Load Testing: Which Tool to Choose?

Alexander Podelko
Oracle

The topic of load testing tool selection always triggers a lot of discussion. Unfortunately, it most often turns into religious wars than objective technical analysis. There are many aspects differentiating load testing tools and it is probably better to evaluate tools on each aspect separately. The paper discusses some aspects of load testing tools and lists some considerations impacting the selection process. The list is far from comprehensive and is provided rather to illustrate the existing issues and show how the selection process for specific needs may be approached.

The topic of load testing tool selection always triggers a lot of discussion. Unfortunately, it most often turns into religious wars than objective technical analysis – partially because the outcome will depend upon your specific needs, partially because few people have time to really investigate different tools, partially because vendors are deeply involved and have their own agenda.

Let us first define load testing as the terminology is rather vague here [STIR02, MOLY09, PERF07]. The term is used here for everything requiring application of multi-user synthetic load. Many different names may be used for such multi-user testing, such as performance, concurrency, stress, scalability, endurance, longevity, soak, stability, or reliability testing. There are different (and sometimes conflicting) definitions of these terms. However, they describe testing from somewhat different points of view, meaning they are not mutually exclusive.

While each kind of performance testing may have different goals and test designs, in most cases they use the same approach: applying multi-user synthetic workload to the system. The term "load testing" is used here to better contrasts multi-user testing with other performance engineering methods such as single-user performance testing without the need for a load testing tool.

A typical load testing process is shown in Figure 1.

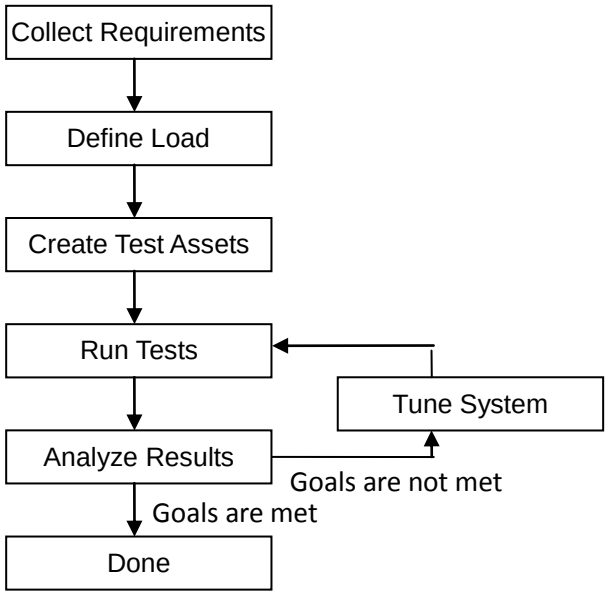

Figure 1 – Load Testing Process

Most readers have probably seen something like this – but here two different steps are shown explicitly: "define load" and "create test assets." The "define load" step (sometimes referred to as workload

characterization or workload modeling) is a logical description of the load to be applied (e.g. users log in, navigate to a random item in the catalog, add it to the shopping cart, pay, and logout with an average of ten seconds of think time between actions). Whilst the "create test assets" step is the implementation of this workload and the conversion of the logical description into something that will physically create that load during the "run tests" step. Manual testing may still be an option in a few cases (when load is low and repeatability is not needed) – then it can be just the description given to each tester. But in all other load testing cases, it should be a program or a script.

As far as a tool is needed for load testing, the subject of selecting one becomes very important. Moreover, there are attempts (not often without vendor involvement) to present a load test tool as a complete solution to load testing, making the question of selection bigger than it should be. Yes, a good tool in load testing is very important – but it is still just a tool. A carpenter needs good tools, but tools do not make his job; he still needs skills and experience to use them. The same is true in load testing. Let's look at different aspects of load testing tools, keeping in mind that they are only tools to help you do your job – they won't do it for you.

Classifying and evaluating load testing tools is not easy, as they include different sets of functionality often crossing borders of whatever criteria are used. In most cases, any classification is either an oversimplification (which in some cases still may be useful) or a marketing trick to highlight advantages of specific tools. There are many aspects differentiating load testing tools and it is probably better to evaluate tools on each aspect separately.

