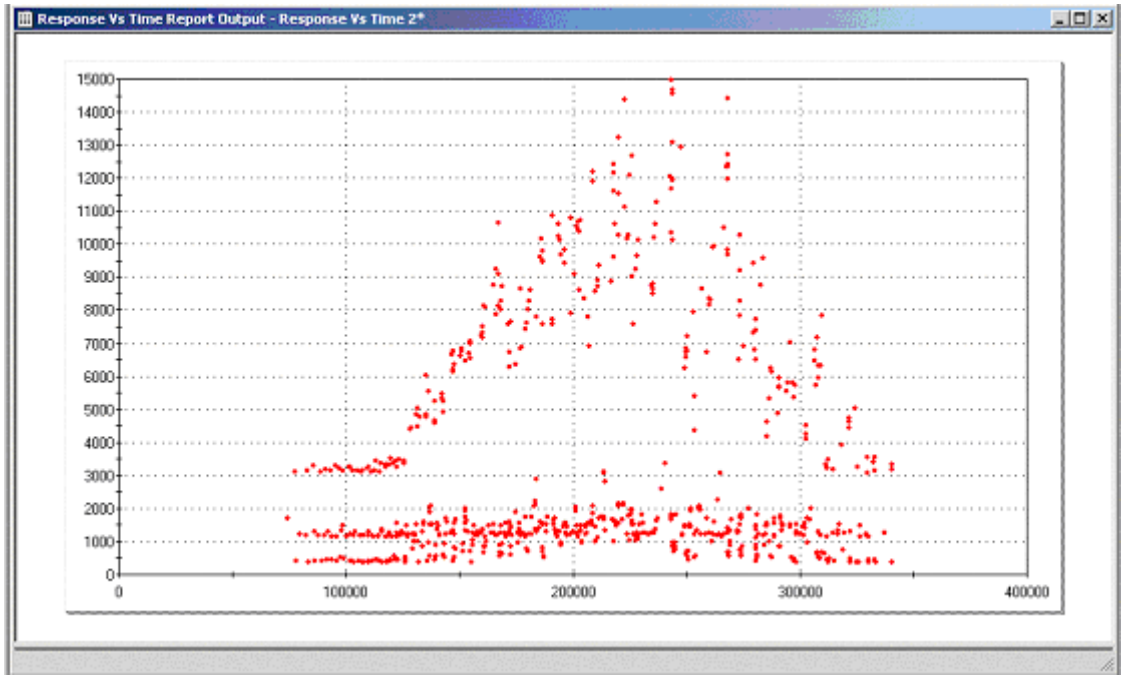


Figure 8: A "classic slowdown" pattern

A "Stacking" Pattern

The "stacking" pattern is one to keep a close eye out for. When a chart demonstrates this pattern, the test is likely to be invalid. That doesn't mean that you can't get a plethora of valuable information from a chart with this type of pattern, but it does mean that you have to analyze it differently. Let's start by looking at Figure 9.

