User Experience, Not Metrics

by:
R. Scott Barber

Part 4: Modeling Groups of Users

"The one thing that matters the most is not how your site behaves under theoretical or simulated conditions, but how well it works when you plug it into the wall and let everyone come hit your box from all across the world." (Excerpt from *Website Stress Testing* appearing on ExtremeTech written by Serdar Yegulalp, November, 2001)

The *User Experience, Not Metrics* series focuses on correlating customer satisfaction with performance as experienced by external users. In the first two articles, we explored modeling individual external users. This article will expand on the topics we have covered so far to modeling groups or communities of external users. Once again, this article will both discuss the theories behind this kind of modeling, and demonstrate how I have applied these theories using Rational Suite TestStudio on many Performance Testing projects.

This article is intended to provide the reader with all of the necessary theory and practical application to use this approach. This particular topic does not require complex modification to scripts, but does deal extensively with suites in TestManger and assumes the reader is familiar with the creation and use of datapools. The article is intended for all levels of TestStudio users, but will be most useful to Intermediate tool users and above.

Introduction

You may be wondering exactly what this article is meant to discuss. To model groups of users, we simply take all of the individuals that we modeled using the techniques discussed in the last two articles and throw them together into a suite and *SHAZAM,* a user group appears, right? Unfortunately, it's not quite that easy. There are several more issues that should be resolved before a complete and accurate user community model is available for testing. Rather than turning this into a complex mathematical process that would delve deeply into chaos theory (ok, I don't know if it is actually chaos theory or not, I just don't like complex math), we will continue our common sense approach to producing realistic models. Specifically, this article will discuss user community models, usage over time vs. concurrent user load, ramp up/ramp down issues, and data considerations for user communities.

Once again, I must stress that the performance results obtained by applying the concepts presented in this series are only as accurate as your overall user community model. Performance measurements will reflect what a user experiences when accessing the system under the same conditions applied during test, but may not reflect what a user will experience under different

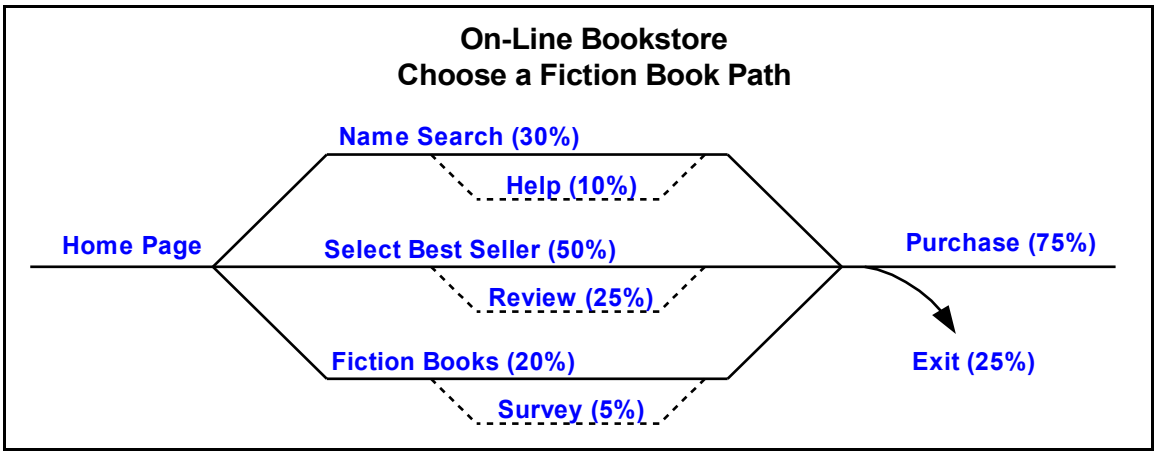
conditions. The next three articles in this series will discuss the actual collection of performance measurements.

Modeling User Communities

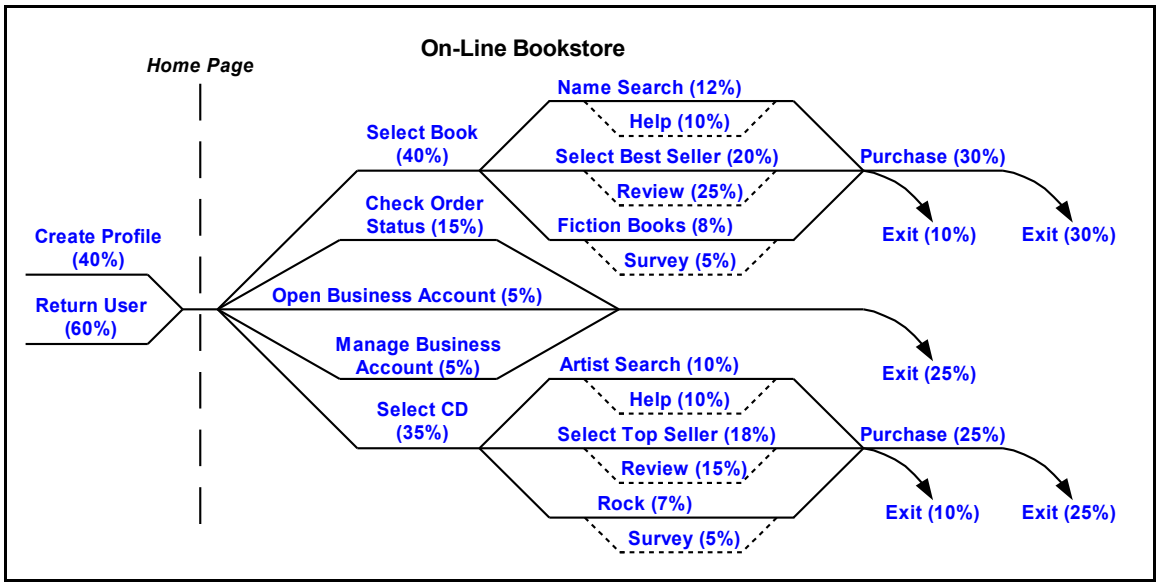
Modeling a user community has some special considerations in addition to those we have already discussed for modeling individual users. Below I will demonstrate how to develop models, how to decide whether one or multiple models are needed, the differences between user rates and concurrent users, and how to handle ramping-up and ramping-down.

Creating a User Community Model

Recalling the online bookstore example from the previous article, you'll remember that we modeled three different paths that an individual user can take to purchase a New York Times Best Seller, and the likelihood of a specific user choosing each path. I've included our final diagram here as a starting point for this article.



In this article we must further model the on-line bookstore to include other normal system users. It is clear that there are other functions a user can perform on this website. Also recall, we had assumed the user modeled above was a return user. First time users would have to enter their personal and billing information and optionally create a username and password before purchasing a book. Users of the online bookstore may also search and purchase CDs or check an order status. In addition, the site could include an option for businesses to open and manage corporate accounts. The diagram below shows the possible usage of the entire site.



