Bernie Velivis on Performance Testing Strategies:

PERFORMANCE TESTING STRATEGIES

The terminology used to discuss Performance testing in technical publications and support forums can be ambiguous or inconsistent. Hopefully this article will help participants in the OpenSTA user support forum by providing a common frame of reference for discussing tools, testing, and test results. It may also be helpful to those new to performance testing.

CAPACITY TESTING

If your goal is to determine the CAPACITY of the system under test, start by creating a "realistic" workload consisting of a mix of the most popular transactions plus those deemed critical or known to cause problems even when executed infrequently. Pick a manageable set of transactions to emulate (considering time, budget, and goals), determine the probability of executing each transaction, the work rate for the emulated users, and the "success criteria for performance metrics (i.e. response time limits, concurrent users, and throughput).

One way to implement this approach is to create a master script, assign it to each VU, and have it generate random numbers and then call other scripts which model the individual workload transactions based on a table of probabilities. The scripts should be modeled with think times consistent with the way your users interact with the system. This varies greatly from one application to another and unless you are mining log files from an application already in use, this is a somewhat subjective process. The best advice I can give in defining the workload is to get input from people who know how the application is (or will be) used, make conservative assumptions (but no so much so that the sum of all your conservative decisions is pathological), and balance the scope of the workload vs. time to complete the project. Another important consideration is the data demographics of the transactions and the size and contents of the database.

When it’s time to test, increase the number of emulated users and monitor how response times, server resource utilization (CPU, disk IO, network, and memory), and throughput (the rate of tasks completed system wide) vary with the increased load. You might construct a test that ramps up to a specific number of users, lets them run for a while, and then repeats as necessary. This way, you can observe the behavior of the system in various steady states under increasing load. Workloads containing transactions having a low probability of being executed and/or a disproportionately large impact on the performance of other transactions usually need to run longer to reach a steady state. If you can't get repeatable results, your steady state interval might be too small. As a rule of thumb I would suggest a minimum ramp up time equal to the duration of the longest running script and the steady state observation period at least twice as long as the ramp up period. I also tend to ignore response times and performance statistics gathered during the ramp up periods and focus instead on the data collected during the steady state periods.

That's a rough outline of one approach to capacity testing which in summary is an attempt to load up the system with VUs in a way that is indistinguishable from a "real users" in order to find the capacity limit. Pick the wrong workload however and you might miss something very important or end up solving problems that won't exist in the real world.

The end game here is to increase load until response times become excessive at which point you have found the system's capacity limit. This limit will be due to either a hardware or software bottleneck. If time and goals allow, analyze the performance metrics captured, do some tuning, improve code efficiency or concurrency, or add some hardware resources. Make one change at a time and repeat as necessary until you meet capacity goals, find the limits to the architecture, or run out of time (which happens more then most performance engineers would like).

SOAK TESTING

The same scripts created for capacity testing can also be used for SOAK TESTING where you load up the system close to its maximum capacity and let it run for hours, days, etc. This is a great way to spot stability problems that only occur after the system has been running a long time (memory leaks are a good example of things you might find).

FAILOVER TESTING

Get the system under test into a steady state and start failing components (servers, routers, etc) and observe how response times are effected during and after the failover and how long the system takes to transition back to steady state and you are on your way towards FAILOVER TESTING. (A gross simplification and again there is lots of good reading material out there on failover and high availability testing).

STRESS TESTING

If your goal is to determine where or how (not if) the system will fail under load, then you are doing STRESS TESTING. One way to do this is to comment out the think times and increase VUs until something (hopefully not your emulator!) breaks. This is one form of stress testing, a valuable aspect of performance testing, but not the same as capacity testing. How the VUs compare to "real users" may be irrelevant as you are trying to determine how the system behaves when pushed past its limits.

A report illustrating how these concepts were used to performance test a SOAP application using OpenSTA can be downloaded at:

http://iperformax.com/downloads/SamplePerformaxPerformanceReport.pdf

Bernie Velivis
Principle Consultant and President, Performax Inc
www.iPerformax.com