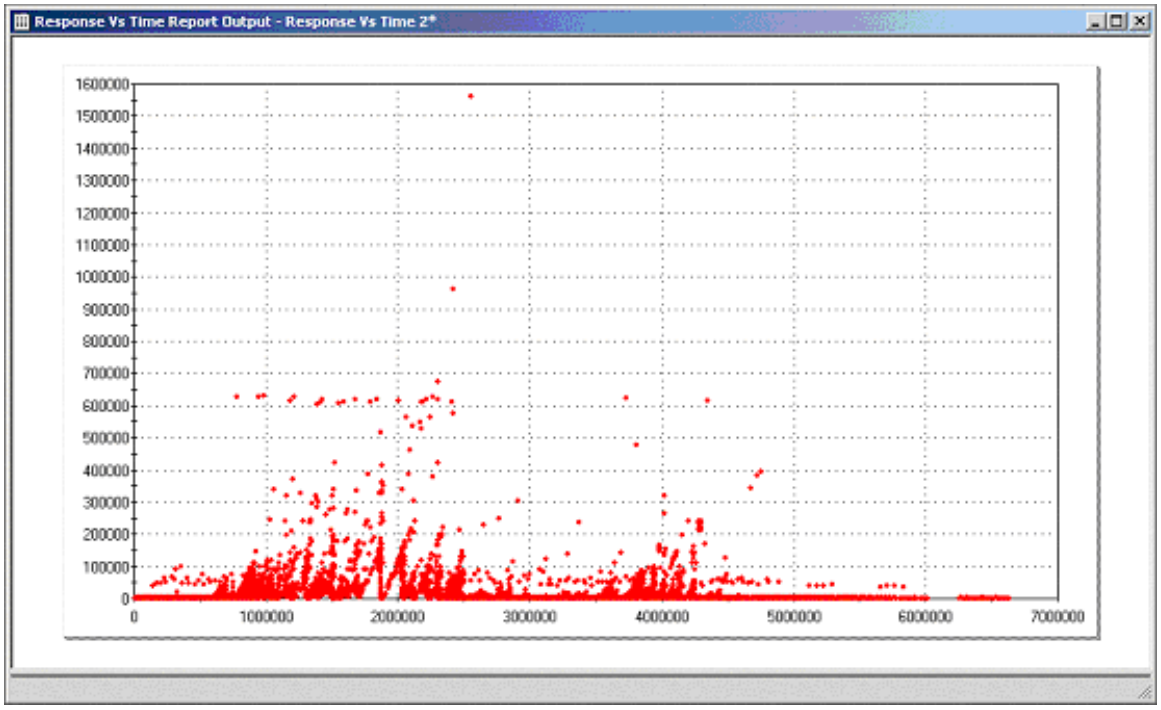


Figure 9: A "stacking" pattern

In this chart the results appear to start out fine but then begin to display a spiking pattern shortly before 1,000,000 milliseconds into the test. You can see that pattern most clearly at around 2,000,000 milliseconds, where the dots form a pattern resembling sharks' teeth or a jagged mountain range. This pattern is extremely uncommon in real-life situations. What it represents is all the virtual users "stacking up" and requesting the same objects at almost exactly the same time.

In Part 2 of "User Experience, Not Metrics," we talked about the importance of simulating realistic (variable) think times for individual users to avoid load scenarios that are equivalent to "putting ten users in a room with ten identical computers with a coach yelling 'On your mark . . . get set . . . Click "Home Page" . . . wait . . . wait . . . Click "Page1."'" What the stacks on the chart in Figure 9 are indicating is that this is exactly the scenario that we've created. This generally happens for one of two reasons:

- We didn't model our think times and/or abandonment correctly.
- At some point something happened to the application that caused all of the users to "stack up."

In the case of this chart, it was the second reason. The way we can tell is by locating the first stack (see Figure 10).

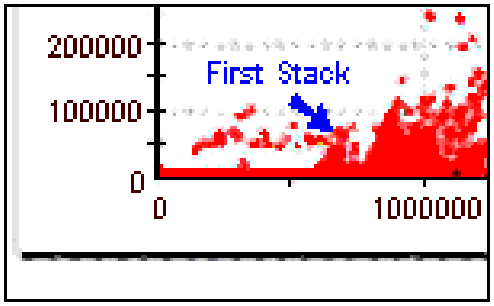


Figure 10: First stack from Figure 9

In Figure 10 you can see that the test ran for more than 500,000 milliseconds (8 minutes) before the first stack occurred. Our test scripts averaged about 10 minutes, so we were more than halfway through our first iteration when this occurred, and thus it wasn't the model. Rather, we had exploited a database connectivity bottleneck that caused users to stack up on one another. Once they stacked up, our delays weren't random enough to overcome it, so the remainder of the test simulated those coached testers we mentioned above.

So why am I making such a big deal about this? The value of this test run ends at the top of the first stack, because everything that happened after that wouldn't happen in real life. Real users would abandon, hit the Refresh button, start over, and so forth. Our scripts simply aren't sophisticated enough to instantly morph from expected case to "weird exception case" in the middle of a test execution. Don't waste your time figuring out what happened after the first stack. Figure out (and fix) whatever caused the first stack, then run your test again. That should result in a chart with a pattern other than the stacking one.

A "Compound" Pattern

There's one more pattern we should discuss before we move on to consider the overlaid scatter chart, and that's a "compound" pattern. Basically, a compound pattern is any combination of the patterns we've already talked about and/or of the patterns we haven't talked about.

Figure 11 depicts a test run with "caching" at the front, a "good" run for a period after that, then a mostly "classic slowdown" toward the midway point. We also have what appears to be a "stack" at about the 600,000-millisecond mark. This may or may not be of interest since the pattern doesn't repeat itself, and it's obviously after other things have started happening that deserve further research.