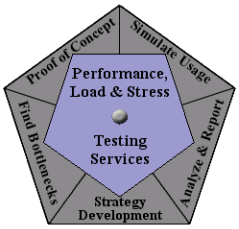
As you can see, there are a ton of activities on this diagram. Let us first address the concepts involved in creating this model that are easiest to explain, then tackle the more difficult issues involved.

Modeling User Distribution Percentages for Multiple Types of Users

Remember that this is a make-believe website, based on my experience with normal online bookstores. Therefore, the percentage distribution of users is also make-believe, based on my own experience as a tester and web user. These percentages represent the percent of total users conducting these activities over some period of time, probably a month. For a discussion on how to determine those percentages, refer to the last article.

Notice as well that the percentages on the book ordering branch are different in the two diagrams above. That is because each percentage in the graph is always normalized over the entire user community. For example, on the Choose a Fiction Book diagram the Name Search activity was conducted by 30% of the users who searched for books. On the Online Bookstore diagram that same activity shows 12% instead of 30%. At a glance that seems to be very different, but notice that only 40% of all users of the site search for a book. Easy enough, 30% of that 40% is 12%. That means that those two models actually represent the same distribution of users within the bookstore's ordering branch. The percentages on the dotted lines have remained the same because they still refer to the percentage of users that reach the solid line branch that they are associated with.

The observant diagram viewer will have realized that only Fiction books and Rock CDs were searched and purchased in this model, which is probably not realistic. In an attempt to create a diagram containing all of the concepts that we need to discuss without making it too small to read, I pretended that the other genres of books and CD's weren't very popular on this particular site and didn't model them. Realistically, it would be very important to model these options if the information about books and CDs of different genres are stored in different tables or databases. It is critical to remember to model system intensive activities, even if they aren't the most popular activities on the website. For instance, in our example, 5% of all user activity was managing business accounts. One may argue that



this activity could be left out of the model, but in this case, it was modeled because part of managing business accounts is generating a monthly usage report. These monthly usage reports require that the system extract and compile large amounts of data from the database and then format the data before displaying it to the user. As you can imagine, one extra user surfing to a static web page will not cause a noticeable change in performance, but one user running this report may cause significant performance degradation. Therefore, these system intensive activities will always need to be modeled and eventually scripted and evaluated.

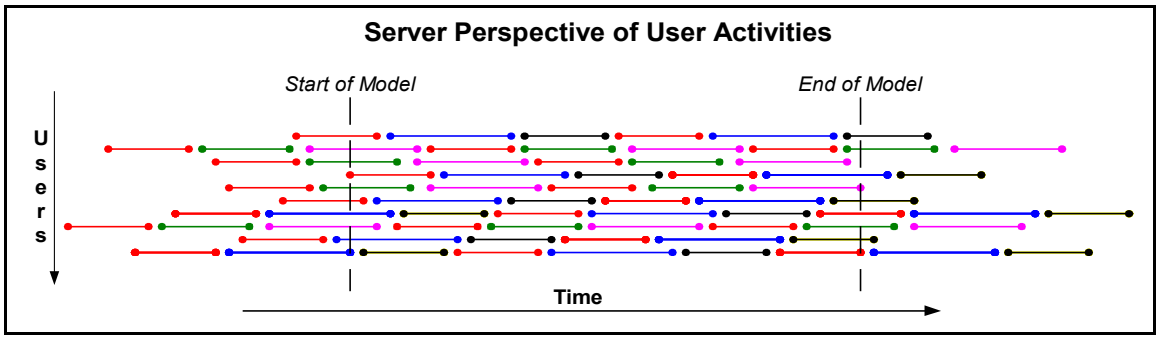
A good model for accurate performance related results will include all system intensive activities and approximately 80% of all other activities.

Viewing Activities from the Server's Perspective

This topic is what I refer to as a "light bulb" topic. What, you may ask, is a "light bulb" topic? It is a topic that just doesn't make any sense until all of the sudden a cartoon light bulb appears over your head and it all makes sense. When my mentor first presented this topic to me, I simply did not get it. After several explanations, the light came on. Now the concept seems completely intuitive. Because of this, I have taken the long-winded approach to explaining. Please forgive me if your light bulb comes on before mine did and you find this discussion to be overkill.

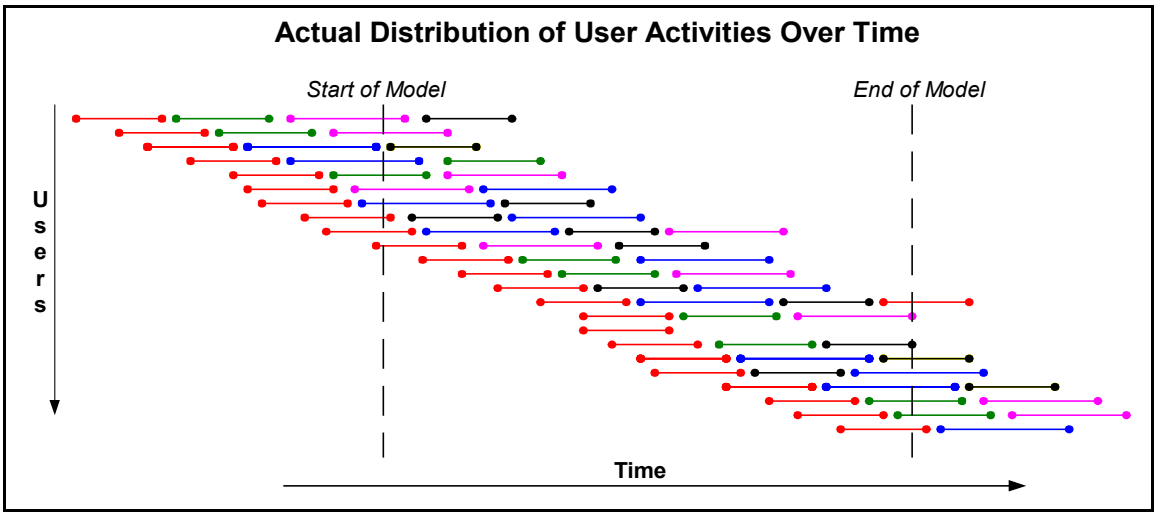
Notice from our example that the only place that new users can create accounts, enter their billing and shipping information, etc is at the beginning of the model (prior to performing other types of activities from the home page). We all know that first-time users on a website don't often create profiles immediately. First-time users will usually navigate the site and eventually create an account and enter personal information only when required to do so to continue, and even then, they sometimes leave the website instead. The first time someone pointed that out to me, I asked, "If you know that, why did you put all of the profile creation at the front of the model?". The answer was simple, "Because it doesn't matter.", but I was confused. Eventually I came to understand that for the results of a performance test to be statistically valid, the correct activities must be executed in the correct distribution over a given period of time, not necessarily distributed by what order an individual user accomplishes the tasks. Below, I demonstrate this concept graphically.