There are probably more than a hundred different commercial and open source tools – and very few attempts to compare them, especially the more recent ones – and a lot changes in the area are happening right now. For commercial tools, Gartner Magic Quadrant for Integrated Software Quality Suites [GART13] may be worth mentioning. The report covers Integrated Software Quality Suites, which includes other products for functional testing and test management – but it adds a special note for load testing products.

Gartner names the following companies as leaders (with their load testing products in parenthesis): HP (LoadRunner), IBM (Rational Performance Testing), Microsoft (Visual Studio Web Performance and Load Tests), Oracle (Application Testing Suite), SOASTA (CloudTest), and Borland (SilkPerformer) – and mentions separately Neotys (NeoLoad) and BlazeMeter, not included in the Quadrant because they specialize in performance testing.

The Gartner report doesn't discuss open source tools. There are many open source tools [OPEN], but only a few are somewhat mature and sophisticated. JMeter is probably the most popular tool, Gatling is getting popularity recently. Some open source tools, popular some time ago, such as OpenSTA and Grinder, don't look to be in active development anymore. Most other open source tools are either very simple, or specialized, or both.

Technical Aspects

This section will discuss some aspects of load testing tools and list some considerations impacting the selection process. The list is far from comprehensive and is provided rather to illustrate the existing issues and show how the selection process for specific needs may be approached. A few tools are mentioned to illustrate certain aspects, but as there is no intention to provide a deep analysis of all available tools (there are probably at least a hundred such tools around), there is no implication that the mentioned tools are necessarily better than others. Let's consider technical aspects first.

Load Generation

There are three main approaches to workload generation [PODE12] and every tool may be evaluated on which of them it supports and how.

Protocol-level recording/playback

This is the mainstream approach of load testing: recording communication between two tiers of the system and playing back the automatically created script (usually, of course, after proper correlation and parameterization). As far as no client-side activities are involved, it allows the simulation of a large number of users. Such tool can only be used if it supports the specific protocol used for communication between two tiers of the system.

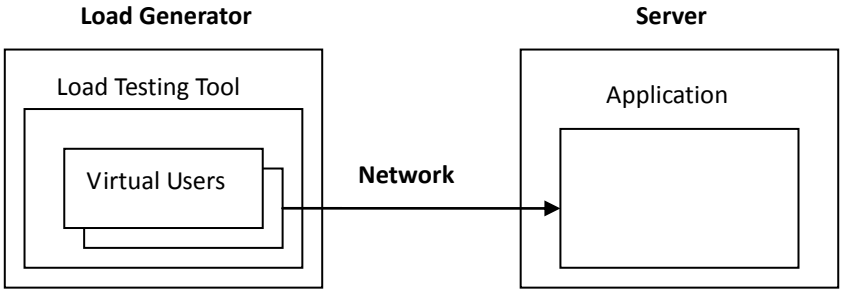

Fig.2 Record and playback approach, protocol level

With quick internet growth and the popularity of browser-based clients, most products support only HTTP or a few select web-related protocols. To the author's knowledge, only HP LoadRunner and Borland SilkPerformer try to keep up with support for all popular protocols (other products claiming support of different protocols usually use UI-level recording/playback, described below). Therefore, if you need to record a special protocol, you will probably end up looking at these two tools (unless you find a special niche tool supporting your specific protocol). This somewhat explains the popularity of LoadRunner at large corporations using nearly all possible protocols. The level of support of specific protocols differs significantly, too. Some HTTP-based protocols are extremely difficult to correlate if there is no built-in support, so it is recommended to look for that kind of specific support if such technologies are used. For example, Oracle Application Testing Suite may have better support of Oracle technologies (especially new one as Oracle Application Development Framework, ADF).

Quite often the whole area of load testing is reduced to pre-production testing using protocol-level recording/playback [BUKSH12]. While it was (and still is) the mainstream approach to testing applications, it is definitely just one type of load testing using only one type of load generation – such equivalency is a serious conceptual mistake, dwarfing load testing and undermining performance engineering in general [SMITH02].