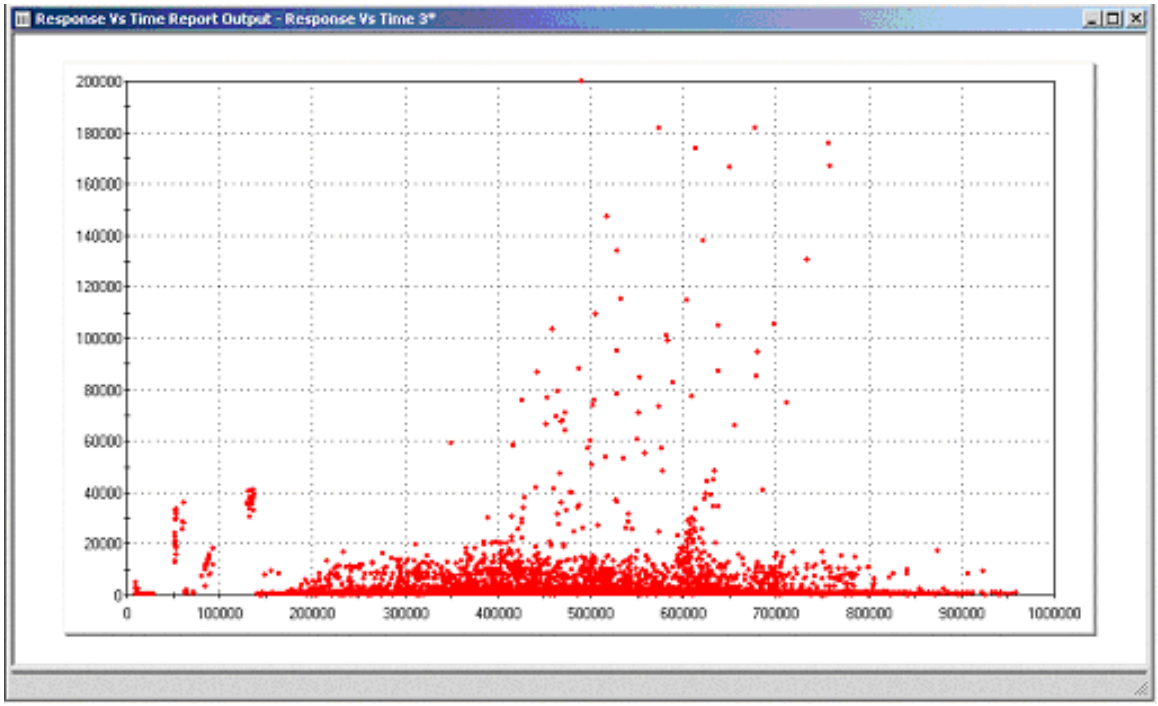


Figure 11: A "compound" pattern

We can try to determine the causes of the compound pattern we see in this chart by overlaying various resource measurements on the chart. That's what I'm going to show you how to do next.

Creating and Reading Overlaid Scatter Charts

What I'm calling overlaid scatter charts are really just basic scatter charts with other data such as CPU and memory utilization — basically any measurement that could be collected by a program such as Perfmon (the Windows resource performance monitoring program) — overlaid on them. Creating one of these charts often takes much more effort, but this type of chart is very valuable in many circumstances. You'll see why when I show some examples of overlaying various resource measurements on our compound pattern after briefly describing how to create overlaid scatter charts in TestManager and Excel.

Creating Overlaid Scatter Charts in TestManager

To collect resource statistics using TestManager, you need to have Rational TestAgent installed on the machine whose resources you want to monitor. This has proven to be a major stumbling block for the clients of mine who simply won't allow anything to be loaded onto their Web, application, and/or database servers. If that's your situation, feel free to skip to the next section.

The basic steps to collecting resource statistics and displaying them in TestManager are as follows:

1) Install the correct version of Rational TestAgent on each machine you want to collect statistics about.

2) Ensure that the `Info Server` command is included in your script(s) and is pointed to the correct machines.

3) Ensure that when you run your suite, the "Monitor resources" check box is checked in the Run Suite window (see Figure 12).