In the graphs below, each line segment represents a user activity, and different activities are represented by different colors. For the sake of this discussion, we will say that the red line segment represents the activity of "Load the Home Page". Users (or possibly sessions or threads) are represented horizontally across the graph. For simplicities sake, we'll assume that the same activity takes the same amount of time for each user. The time elapsed between the Start of Model and End of Model lines is one hour. Let us first look out from the perspective of the server (in this case a web server).



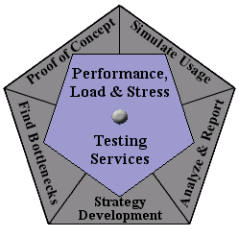
Reading the graph from top to bottom, left to right, we see that user 1 surfs to page "red" then "blue", "black", "red", "blue", and "black". User 2 also starts with page "red, but then goes to "green", "purple", etc. We also notice that virtually any vertical slice of the graph between our start and end times will reveal 10 users accessing the system, showing that this distribution is representative of 10 concurrent users. There are a total of 17 "red", 8 "purple", 11 "black", 11 "blue", and 8 "green" activities that are at least partially within the vertical start and end lines of the time we want to model. What should be clear is that the server only knows that 10 activities are occurring at any moment in time, and that those activities have the distribution just described.

Now look at a distribution of activities by individual user that would generate the server perspective graph above.



In this graph, you can see that 23 individual users have been captured. These users conducted some activity during the time span modeled here. We also see that those 23 users all began interacting with the site at different times. There is no particular pattern to the order of activities, with the exception of all users starting with the "red" activity. These 23 users represent the 10 concurrent users from the server's perspective. In this example, the volume of the test could be expressed as either 23 users per hour or 10 concurrent users.

If we could overlay one of these graphs onto the other, we would see that each activity is distributed



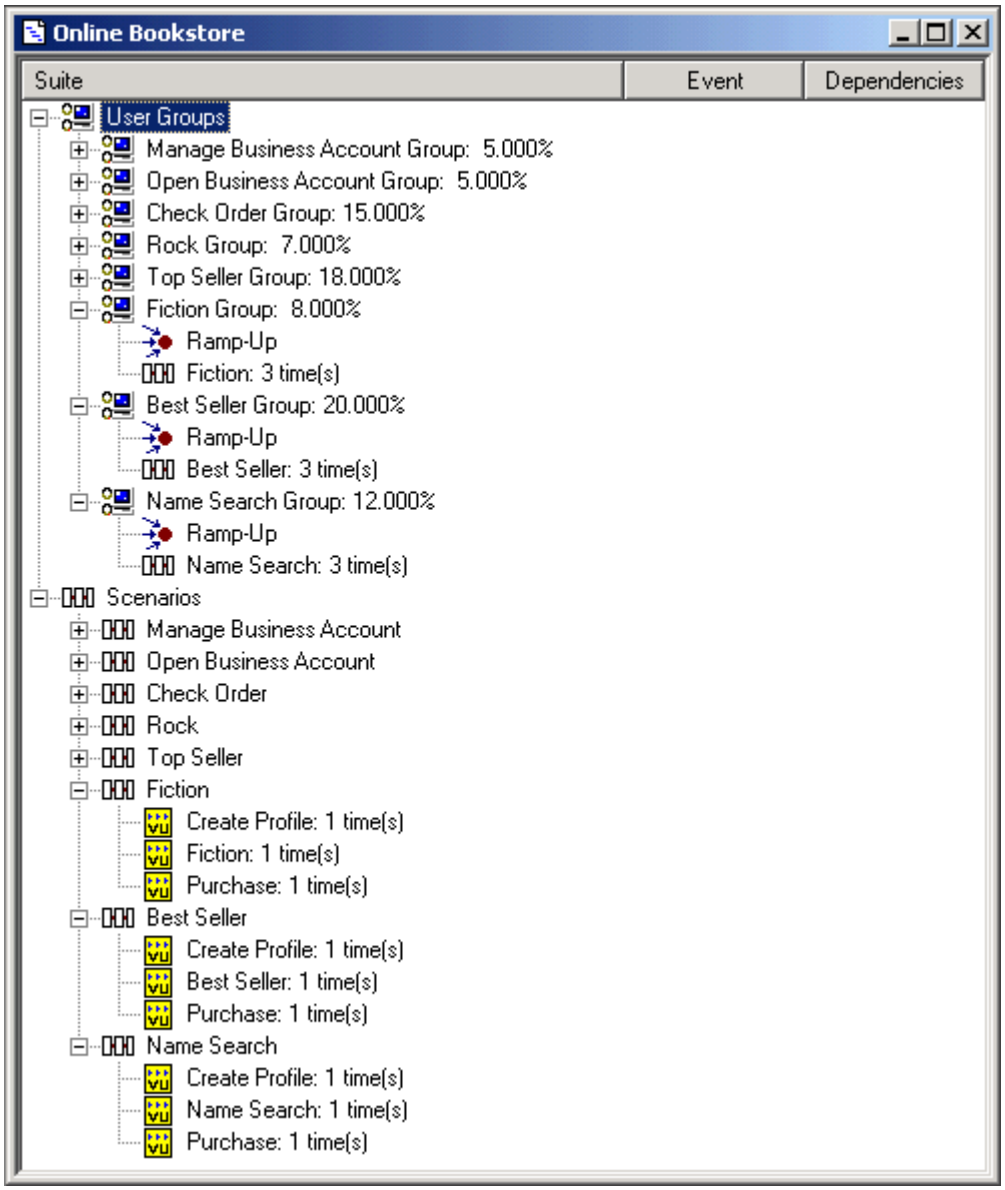
over time nearly identically. When this theory is applied across hundreds or thousands of users, the variations between the actual distributions and the modeled distributions quickly become statistically insignificant.

Modeling Activities from the Server's Perspective

To model the distribution we have been discussing, first record 2 scripts. One navigating "red", "blue", "black" and another navigating "red", "green", "purple". This is because "black" and "blue" occur with the same frequency as do "green" and "purple" with "red" being the start point for all navigation. Then create a suite to execute the first script 11 times, and the second script 8 times distributed evenly distributed over the hour to be modeled.

Applying this concept to the online bookstore model, imagine that the "create profile" activity is the "purple" activity in the graphs above. In the Actual Distribution graph, it is easy to see that "purple" occurred before and after many different activities, but in the Server Perspective graph, it always occurred after "green" and before "red". The truth is that the server can't tell the difference (For any of you jumping up and down screaming about secure session ids and other types of cookies to identify users, see the "For Advanced Users" section at the end of this article to be fully convinced). For that reason, we modeled logging in directly and creating new user profiles (which end with logging in with your new username and password) as separate scripts that will start each scenario. Since both of these activities end at the Home Page, and each of the other scripts start at the Home Page, these can be easily organized into a suite to represent the correct model.

