Load Testing Basics:

These are the basic ideas in setting up a load test

By: Bob Wescott

Summary

Load testing requires you to select transactions that are important to you and then synthetically generate them at a rate that tests your computing world's ability to handle them. Here we look at the basic ideas of load testing.

This is an excerpt (with modifications) from: ***The Every Computer Performance Book*** a short, practical, and occasionally funny book I wrote on doing computer performance work.

About Load Testing...

With good capacity planning you can project the measurements of the current workload into a busier future and see where trouble may lie. That is good and useful, but there is nothing quite as reassuring as watching your computing world smoothly handle a workload of synthetically generated transactions long before the real peak arrives.

The fundamental reason you go through all the trouble of a load test is to buy yourself time to fix things and to save money. You could just wait for the peak you are planning for to come naturally and hope for the best. If it works, then all is well. If not, then you have no time to fix anything, no time to test the fixes, and only the most expensive options to fix things remaining on the table. The company will have to throw money and hardware at the problem and endure significant risk with untested workarounds. If performance is bad, there will be additional costs as customers move to your competitors, and the call center spends a lot more time saying "we're really sorry" to the customers who bother to call.

Load testing can also increase your confidence in your capacity planning efforts and show where you have way too much of a given resource, both of which can save your company serious money as you may be able to narrow the safety margin.

Creating The Load

In real life, the users of your computing world generate the workload. In load testing, you define the workload. You need to know the individual transactions (e.g., deposit, withdraw) that make up the workload, the right ratio of these transactions (e.g. ten withdraw transactions for every deposit transaction), and figure out at what rate you want them delivered to emulate the peak load.

All of this will take some creative application performance metering and some discussions about which transactions to emulate. Let's tackle these problems one at a time, but first a word about abandoning your quest for perfection.

Good Enough Is Just Fine

Give up on the idea of perfection. There is no "perfect" in load testing, as the users are always changing their behavior and you will never emulate all the different transactions with all the possible user choices. Why?

Unlike capacity planning, where you can do your work by yourself with a spreadsheet and some metering data, load testing costs money, often requires the help of others, and can be disruptive. Also, load testing is usually done at an hour that is inconvenient for everyone involved. All of this tends to push back hard against perfection.

You are looking for "good enough" to get the job done. There are many choices you'll have to make and guesses you'll have to take, so how do you know the load test you've designed is good enough?

Test Validation

When you've designed and built your load test, run it at a normal everyday load and see if it works without errors and returns performance meter values that are similar to the ones you get on an average day. This is known as test validation.

For example, at noon on a pre-peak day your computing world is handling 50 TX/sec, and a key machine is around 20% busy. At midnight that machine is almost idle. You are planning for an upcoming peak that is four times (4X) the noon peak. If your load test reasonably emulates the real user load, then when you run your load test at midnight, sending in 50 TX/sec, the key machine should show 20% busy, and the other key meters in your computing world should also resemble their noon time values. If half your computing world is idle under this test, clearly you've got more work to do. If the numbers are close enough, then your test is good to go. But, what's close enough?

Begin with the peak in mind. The peak you are planning for is 4X the normal noon load of 20% busy. Any differences in the meters between the real user load and the midnight load test will be 4x greater at the peak, and that means little unimportant differences can become big significant differences. Let's look at the numbers on three key meters in the table below.

	Real User Load Measured At Noon	**Midnight Load Test**
Meter X	20%	19%
Meter Y	30%	20%
Meter Z	10%	8%

Meter X matches up nicely. If we were emulating the real user load perfectly we'd expect meter Y and meter Z to match up nicely as well, but they don't.

Meter Y is much busier at noon than during the midnight load test. Here it makes a difference, as we expect the peak to be 4X the measured day. So just using basic capacity planning two things are clear:

1. The resource watched over by Meter Y will bottleneck at peak as: 30% * 4 = 120%
2. This is a difference that makes a difference, as Meter Y during the load test only showed a utilization of 20% busy and that works out to 20% * 4 = 80% at peak. Meter Y tells us we have more work to do on this load test.

Meter Z is a little off between the noon and the midnight numbers, but this is a difference that you could live with because, even when you scale up the larger sample (10% * 4 = 40%), it's clear this is not going to be a bottleneck.