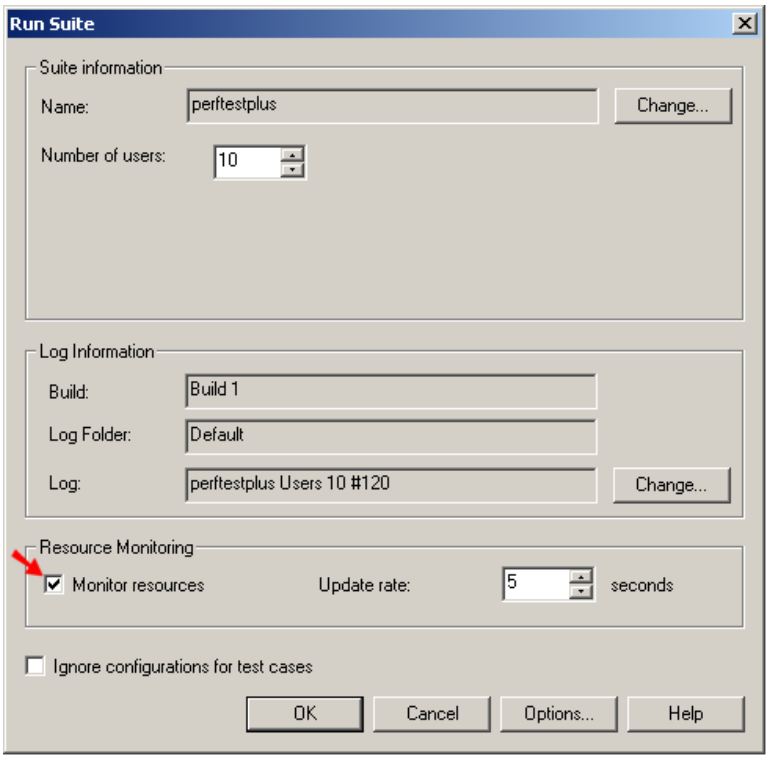


Figure 12: Checking "Monitor resources"

4) After test execution, run the Resp vs. Time report with the scatter option selected, right-click the scatter chart, and choose Show Resource (Figure 13).

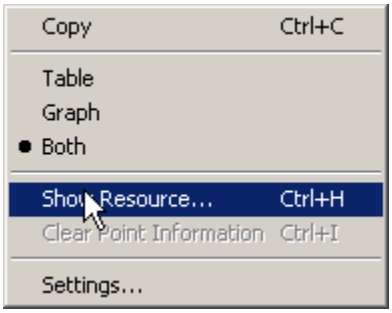


Figure 13: Choosing Show Resource

5) In the dialog window that appears, check the computers and resources you want to have overlaid on the scatter chart (Figure 14).

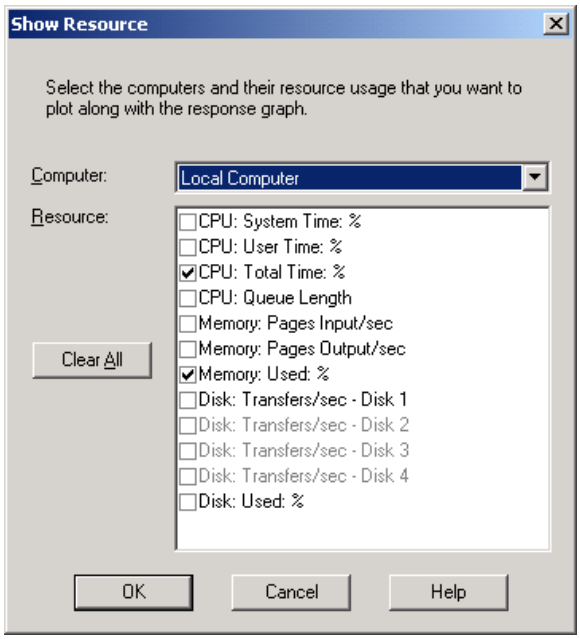


Figure 14: Checking the resources to be shown

Once you've done all that, you'll have a scatter chart with overlaid resources. Figure 15 shows a simple scatter chart created using this method.