While it seems intuitive to record complete scripts for each possible different navigation route through a system. In our example, that would lead to 102 individual scripts. It often makes sense to record complete scripts for each path when testing a client/server application, but as you can see that this is not really a viable solution for diverse web applications. Last month, we discussed how to limit the number of scripts by using three different recording and editing methods. Using those scripting methods in combination with the modeling concepts above, we ended up with 12 scripts. The suite below shows one way to create this model. I did not expand all of the Groups and Scenarios, but the same methods were used for each. A discussion of this use of synchronization points can be found later in this article.



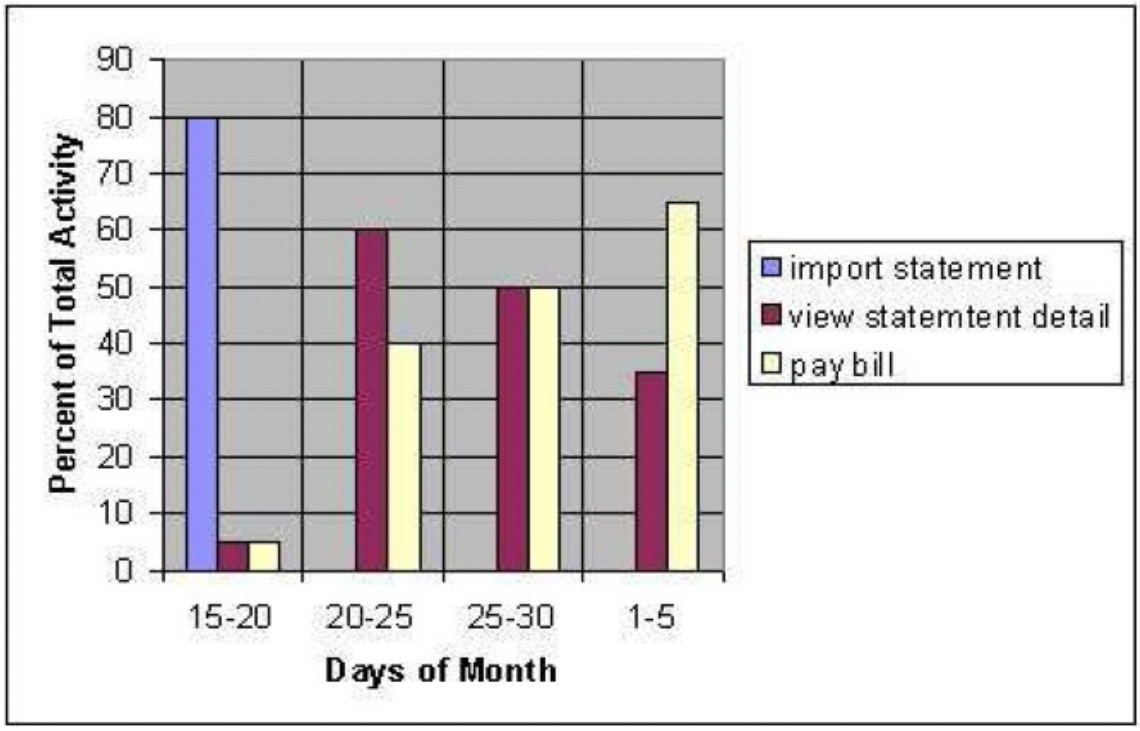
There are other ways that this model could have been scripted and organized into a suite to accurately represent our model. This was simply the method I chose to employ for this example..

I hope each of you now have the cartoon light-bulb shining proudly over your heads!

When Multiple User Community Models are Important

Sometimes, one usage model is not enough. Research and experience tell us that, user activities often vary greatly over time. To ensure test validity, we must validate that activities are evaluated by time of day, day of week, day of month and time of year. As an example, consider an on-line bill payment site. If all bills go out on the 20th of the month, the activity on the site immediately before the 20th will be focused on updating accounts and importing billing information, etc. by system administrators, while

immediately after the 20th, customers will be viewing and paying their bills until the payment due date of the 5th of the next month. See chart below:

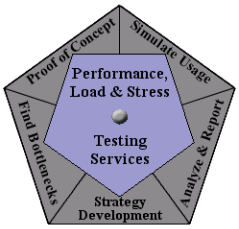


This example shows that the performance of the system could be very different between the 15th and the 20th when account information is being imported as compared to the time between the 20th and the 25th when customers are viewing and paying their bills. In this case, two very different user community models must be created and tested. Applying this concept over various time segments while developing models will ensure most variances in the expected user activity are accounted for.

While developing community models, special models may need to be developed for Sustained Normal and Spike usage conditions. A Sustained Normal usage condition is when the application is being used at normal, or expected loads, but over an extended period of time. A Spike usage condition is when abnormal, or unexpected surges in usage loads occur. Scenarios that drive these conditions vary greatly from one system to another. On some systems, it is valid to simply define the number of users to be executed and the length of time for the load to be applied, using the same model that has been developed for typical usage. For other systems, these may be completely separate models if these usage conditions also involve a significant deviation from the typical usage model (as with the example above).

Comparing Concurrent Users to User Rates

Most record/edit/playback load generation tools are licensed by number of concurrent users. The term concurrent user actually was created to describe client/server systems where users log into the system in the morning and perform activities through out the day. In that case, you can see that if you have



500 system users, they will be accessing that system concurrently throughout the day. This is also true for Rational TestStudio, though Rational refers to these concurrent users as Virtual Testers. Partially because of this, it has become common for the capabilities of systems to be measured in concurrent users. Although this is common, many people new to performance testing misunderstand the meaning of the term.