UI-level recording/playback

This option has been available for a long time, but it is much more viable now. For example, it was possible to use Mercury/HP WinRunner or QuickTest Professional (QTP) scripts in load tests, but a separate machine was needed for each virtual user (or at least a separate terminal session). This drastically limited the load level that could be achieved. Other known options were, for example, Citrix and Remote Desktop Protocol (RDP) protocols in LoadRunner – which always were the last resort when nothing else was working, but were notoriously tricky to play back [PERF].

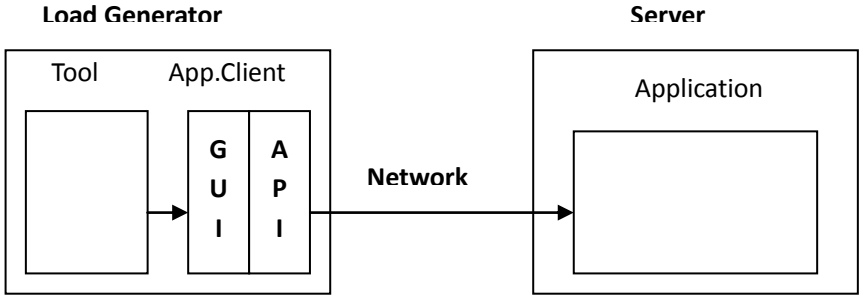

Fig.3 Record and playback approach, GUI users

New UI-level tools for browsers, such as Selenium, have extended possibilities of the UI-level approach, allowing the running of multiple browsers per machine (limiting scalability only to the resources available to run browsers). Moreover, UI-less browsers, such as HtmlUnit or PhantomJS, require significantly fewer resources than real browsers.

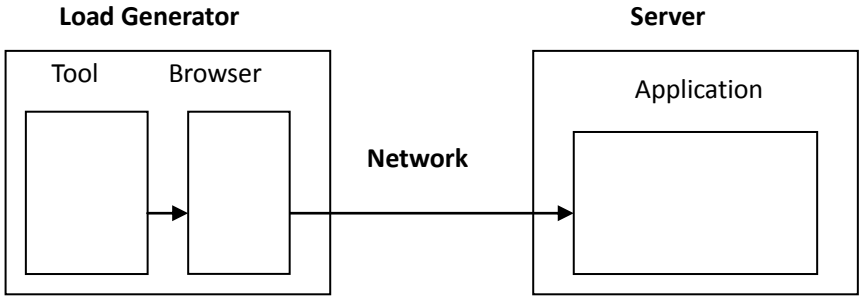

Fig.4 Record and playback approach, browser users

Today there are multiple tools supporting this approach, such as Appvance, which directly harnesses Selenium and HtmlUnit for load testing; or LoadRunner TruClient protocol and SOASTA CloudTest, which use proprietary solutions to achieve low-overhead playback. Nevertheless, questions of supported technologies, scalability, and timing accuracy remain largely undocumented, so the approach requires evaluation in every specific case.

Programming

There are cases when recording can't be used at all, or even when it can, it is only with great difficulty. In such cases, API calls from the script may be an option. Sometimes it is the only option for component performance testing. Other variations of this approach are web services scripting or use of unit testing scripts for load testing. And, of course, there is a need to sequence and parameterize your API calls to represent a meaningful workload. The script is created by whatever mean is appropriate and then either a test harness is created to execute it or a load testing tool is used to execute scripts, coordinate their executions, and report and analyze results.