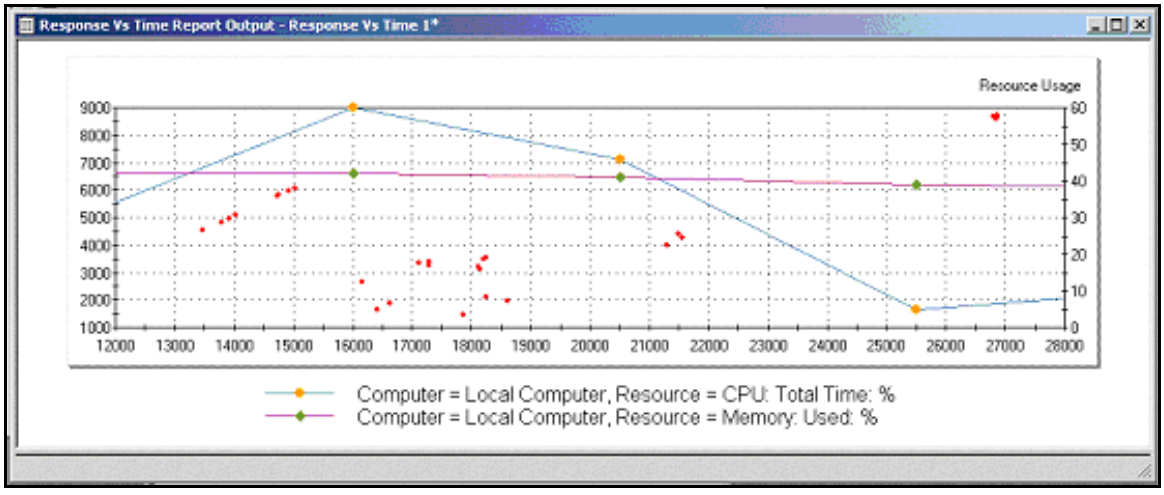


Figure 15: TestManager scatter chart with resource overlay

I intentionally made this chart simple so we could review how to read the chart without too many distractions. The blue line with yellow datapoints represents the percentage of local computer CPU total time used and the purple line with green datapoints represents the percentage of local computer memory used. A scale labeled "Resource Usage" has been added on the right to read the two new lines against, while the scatter chart dots are still read against the scale on the left. This particular chart shows us that the CPU on the local computer (the master station) gets utilized up to 60% while driving this particular test and that the memory usage stays pretty steady — not very exciting news, but easy to read and understand.

Creating Overlaid Scatter Charts in Excel

If you can't put Rational TestAgent on your servers or you want to overlay some data that you can't collect with Rational TestAgent, you're going to have to build the chart yourself in Excel. The very simplified version of this process is as follows:

1) Copy in your timer and response data from TestManager as described earlier in this article, under "Recreating a Scatter Chart in Excel."

2) Copy your resource utilization data into new rows, then sort by time from start of the test execution. See Figure 16.

Time into Run (sec)	Page Response	App Server CPU Used	App Server Queuing HR
0	2.61		
16	0.36		
18	0.37		
20	0.38		
28		20	0
35		33	0
42	12.62	31	1
42	22.86		

Figure 16: Example Excel table for overlaid scatter chart

3) Highlight the entire table, then choose Insert > Chart from the menu bar. On the Standard Types tab, under Chart Types, select XY (Scatter).

4) Right-click any datapoint representing something other than page response time and choose Format Data Series (Figure 17) to get the Format Data Series window.

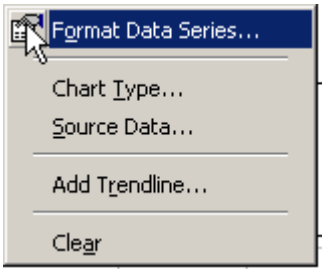


Figure 17: Choosing Format Data Series

5) On the Axis tab in the Format Data Series window, click the "Secondary axis" radio button (Figure 18).

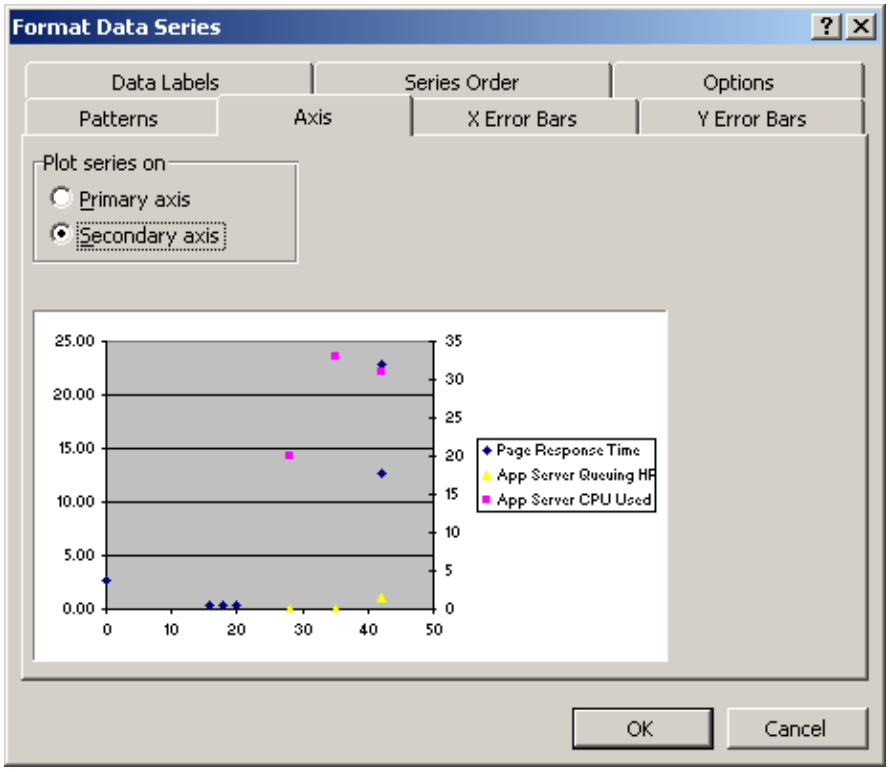


Figure 18: Checking the "Secondary axis" option

6) Repeat steps 4 and 5 for each set of datapoints other than page response time.