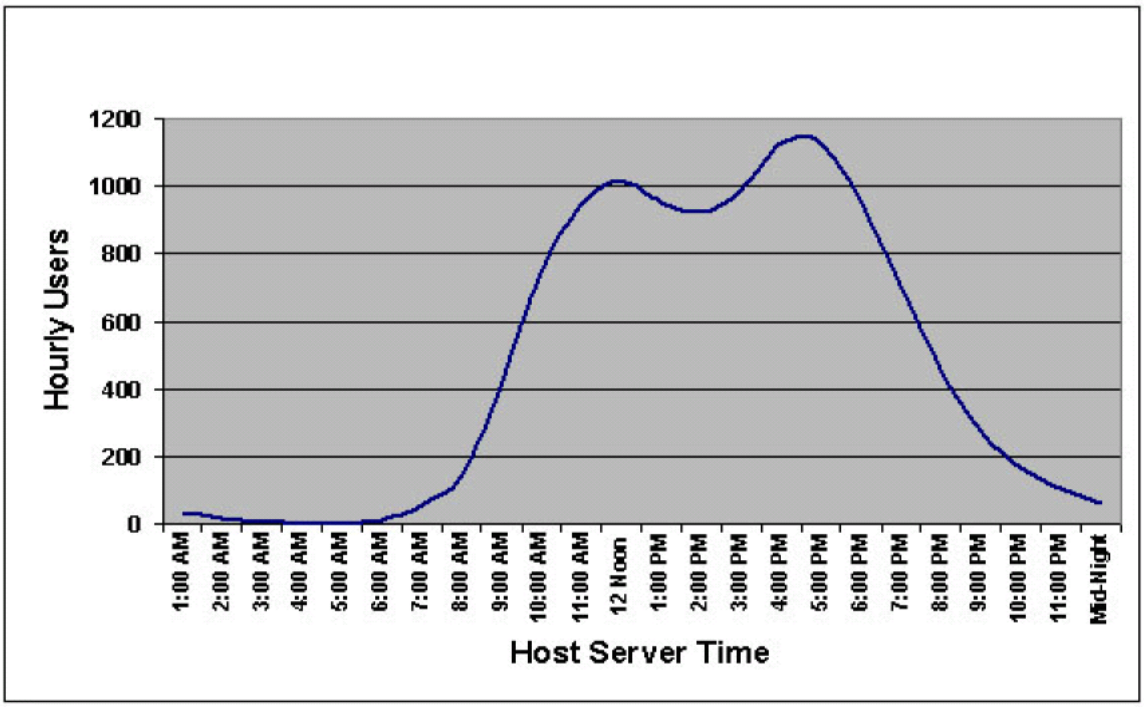
As we saw in our Online Bookstore example, 10 concurrent users, or Virtual Testers, were used to accurately represent 23 users per hour. There is no "rule of thumb" to estimate the number of concurrent users that map to specific hourly or daily user rates. Performance tests need to be created to simulate actual or expected user rates.

One cannot easily estimate the number of concurrent users required to simulate a specific user rate for a particular user model without excessive use of mathematics. Instead of attempting to create this estimate, it is safe to obtain the same number of concurrent user licenses as hourly users intended for the test. This ensures that you are not limited in script creation methods and that you have sufficient concurrent users to conduct specialty tests. Tests such as Stress (high user load distributed evenly over an extended time), Spike (high loads separated by short periods of no load), and Hammer (excessively high loads) are often very useful in detecting bottlenecks and other performance issues, but require the use of more concurrent users than actual or expected usage tests.

Dispelling Ramp-Up/Ramp-Down Myths

I have spent several hours building Lego models while considering the best approach to this section of the article. The concept is really quite simple, but I found it difficult to decide where, exactly, to start (or maybe I just wanted an excuse to build the Lego model my officemate bought me for Christmas). Finally, I decided that I should first define a daily usage pattern for a site. After this, I should demonstrate what the terms "Ramp-Up" and "Ramp-Down" are meant to represent. Next, I should show the method I have devised to account for these concepts. Finally, I should dispel some of the myths and show the shortcomings of some common ways of handling ramping. I decided yes, this is the right order of explanation, and luckily, finished my Lego model in the process!

Assume that the Online Bookstore is almost exclusively used by customers in North America and that the server resides in the Eastern Time Zone. After analyzing the log files generated during a given month, we determine that about 11,000 total customers visit the site a day, that the peak hourly usage is 1,500 and the peak concurrent user load is 500. By plotting the "average" day, along with the rest of the information we gleaned from our imaginary log files, we can generate the typical hourly usage (the number of users conducting activities on the site each hour) chart below.

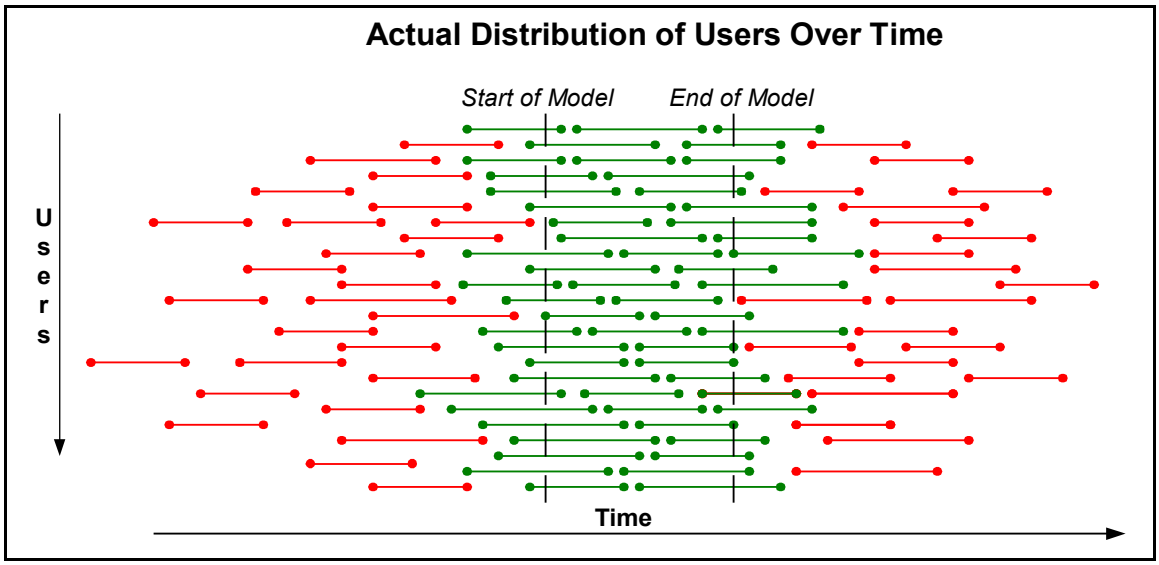


This chart provides us with a lot of information. Immediately, we see that the site is essentially not used between 3:00 AM and 6:00 AM EST. We see that usage peaks at about 5:00 PM EST. We also see that usage ramps up fairly evenly from 8:00 AM to Noon and ramps down fairly evenly from 5:00 PM until Midnight. This information is all very useful for creating performance tests.