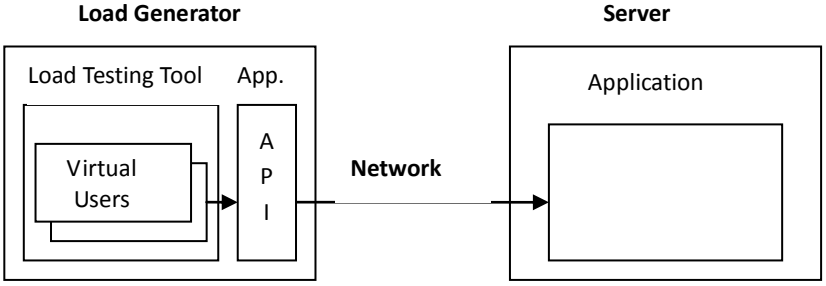

Fig 5. Programming API using a Load Testing Tool.

To do this, the tool should have the ability to add code to (or invoke code from) your script. And, of course, if the tool's language is different from the language of your API, it would be a need to figure out a way to plumb them. Tools, using standard languages such as C (e.g. LoadRunner) or Java (e.g. Oracle Application Testing Suite) may have an advantage here. However, it is understanding all the details of the communication between client and server to use right sequences of API calls that is often the challenge.

Special cases

There are special cases which should be evaluated separately, even if they use the same listed above generic approaches. The most prominent special case is mobile technologies. While the existing approaches remain, basically, the same – there are many details on how these approaches get implemented that need special attention. The level of support for mobile technologies differs drastically – from very basic ability to record HTTP traffic from a mobile device and play it back against the server up to end-to-end testing for native mobile applications and providing a "device cloud".

Test Management

One of the main advantages of load testing tools is built-in ability to manage test execution and collect test data. Starting from coordination of test execution from multiple load generators and collecting all information. In more sophisticated cases coordination between virtual users is needed, such as synchronization points, data exchange during execution, or sophisticated scheduling.

Accuracy of user simulations is very important too. The test results may vary depending on specific browser and browser's settings, protocol used, network connection details, etc. Details of implementation of browser's cache and threading may have a major impact. Sophisticated tools allow creation of very complicated test scenarios and configure each group of users in the way required.

Other important aspects are, for example, network simulations (allowing to simulate different network conditions for different groups of users) or IP spoofing (allowing every virtual user pretends to have its own IP address, which may be important for proper work of load balancers), allowing to create realistic load.

Deployment Model

There were many discussions about different deployment models: lab vs. cloud (some parts or everything there) vs. service. There are some advantages and disadvantage of each model. Depending on specific goals and the systems to test, one deployment model may be preferred over another.

For example, to see the effect of performance improvement (performance optimization), using an isolated lab environment may be a better option. To load test the whole production environment end-to-end without concerns regarding small variations, testing from the cloud or a service may be more appropriate. To create a production-like test environment without going bankrupt, moving everything to the cloud may be a solution.

But for comprehensive performance testing, there is a need for both lab testing (with reproducible results for performance optimization) and distributed realistic outside testing (to check real-life issues you can't simulate in the lab). Doing both can be expensive and makes sense only when performance is critical and the system is global – but even it is not so yet, it may end up there eventually. If there is a chance of facing these issues, it would be better to have a tool supporting different deployment models.

Whether lab or cloud, another important question is what kind of software/hardware/cloud the tool requires/supports. Many tools use low-level system functionality, so minor differences in OS or browser version may make a big difference – and it would be a very unpleasant surprise if your platform of choice or your corporate browser standard is not supported by the tool. It also makes it important that the tool should be under active development / support as far as environments are changing all the time and somebody should make sure that the tool would be able to support it.

Scaling

When you have only a few users to simulate, it is usually not a problem. The more users you need to simulate, the more important the right tool becomes. Tools differ drastically on how many resources they need per simulated user and how well they handle large volumes of information. This may differ significantly even for the same tool, depending on the protocol used and the specifics of your script. As soon as thousands of users is reached, it may become a major problem. For a very large number of users, some automation – like automatic creation of a specified number of load generators across several clouds in SOASTA CloudTest – may be very handy.

Environment Monitoring and Result Analysis

Environment monitoring and result analysis are two very important sets of functionality. They are grouped together here for one single reason: while theoretically it is possible to do them both using separate tools, it significantly degrades productivity and may require building some plumbing infrastructure. So while these two areas may look optional, integrated and powerful monitoring and result analysis are both very important. The more complex system and tests, the more important they become. A possibility to analyze monitoring results and test results together helps a lot.