Once you've completed those basic steps, you'll have to customize the chart using Excel's standard options. I commonly add a trend line to the resource utilization datapoints.

Examples of Overlaid Scatter Charts

Now we can get back to the compound pattern shown in Figure 11. As I mentioned, we'll overlay various resource measurements on that scatter chart to try to determine why response times slowed down approximately halfway through the test.

The first place I looked was to the Web servers in the load-balanced system under test. While I looked at many different statistics, I chose to show the scatter chart overlaid with CPU utilization data for both Web servers. See Figure 19.

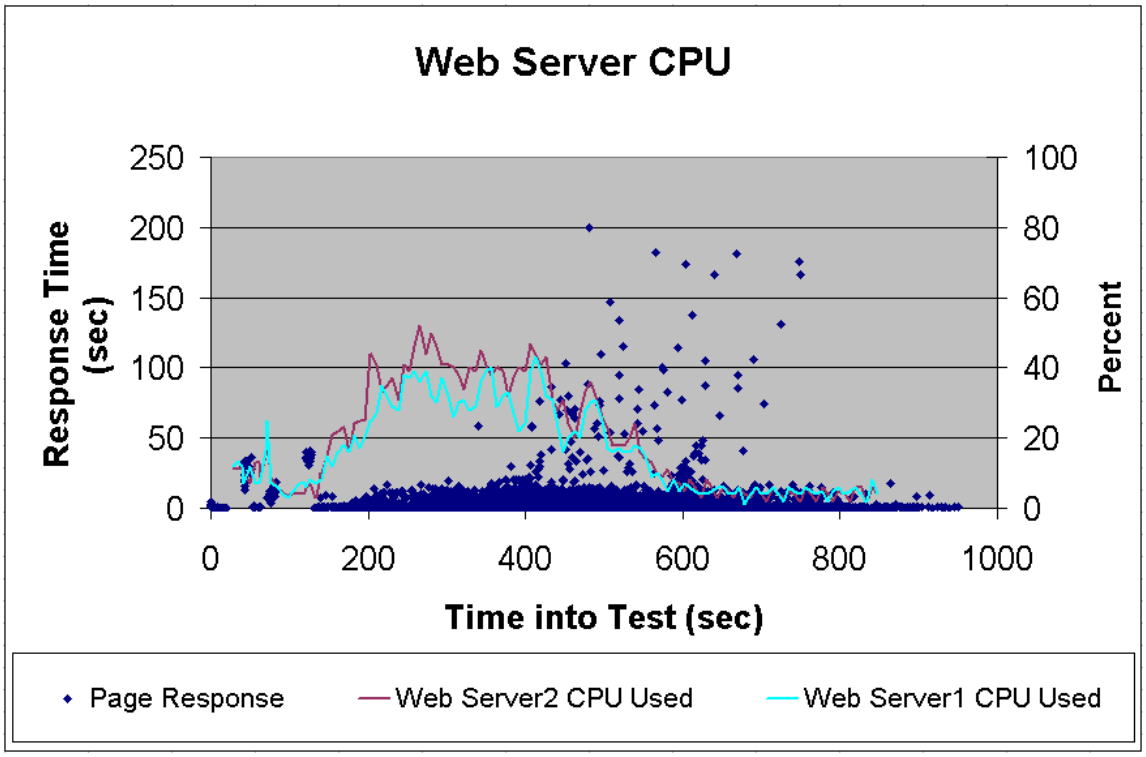


Figure 19: Scatter chart overlaid with Web server CPU utilization data

In this case we can see that neither Web server's CPU utilization went above 50%, and it wasn't at a peak during the period with the results in question, so this wasn't the problem. Next, I checked the application server CPU utilization. See Figure 20.