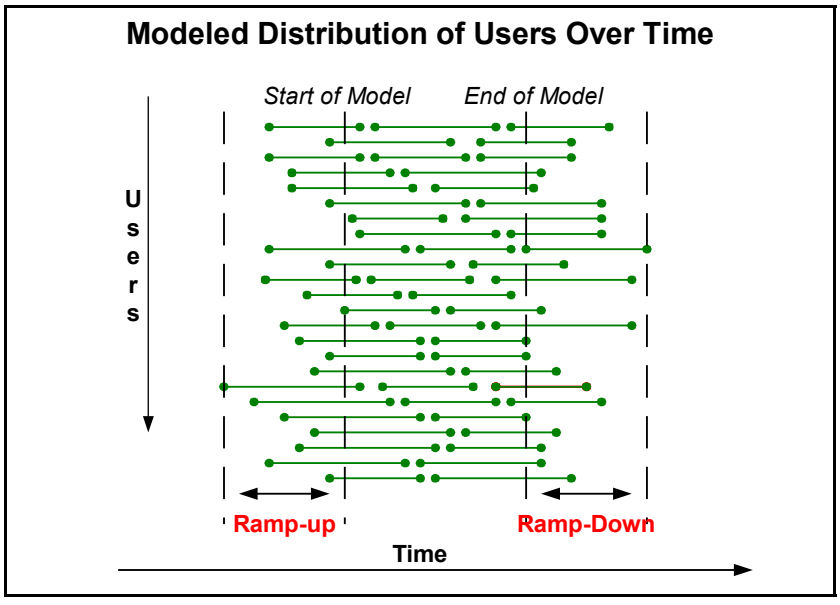
Using this information, one could assume that the goals of testing would be to validate that the application can handle a peak usage of 1,500 hourly users and a sustained usage of 1,000 hourly users over 8 hours. The question becomes "How do I get up to those usage rates, and how do I ensure that users are distributed properly once I get there?" Before I discuss how to do this using TestManager, I should explain the steps involved in modeling the Ramp-Up period and what the added timers will measure during the test.

Ramp-up/down Periods

The chart implies that it takes about 4 hours to ramp-up and 7 hours to ramp-down from the user loads we are interested in testing. In truth the ramp-up and ramp-down period only needs to be about 10 minutes for an accurate test. The graph below is similar to the activity graphs we discussed earlier in the article, but in this case each line segment is an entire user session, rather than an individual activity. The green segments represent users that perform some activity between the model start and stop times. The red segments represent users that perform no activities during the time we wish to model.



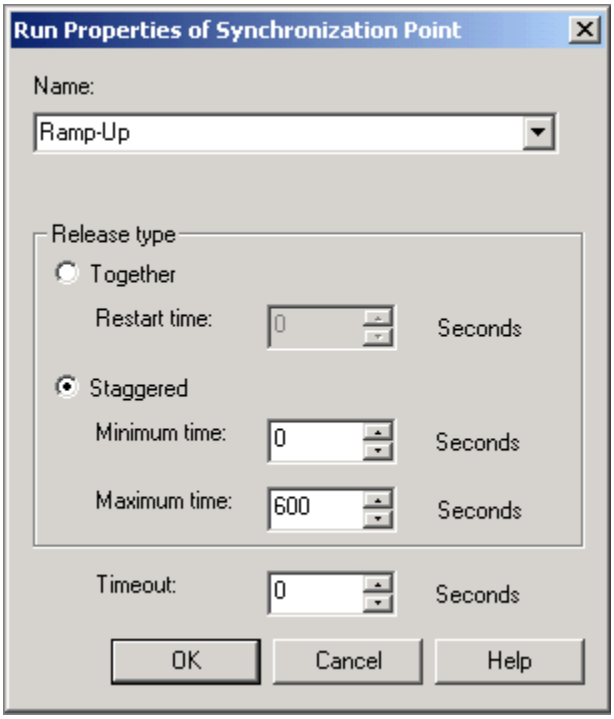
Since we are interested in the performance of the system only during the time between the start and end points, no performance data from outside of that time will be collected (collection of performance data is the focus of next month's article "*What should I time and Where do I put my Timers?*"). Since a user obviously can't start in the middle of their activity on the site, all of the green users need to be included. The ramp-up time becomes the time between the beginning of the first user activity and the start of the model, and the ramp-down time becomes the time between the end of the model and when the last user completes their activity. See the chart below.



It is true that modeling this way does not take into account such things as memory leaks, or other performance issues that occur based on cumulative use. Extended ramping periods sometimes bring performance issues such as these to light. The tests mentioned above (Stress, Spike, Hammer, Extended Peak, etc) account for those types of performance issues.

Modeling Ramping

One way to create a suite to generate a load with this type of ramp-up and ramp-down is to use synchronization points. In the suite above, you already saw where synchronization points were inserted. Refer to the graphic below to see the run properties of those synchronization points.



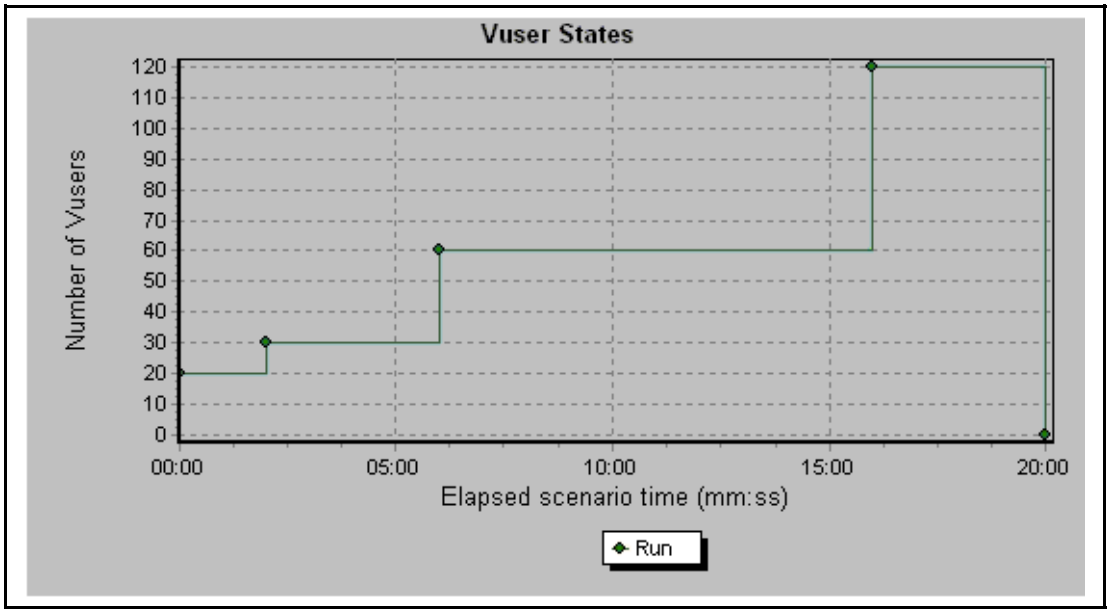
What we see in the suite is that when the performance test is executed, all of the virtual testers are released into user groups by the assigned percentage distribution. Then each virtual tester is released into the synchronization point. While synchronization points are normally used to align virtual testers with one another, in this case they are being used to do just the opposite. They are being used to spread out the times that each virtual tester begins. This synchronization point distributes user start points uniformly over 600 seconds (10 minutes). Each scenario is then repeated 3 times. The big question is where did the 600 seconds figure come from? The time that should be entered into the Maximum time block is the same as the amount of time between the dashed lines labeled Ramp-Up in the Modeled Distribution of Users over Time chart mentioned above. That amount of time can also be interpreted as "a few seconds shorter than the time it takes your longest script to run to completion.", which we can determine by running our script individually and noting the run time. In the case of our example, our imaginary research showed that 600 seconds was the correct time. With only three iterations of each scenario being executed, this test will only run for about 30 minutes. If we wanted to simulate ah hour of actual usage, more iterations would need to be scheduled.