You can also do this checking with any meter that counts things of importance to you like packets, IO's, thingamajigs, whatever. Once the results are close enough, you can trust that your load test will do a good job pushing your computing world as hard as the users will at peak. Now, let's design a load test.

Selecting Transactions To Emulate

Your computing world handles many different types of transactions, but the bulk of the workload, and/or the bulk of the revenue, comes from just a few of them. It is also the case that many transactions have important differences for users but are computationally identical for your computing world – the bits flow through the same processes and consume the same resources.

Start building a list of transactions to emulate. First add the ones that make up the bulk of your workload. Then add any transactions that bring serious money into the company even if they are not all that numerous.

"To a corporation, nothing is more important than money. Follow the money."
– Bob's Seventh Rule of Performance Work

Then add any transactions that have recently caused you trouble and are thus politically sensitive at this time. Now look that list over and, if it makes things simpler, you can combine transactions that are computationally similar into a generic transaction – the buyX, buyY and buyZ transactions get grouped together into the generic buyStuff transaction. This will typically leave you with a short list of transactions.

Scripting Transactions
Users are not identical robots typing the same things over and over at machine-like speeds.

Users are unique, they pause to consider and to choose, and they do different things. There are constraints (logical, legal, and practical) as to what your users can do. You can't login simultaneously from two different cities, you can't withdraw with a zero balance, and you can't put a trillion things in your shopping cart. Whatever generates the load for your load test has to be able to handle that. Specifically look for load generation tools that have a way to:

- Record a script of actions to follow for each type of transaction you emulate
- React to variations during the transaction, such as security questions
- Notice if the transaction worked or failed ("Deposit OK" vs. "Database error…"
- Add variability to that script by doing different things on each visit
- Authenticate themselves to your security software
- Build think time between steps of a transaction to emulate the behavior of real users as they pause and consider

Scripting transactions takes work. First you get it to work once, then you add variability into it (different users doing different things), and then you find security or reasonableness checks you have to either deal with or work around. For multi-step transactions (e.g., login, shop, buy), you need to test at each step to see if it was successful and to "report & abort" the transaction if it was not.

Typically, you work on one transaction at a time, testing it over and over until you are satisfied. Then you create and test the next transaction. When you've got all the transactions in your load test working and tested individually, then you test them together, typically at low-power, to convince yourself that they all work together smoothly. Look for the results you see in the meters to match up nicely with the meters you get from the normal load generated by real users.

Generating The Load
Something has to generate the load for your load test. Depending on your situation you might build it yourself, buy it, or have some company generate the load for you, typically through the Internet. Here are some things to look for when evaluating your options.

Location

Where you generate the load matters. The load should flow though as much of your computing world as it would normally. Any part of your computing world that you do not test is, by definition, untested. That untested part is likely to keep you up at night worrying and surprise you, in an unpleasant way, during the peak with its shocking lack of throughput.

If you are doing a stand-alone load test of a small subsystem, then the generated load should come from outside the tested computer(s). Why? First, it takes resources to generate load, and you'd like a clean set of performance data from the tested system(s). Also, if you generate the load on the tested system, then you are not testing the network connections though which the real load will have to flow.

If you are doing an end-user load test, then the load should be generated outside your company and from the locations where your users live. As you saw in chapter four, distance matters on the Internet.

Ease Of Use

The sales pitch for the load test tool will tend to focus on the beauty and the flexibility of how it displays results. That's all good, but you'll spend a lot more time creating and debugging the load test than you will spend running and evaluating it. When selecting a load generation tool carefully note:

- The ease with which you can create new transactions and modify existing ones.
- The quality and clarity of diagnostic info you get back when transactions are failing.
- How easily and rapidly the tool can schedule, stop, and restart tests. Load testing is a team sport. Making people wait for you, and the load testing tool, is never fun.
- How close to real time do you get the results for transactions started and completed, transaction response time and failure rate. You want the bad news as soon as possible, so you can stop, fix, and restart the test.

Money

Generating load costs money. More load, more money. Budget for the testing you'll have to do before the big load test, and plan to work though a couple of failures where you have to stop the test, fix something, and restart it.