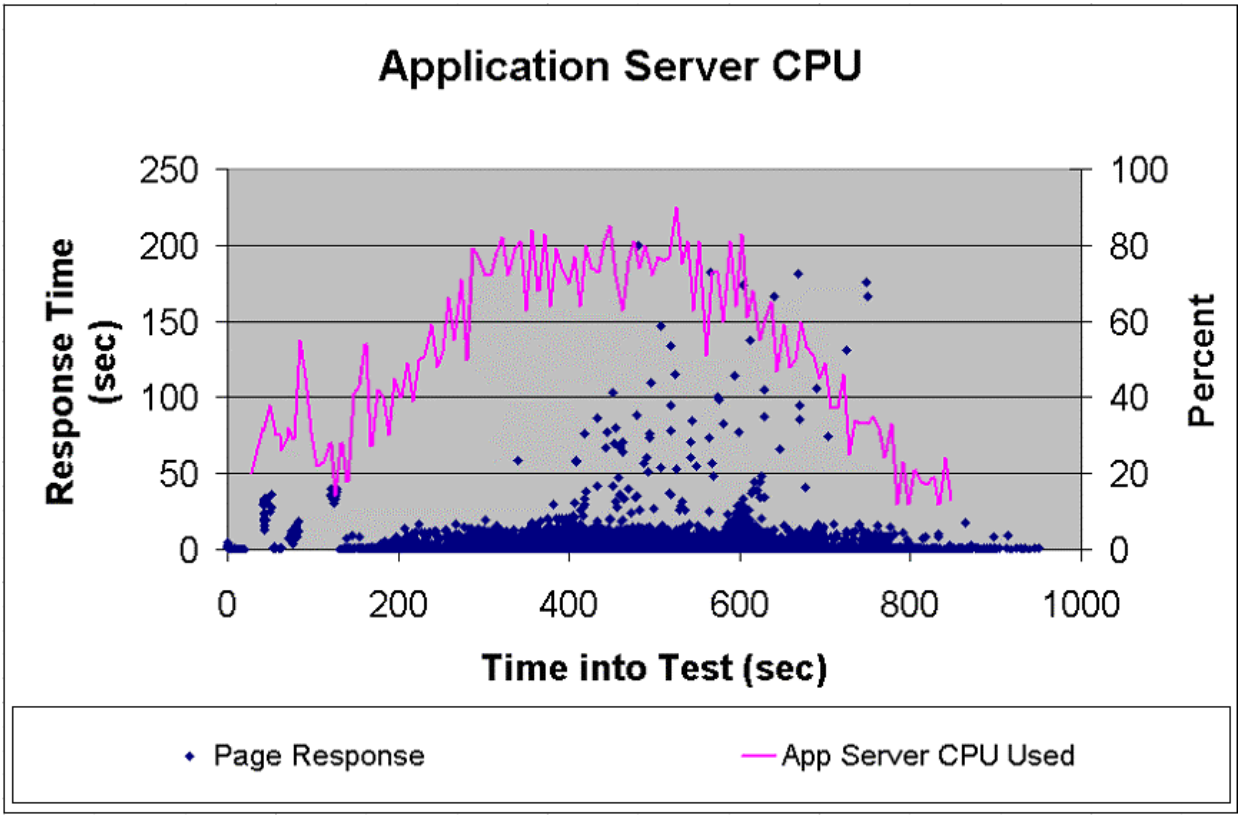


Figure 20: Scatter chart overlaid with application server CPU utilization data

In this figure we can see that the application server hovered around 80% CPU utilization throughout much of the center part of the test. This percentage isn't necessarily bad, and since CPU utilization reached 80% long before the poor performance started, I decided to look at one more resource, the CPU queue length. See Figure 21.

