Load Testing: See a Bigger Picture

Alexander Podelko
Oracle

Load testing is an important part of the performance engineering process. It remains the main way to ensure appropriate performance and reliability in production. Still it is important to see a bigger picture beyond stereotypical last-moment load testing. There are different ways to create load; a single approach may not work in all situations. Many tools allow you to use different ways of recording/playback and programming. This paper discusses pros and cons of each approach, when it can be used and what tool's features we need to support it.

Load Testing as a part of the Performance Engineering Process

Recent trends of agile development, continuous deployment, DevOps, web operations somewhat question importance of load testing. Some (not many) openly saying that they don't need load testing, some still paying lip service to it – but just never get to it. In more traditional corporate world we still see performance testing groups and important systems usually get load tested before deployment.

Let's first define load testing as far as terminology is rather vague here. The term is used here for everything that requires applying multi-user synthetic load. Many different terms are used for such kind of multi-user testing, such as performance, concurrency, stress, scalability, endurance, longevity, soak, stability, reliability, etc. There are different (and sometimes conflicting) definitions of these terms (for example, [STIR02], [MOLY09]). Mostly these terms describe testing from somewhat different points of view, so 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 further in this paper because, by author's opinion, it better contrasts multi-user testing with other performance engineering methods including single-user performance testing. Everything mentioned here applies to performance, stress, concurrency, scalability, and other kinds of testing as far as the system is tested by applying multi-user load.

If we define load testing in this way, it becomes evident that it is much wider the stereotypical waterfall-like last-moment record-and-replay load testing that we often see in large corporations. Unfortunately load testing often became associated with that stereotype that makes difficult to see a larger picture of load testing as an important and integral part of the performance engineering process.

Load testing is a way to mitigate load- and performance-related risks. There are other approaches and techniques that also alleviate some performance risks ([SMITH02], [MICR04]):

- Single-user performance engineering practices (profiling, tracking and optimization single-user performance, Web Performance Optimization).
- Software Performance Engineering (SPE – including modeling and performance patterns)
- Instrumentation/Application Performance Management providing insights in what is going on inside the production system.
- [Auto] scalable architecture.
- Continuous integration/deployment allowing quickly deploy and remove changes.

Still they don't replace load testing, but rather complement it. They definitely decrease performance risks comparing with situation when nothing was done about performance at all until the last moment before rolling out the system in production, but it still leaves risks of crashing and performance degradation under multi-user load. There is always a risk of crashing or performance issues under heavy load – and the best way to mitigate it is actually to test it. Even stellar performance in production and highly scalable architecture don't guarantee that it won't crash with a slightly higher load. So if the cost of it is high, you should do load testing (what exactly, when and how are other large topics; only one side of them – load generation – is discussed in this paper). Although, to

be exact, even proper load testing isn't a full guarantee (real-life workload always would be somewhat different from what you have tested), but it drastically decreases the risks.

Another important value of load testing is making sure that changes don't degrade multi-user performance. Unfortunately, better single-user performance doesn't guarantee better multi-user performance. In many cases it improves multi-user performance too, but definitely not always. And the more complex system, the more chances of exotic multi-user performance issues no one even thought of. And load testing is the way to ensure that you don't have such issues.

In particular, when you do performance optimization, you need a reproducible way to evaluate the impact of changes on multi-user performance. The impact on multi-user performance won't probably be proportional to what you see with single-user performance (even if it still would be somewhat correlated). Without multi-user testing the actual effect is difficult to quantify. It is also important for issues happening only in specific cases that are difficult to troubleshoot and verify in production – using load testing can significantly simplify the process.

Load Testing Process Overview

Load testing is emerging as an engineering discipline of its own, based on “classic” testing from one side, and system performance analysis from another side. A typical load testing process is shown in figure 1 (for variations see [BARB06], [MICR04]).

