Performance Testing

Monday, October 08, 2007

Search blog ...
Labels
Archive
Feed
Subscribe by Email
Follow
Follow @googletesting

Posted by Goranka Bjedov, Senior Test Engineer

This post is my best shot at explaining what I do, why I do it, and why I think it is the right thing to do. Performance testing is a category of testing that seems to evoke strong feelings in people: feelings of fear (Oh, my God, I have no idea what to do because performance testing is so hard!), feelings of inadequacy (We bought this tool that does every aspect of performance testing, we paid so much for it, and we are not getting anything done!), feelings of confusion (So, what the heck am I supposed to be doing again?), and I don't think this is necessary.

Think of performance testing as another tool in your testing arsenal - something you will do when you need to. It explores several system qualities, that can be simplified to:

- Speed - does the system respond quickly enough
- Capacity - is the infrastructure sized adequately
- Scalability - can the system grow to handle future volumes
- Stability - does the system behave correctly under load

So, I do performance testing of a service when risk analysis indicates that failing in any of the above categories would be more costly to the company than performing the tests. (Which, if your name is Google and you care about your brand, happens with any service you launch.) Note that I am talking about services - I work almost exclusively with servers and spend no time worrying about client-side rendering/processing issues. While those are becoming increasingly more important, and have always been more complex than my work, I consider those to be a part of functionality tests, and they are designed, created and executed by functional testing teams.

Another interesting thing about performance testing is that you will never be able to be 100% "right" or 100% "done. Accept it, deal with it, and move on. Any system in existence today will depend on thousands of different parameters, and if I spent the time analyzing each one of them, understanding the relationships between each two or each three, graphing their impact curves, trying to non-dimensionalize them, I would still be testing my first service two years later. The thought of doing anything less filled me with horror (They cannot seriously expect me to provide meaningful performance results in less than a year, can they?) but I have since learned that I can provide at least 90% of meaningful information to my customers by applying only 10% of my total effort and time. And, 90% is more than enough for vast majority of problems.

So, here is what I really do - I create benchmarks. If I am lucky and have fantastic information about current usage patterns of a particular product (which I usually do), I will make sure this benchmark covers most operations that are top resource hogs (either per single use or cumulative). I'll run this benchmark with different loads (number of virtual users) against a loosely controlled system (it would be nice to have 100 machines all to myself for every service we have, which I can use once a day or once a week, but that would be expensive and unrealistic) and investigate its behavior. Which transactions are taking the most time? Which transactions seem to get progressively worse with increasing load? Which transactions seem unstable (I cannot explain their behavior)? I call this exploratory performance testing, and I'll repeat my tests until I am convinced I am observing real system behavior. While I am doing this, I make sure I am not getting biased by investigating the code. If I have questions, I ask programmers, but I know they are biased, and I will avoid getting biased myself!

Once I have my graphs (think, interesting transaction latencies and throughput vs. load here) I meet with the development team and discuss the findings. Usually, there is one or two things they know and have been working on, and a few more they were unaware of. Sometimes, they look over my benchmark and suggest changes (could you make the ratio 80:20, and not 50:50?) After this meeting, we create our final benchmark, I modify the performance testing scripts, and now this benchmark will run as often as possible, but hopefully at least once a night. And, here is the biggest value of this effort: if there is a code change that has impacted performance in an unacceptable way, you will find out about it the next day. Not a week or a month later (How many of us remember what we did in the last month? So, why expect our developers to do so?)

Here is why I think this is the right thing to do: I have seen more bad code developed as a result of premature performance optimizations - before the team even thought they had a problem! Please don't do that. Develop your service in a clean, maintainable and extensible manner. Let me test it, and keep regression testing it. If we find we have a problem in a particular area, we can then address that problem easily - because our code is not obfuscated with performance optimization that have improved code paths that execute once a month by 5%.

I can usually do this in two - four weeks depending on the complexity of the project. Occasionally, we will find an issue that cannot be explained or understood with performance tests. At that point in time, we look under the hood. This is where performance profiling and performance modeling come in. And, both of those are considerably more complex than performance testing. Both great tools, but should be used only when the easy tool fails.

Tools, tools, tools... So, what do we use? I gave a presentation at Google Test Automation Conference in London on exactly this topic. I use open source tools. I discuss the reasons why in the presentation. In general, even if you have decided to go one of the other two routes (vendor tools or develop your own) check out what is available. You may find out that you will get a lot of information about your service using JMeter and spending some time playing around with it. Sure, you can also spend $500K and get similar information or you can spend two years developing "the next best performance testing tool ever," but before you are certain free is not good enough, why would you want to?

Final word: monitor your services during performance tests. If you do not have service related monitoring developed and set up to be used during live operations, you do not need performance testing. If the risks of your service failing are not important enough that you would want to know about it *before* it happens, then you should not be wasting time or money on performance testing. I am incredibly lucky in this area - Google infrastructure is developed by a bunch of people who, if they had a meeting where the topic would be "How to make Goranka's life easy?", could not have done better. I love them - they make my job trivial. At a minimum, I monitor CPU, memory and I/O usage. I cannot see a case when you would want to do less, but you may want to do a lot more on occasion.

Labels: Goranka Bjedov