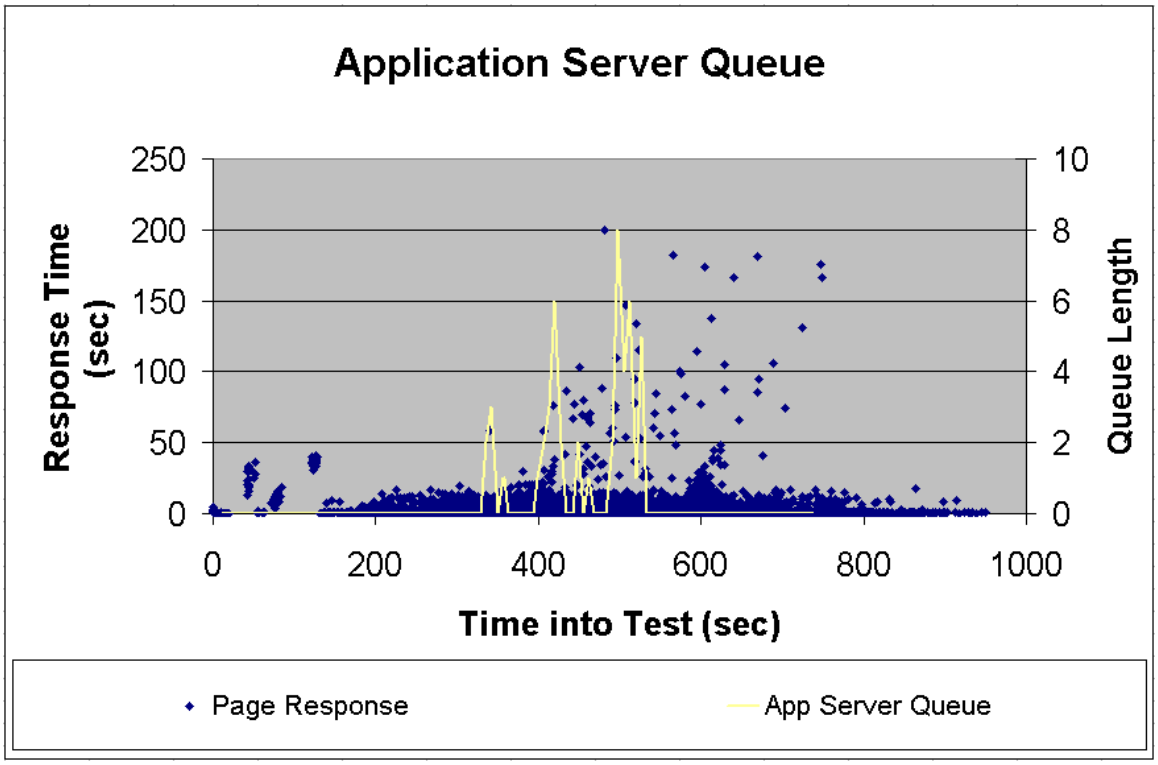


Figure 21: Scatter chart overlaid with application server queue length data

Here we see that the CPU queue went above zero for the first time at the first indication of a performance problem, then continued to read above zero intermittently for the next section of the test. This was the actual cause of the poor performance in this test. After the queue raised above zero for the first time, the application server was unable to completely recover until the load was reduced near the end of the test.

While this analysis could have been done without the use of these overlaid scatter charts, the overlays made what was going on much easier to see. And with very little explanation, even stakeholders with little technical background can see the problem visually. This alone is well worth the time spent to create a graph like this one.

Summing It Up

Scatter charts are extremely powerful analysis tools. No other tool puts more information in front of performance engineers in a way that they can process as quickly. Scatter charts are also a great way to display critical results to stakeholders in a way they can conceptually, if not technically, understand immediately. The keys to analyzing these charts are understanding both your system and your test, and identifying and determining the causes of patterns.

About the Author

Scott Barber is a System Test Engineer and Quality Assurance Manager for AuthenTec, Inc. and a member of the Technical Advisory Board for Stanley-Reid Consulting, Inc. With a background in network architecture, systems design, database design and administration, programming, and management, Scott has become a recognized thought leader in the context-driven school of the software testing industry. Before joining AuthenTec, he was a consultant specializing in Performance Testing/Analysis, a Company Commander in the United States Army, a DBA and a Government Contractor in the transportation industry.

Scott is a co-founder of WOPR (the Workshop on Performance and Reliability), a semi-annual gathering of performance testing experts from around the world, a member of the Context-Driven School of Software Testing and a signatory of the Agile Manifesto. He is a Discussion Facilitator in the Rational Developer Network public forums and a moderator for the Performance Testing and Rational TestStudio related forums on QAForums.com. Scott's Web site complements this series. Please visit it to find more detail on some topics and view slides from various presentations he's given recently. You can address questions/comments to him on either forum or contact him directly via e-mail.

AuthenTec, Inc. is a leading semiconductor company providing advanced biometric fingerprint sensors to the PC, wireless, PDA, access control and automotive markets. AuthenTec's FingerLoc and EntréPad product families utilize the Company's patented TruePrint™ technology, the first technology capable of imaging everyone under virtually any condition.

Stanley Reid Consulting, Inc. is a small, niche consulting company that focuses on IT organizational improvement and recruiting and staffing of technical experts. Stanley Reid Consulting helps your team achieve consistent, successful delivery of IT projects through the following services: IT Organizational Improvement Consulting, Expert Supplemental Staffing, Technical Recruiting & Placement.