Now we get to discuss the concept of ramp-down. This is actually quite a simple concept. Because the actual time when a user started was staggered over time, they will also complete their task and

different, staggered, times. This accomplishes ramp-down. (I told you it was simple) Ultimately, we will only evaluate times collected between the model start and end time, but that is part of the topic of the next article "*What should I time and Where do I put my Timers*".

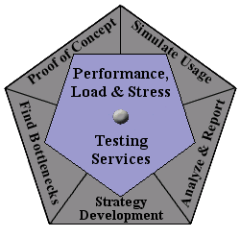
Ramping Myths

So, by discussing that method of handling ramping, we have already addressed one common practice, which would have led to us modeling a 4-hour ramp-up and a 7-hour ramp-down period. The other common approach to ramping is a step-up approach. See the chart below (any of you recognize this chart?) for a graphical representation of this approach.



This model actually has several groups of users that start interacting in the system at the same moment in time. Reading this model from left to right, we see that at exactly the same instant, 20 users request the home page, then at exactly 2 minutes and 30 seconds into the test run, 10 more users request the home page, then 30 more at exactly 6 minutes etc. It is clear that this is not representative of how real user groups would interact with the system and therefore would result in the actual outcome being incorrect, or at least misleading.

One of the reasons that this type of modeling is done is to be able to compare timing results at different user levels during a single test run. This presents the problem of making it impossible to determine if increased times are due to the current total user load or as a result of the cumulative effects of users, which have already exited the system. In the methods we have discussed, separate tests are created to detect performance issues related to load and performance issues related to cumulative effects. A detailed discussion of these types of tests are included in the "*What Tests add Value?*" article to be published in a few months.



Data Considerations for User Communities

It is important to think about data early in the modeling process. For the purpose of this article, I have assumed that the readers know how to physically create datapools for input data requiring variation. The discussions below address what data should be varied and how to handle database issues for test accuracy.

Unique Data Requirements

On the surface, it would seem that all data that a user enters should be varied and unique for each user. There is a flaw with that theory. Every piece of data that is varied using datapools adds overhead to the test, and ultimately limits the total number of virtual testers that can be executed on a given set of hardware. It also greatly increases the time it takes to create tests. If you have infinite hardware and time resources, this would be the most accurate approach, but since most of us have neither infinite time nor hardware resources it is important to limit the varied data to the minimum necessary to create accurate tests. So how does one determine what data needs to be varied and what data doesn't?

First, it is obvious that any data that must be unique to identify a user must be varied, such as username and password, or credit card numbers. Next, data that varies greatly in size needs to be varied. For instance, if we are testing a site that has free text fields that can receive input from 0 to 5000 characters, this field must be varied to ensure that performance based on field size is accounted for. It is not important to simulate all possible data sizes. Rather, four sizes of data distributed across all users is normally sufficient. For example, research may show that 25% of users leave the field empty (empty), 25% enter approximately 250 characters (small), 25% about a thousand (medium), 20% about three thousand (large), and 5% use the full five thousand (jumbo) available. This distribution can easily be simulated with 20 fields in a datapool. This datapool would have 5 each empty, small, and medium entries, 4 large entries, and 1 jumbo entry.

The last type of data that needs to be varied is data that causes the application to respond differently. In our Online Bookstore example, CD genres would need to be varied if each genre is stored in separate database tables. Failing to vary this data during a performance test, in this case, would cause all users to access a single table, thus causing unrealistic stress on the database.

The remainder of the data required to execute the performance tests would normally not require variation. For instance user zip codes can probably all be the same, unless the field will be used later to execute a test requiring server side processing,(like searching by zip code).

Database Maintenance

As with all other components of the system under test, the database should be an accurate representation of the production environment, both in types and volume of data. If the system is live, then a snapshot of the existing database can be taken and used in the test environment. If the site isn't live, a reasonable estimation of the total data volume and data variance must be made. This process may be more or less difficult based on individual applications.

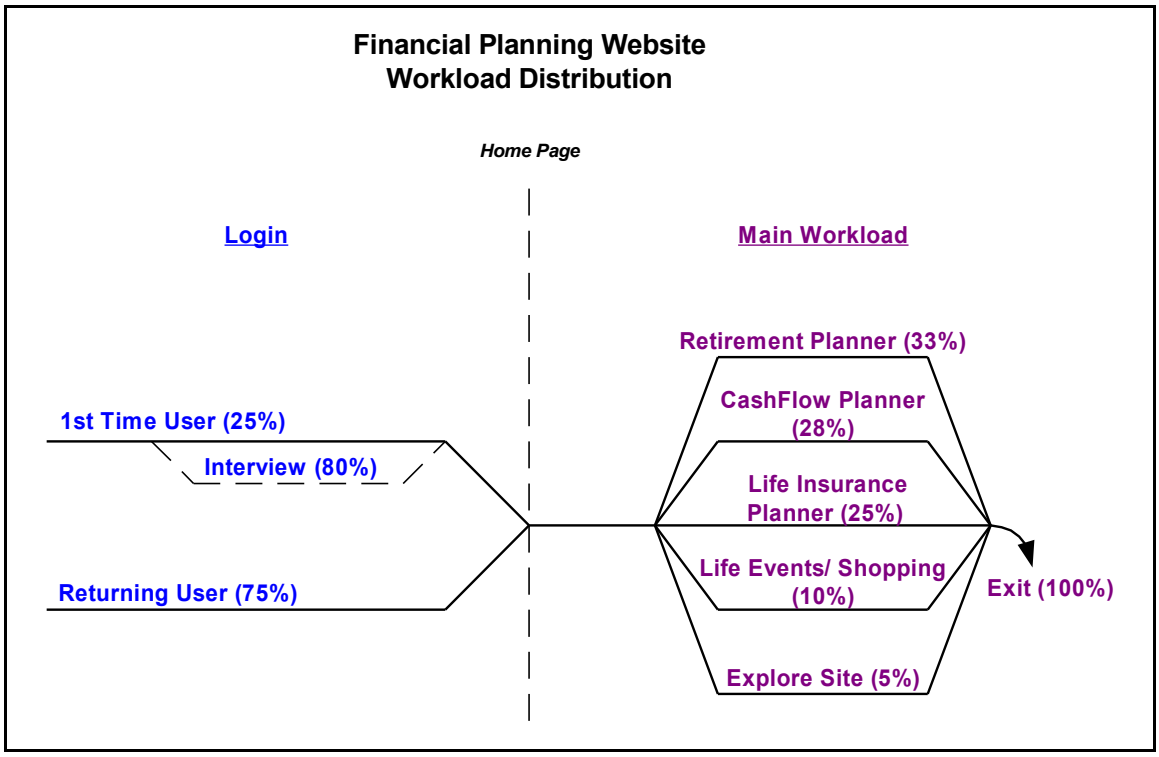
Once the database is initially populated, it must be maintained. Naturally, if the system you are testing doesn't actually change any data in the database, but only reads data, then no further maintenance is

required. If, however, the system under test includes activities that reads and/or writes data to the database, then the database must be reset after each test execution to ensure subsequent tests results are actually comparable to one another.