Talking about integrated monitoring, it is important to understand what kind of information is available and what mechanisms are behind it. While for Windows there is usually Performance Monitor behind the scene, for (L)UNIXes it may differ a lot. Getting information from application (via, for example, JMX) and database servers is important. Many tools recently announced integration with Application Performance Management / Monitoring (APM) tools, such as AppDynamics, New Relics, or Compuware Dynatrace. If using such tools is an option, it definitely opens new opportunities to look inside of what is going on inside the system under load. One thing to keep in mind is that older APM tools and profilers may be not appropriate to use under load due to high overheads they introduce.

Automation Support

Integration support becomes increasingly important as everyone talks about continuous integration (CI) and agile methodologies. Until recently, while there were some vendors claiming their load testing tools better fit agile processes, it usually meant that the tool is a little easier to handle (and, unfortunately, often just because there is not much functionality).

What makes agile projects really different is their need to run large number of tests repeatedly – resulting in the need for tools supporting performance testing automation. Unfortunately, even if a tool has something that may be used for automation, like starting by a command line with parameters, it may be difficult to discover. If continuous integration is on the horizon (to whatever degree), it is important to understand what the tool provides to support CI.

The situation started to change recently as agile support became the main theme in load testing tools [LOAD14]. Several tools recently announced integration with Continuous Integration Servers (such as Jenkins or Hudson). While initial integration may be minimal, it is definitively an important step toward real automation support.

While already mentioned above, cloud integration and support of new technologies are important for integration too. Cloud integration, including automated deployment to public clouds (almost all major load testing tools) and private cloud automation (Oracle Testing as a Service - TaaS), simplifies deployment automation. Support of newest technologies used (such as WebSocket or SPDY by Neoload or ADF in Oracle Load Testing) eliminates time-consuming and error-prone manual scripting and streamlines automation.

Non-Technical Criteria

Of course, non-technical criteria are important too and should be thoroughly considered.

Cost

There are many commercial tools (with wide range of license costs and licensing rules) as well as free and open source tools [OPEN]. There are over a hundred of different tools. The cost is not necessarily a good indicator of tool's functionality and quality. Some free tools, such as JMeter, are mature enough and well-known. But many free and inexpensive tools are very limited in functionality.

An interesting industry trend is getting some choices in between. After SOASTA introduced CloudTest Lite, free up to 100 users (but with some quirks like VM-based distribution), HP announced LoadRunner Community Edition and Neotys announced Neoload Free Edition, both full-functional and free up to 50 virtual users. There are also premium services based on open source product, like provided by Blazemeter on the top of JMeter, bringing open source products to enterprise level and, in a way, reinvigorating them. The combination that in a way boils down to the similar model – free product for small-scale and simple tests, premium service for large-scale tests – but without limitation of the number virtual users for the free option.

It comes in time for the latest industry trends. With agile development and continuous integration (CI) / delivery / deployment the old approach when only a dedicated performance tester works with the tool is becoming history. There is a need to have an opportunity for developers and other testers to use tools, invoke tools automatically as part of CI, etc. It probably won't be large-scale tests anyway – 50 users should be enough for most of such tests. So it gets covered for free – with only a need to pay for large-scale tests.

Information, Support, and Skills Availability

Recording and load generation have a lot of background sophistication, and issues could happen in any area. Availability of good support or at least an active user community may significantly improve productivity. While some information can be found on the Internet for the leading tools, not much is available for other tools.

For open source products it is good to check if communities are still active. For example, even for such products as OpenSTA and Grinder, which were relatively popular at some point, communities don't appear to be active anymore. Not to mention more obscure products, many of which never had an active community around them. The same is true for commercial products – not all, even relatively popular at some point or represented by a well-known company, are progressing at the same speed. And, as soon as new versions of

browsers and operating systems appear regularly, there is a need for tools at least to be upgraded to support them – not to mention all other changes needed to keep up to date with modern industry trends.