When time allows, it is recommended that identical tests be executed with differing volumes of data in the database to determine when the volume of stored data becomes the system bottleneck. Generally, the database administrator should be employed to handle database maintenance. Ensure the database administrator is involved early in the modeling process so they can generate data and/or scripts to reset the data while scripts are being recorded, otherwise testing will be stalled while the data issues are managed.

Now You Try It

For those of you who have been doing all of the sample exercises, I must warn you that these exercises are a little different than the previous ones. Ideally, I would attach log files, WebTrends reports, market research and other materials, have you analyze them, create a model, then compare it with the model I created. I'm sure you can see that this is not realistic. I do encourage you to evaluate some applications and develop your own models, but for the following two exercises, I have provided the model below and some usage data for you to work with.



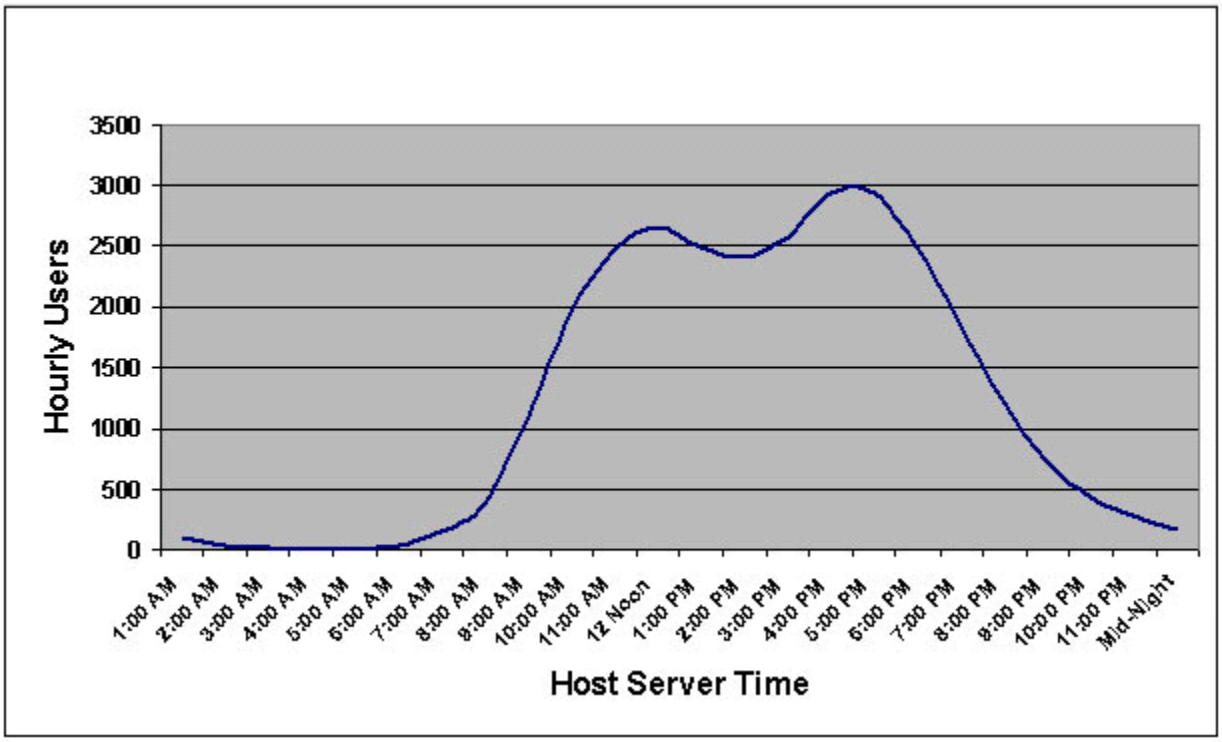
Creating a Suite to Match a Model (using Synchronization Points)

Referring to the model above, create a Rational TestStudio Suite that, when executed, will represent the model and will generate the target load for one hour. Assume that all scripts take approximately 15 min (900 seconds) to execute. Further assume that each solid horizontal line on the graph represents a single script.

To create empty scripts with the names on the horizontal lines, simply launch Rational Robot and select New > Script from the File menu and enter the appropriate name from the graph above. Click here to see my solution.

Determine Number of Concurrent Users

Given the information above and the chart below, determine the number of concurrent users required to simulate the peak user rate.



Notes to Advanced Users

Modeling Back-End Functions

You may have notice that I haven't discussed the need to account for back-end functions within the model. I don't want to spend a large amount of time on this topic. I do want to remind readers to consider these functions. If you are testing global or complex systems, that periodically run batch

process, replication, or backups while users are accessing the system, then it is imperative to execute tests while these back end activities are occurring. This is the only way to determine the associated performance degradation from a user's perspective. A short conversation with the system's administrator will normally reveal any back-end functions that may affect performance.

Server Perspective for Applications with User Persistence

I am willing to bet that some of you jumped straight to this section as soon as I mentioned it. I stated above that the server couldn't tell the difference between which user was conducting which activities. Of course the server can distinguish between users if the application tracks user sessions, whether that is through session ids, cookies, or some other method. A more accurate statement for me to have made would be that a server rarely cares which users perform what activities, nor does it care in which order by which user, as long as the pre-requisite activities are accomplished, from a performance perspective. Because I have scheduled an entire article to the handling of cookies and secure session ids, I don't want to give too much away here. For the time being, I will simply caution you to take extra care when modeling sites that have user persistence to ensure that this modeling method is detailed enough to provide accurate results. If the answer to that question is "No.", then the best I can do is encourage you to keep reading until "*Handling Secure Session Ids* " is published.

Summary

This article completes our discussion on modeling groups and users. Over the last three articles, we have seen how to model tests to simulate real users and user communities and discussed how important that is to obtaining valuable test results. We have also discussed some of the commonly held beliefs about user modeling and why they do not represent actual users. Next month we begin a three month exploration of what measurements to collect and how to collect them using Rational TestStudio.

Acknowledgments

- The original version of this article was written on commission for IBM Rational and can be found on the IBM DeveloperWorks web site

About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of

performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.