Considering the large number of tools and the relatively small number of people working in this field, the labor market supports only the most popular tools. Even for second-tier tools, there are few people around and few positions available. So by not choosing the market leaders, do not rely on finding people with this specific tool experience. Of course, an experienced performance engineer will learn any tool – but it may take some time until productivity reaches the expected level. And, again, materials and training may be available for leading tools – but may be very scarce for others (probably mostly limited to vendor / project site).

Just to illustrate the point, table 1 below includes the numbers of found documents in Google and US positions at Monster.com for a few products with more distinctive names (on 8/26/2014). Many other products, such as IBM Rational Performance Tester, Oracle Application Tester Suite, Grinder or Gatling are not included in the table as search returns a lot of un-related results.

Name	Number of found documents by Google	Number of found US positions at Monster.com
[HP] LoadRunner	894,000	170
[Apache] JMeter	688,000	90
[Borland] SilkPerformer	138,000	12
[Neotys] NeoLoad	87,100	3
[SOASTA] CloudTest	51,500	2
[CustomerCentrix] LoadStorm	18,000	-

Table 1. The numbers of found documents by Google and US positions at Monster.com for a few products with more distinctive names (on 8/26/2014).

Summary

This is, of course, not a comprehensive list of possible aspects of evaluation – rather a few starting points. Unfortunately, in most cases you can't just rank tools on a simple better/worse scale. It may be the case that a simple tool will work quite well in a particular situation. If your business is built around a single website, doesn't use sophisticated technologies, and load is not extremely high, then almost every tool will work for you. The further you are from this state, the more challenging it is to select the right tool. It may even be that you will need several tools.

Two main takeaways are:

- While all load testing tools look similar, they are actually quite different. And unfortunately, generic descriptions (for example, from the vendor website) are usually useless in understanding the differences.

- Your situation *is* different. A tool may be very good in one situation and completely useless in another. The value of the tool is not absolute; rather it is relative to *your* situation.

And while you may use the aforementioned aspects to evaluate tools, it is not guaranteed that a specific tool will work with your specific product (unless it uses a well-known and straightforward technology). That actually means that if you have a few systems to test, you need to evaluate the tools you consider using your systems and see if the tools can handle them. If you have many, choosing a tool supporting multiple load generation options is probably a good idea (and, if possible, evaluating it with at least the most important systems prior to implementation).

References

[BUKSH12] J. Buksh. Performance Testing is hitting the wall. (2012).
http://www.perftesting.co.uk/performance-testing-is-hitting-the-wall/2012/04/11/

[GART13] Magic Quadrant for Integrated Software Quality Suites. Gartner (2013)
http://www.gartner.com/technology/reprints.do?id=1-1H3S92L&ct=130712&st=sg

[LOAD14] Load Testing at the Speed of Agile. A Neotys White Paper. (2014).
http://www.neotys.com/documents/whitepapers/whitepaper_agile_load_testing_en.pdf

[MOLY09] I. Molyneaux. The Art of Application Performance Testing. O'Reilly (2009).

[OPEN] Open Source Performance Testing Tools.
http://www.opensourcetesting.org/performance.php

[PERF] Performance Testing Citrix Applications Using LoadRunner: Citrix Virtual User Best Practices, Northway white paper. http://northwaysolutions.com/our-work/downloads

[PERF07] Performance Testing Guidance for Web Applications. (2007).
http://perftestingguide.codeplex.com/

[PODE12] A.Podelko. Load Testing: See a Bigger Picture. CMG (2012).

[SEGUE05] Choosing a Load Testing Strategy, Segue white paper (2005).
http://www.iiquality.com/articles/load_testing.pdf

[SMITH02] C.U. Smith, L.G.Williams, “Performance Solutions”, Addison-Wesley (2002).

[STIR02] S.Stirling, Load Testing Terminology, Quality Techniques Newsletter (2002).
http://blogs.rediff.com/sumitjaitly/2007/06/04/load-test-article-by-scott-stirling/