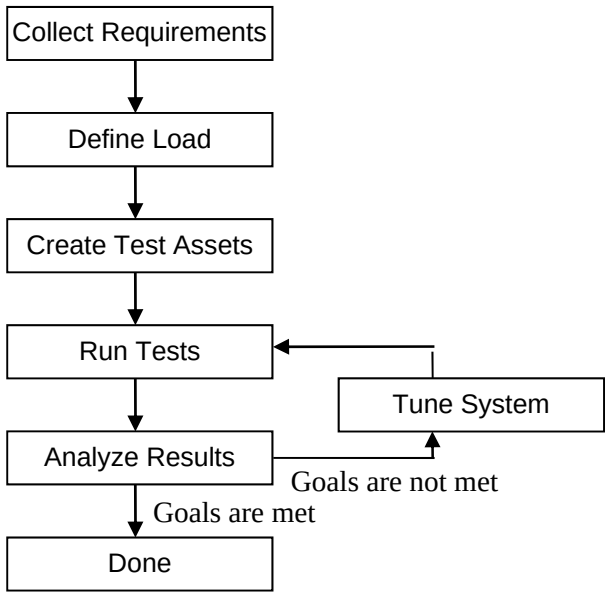

Fig.1 Load testing process

We explicitly define two different steps here: '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 we want to apply (like “that group of users login, navigate to a random item in the catalog, add it to the shopping cart, pay, and logout with average 10 seconds think time between actions”). The 'create test assets' step is the implementation of this workload, and conversion of the logical description into something that will physically create that load during the 'run tests' step. While for manual testing that can be just the description given to each tester, usually it is something else in load testing – a program or a script.

Quite often load testing goes hand-in-hand with tuning, diagnostics, and capacity planning. They are actually represented by the back loop on the fig.1: if we don't meet our goal, we need optimize the system to improve performance. Usually the load testing process implies tuning and modification of the system to achieve the goals.

Load testing is not a one-time procedure. It spans through the whole system development life cycle. It may start from technology or prototype scalability evaluation, continue through component / unit performance testing into system performance testing and follow up in production (to troubleshoot performance issues and test upgrades / load increases).

Before we can move forward from 'define load' to 'create test assets', we need to decide how we are going to generate that load. Load generation can be a simple technical step when you know how to do it for your system (compared with other non-trivial steps like collecting requirements, defining load, or analyzing results). Unfortunately, quite often it is a very challenging task for a new system, up to being impossible in the given time frame. It is important to understand all possible options; a single approach may not work in all situations. The main choices are to generate workload manually (really an option only if you test few users), use a load testing tool (software or hardware), or create a program to do it. Many tools allow using different ways of recording/playing back and programming. The paper considers different approaches to assist in choosing the best approach and tool for specific contexts. It is based on experience with business applications, so handling load in other kinds of environments may differ.

Record and Playback: Protocol Level

The mainstream approach of load testing (at least for business and Internet applications) is recording communication between two tiers of the system and playing back the automatically created script (usually, of course, after proper correlation and parameterization). Tools used for that are usually referred as “load testing tools” and users simulated by such tools are usually referred as “virtual users”. The real client-side software isn’t necessary to replay such scripts, so the number of simulated virtual users can be high; it is theoretically limited only by available hardware (each tool has specific hardware requirements depending on the type and complexness of scripts).

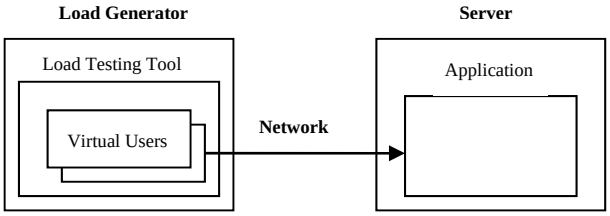

Fig.2 Record and playback approach, protocol level

Both recording and playback happen between the tiers, so the protocol used between the client and the server is extremely important. Other factors, such as what language was used to develop the system, what platform the server is deployed on, etc. are usually irrelevant for scripting (although they can give some hints about what protocol is used for communication).

The process is reasonably straightforward when you test a simple Web site or a simple Web application with a thin client. Even a beginner in load testing can quickly create a few scripts and run tests. That is one reason why the record and playback approach is so popular. However, there is a trap in that easiness: load testing really embraces much more. Load should be validated for correctness (if you don’t see errors in the load testing tool it doesn’t always mean that it works properly) and realism (using unrealistic scenarios is the easiest way to get misleading results). Moreover, load generation is only one step in load testing, there are many other important parts (like getting requirements and doing results analysis), as well as other related activities (like tuning or diagnostics).

Unfortunately, scripting can be challenging even for a Web application. Recording a script and making it work can be a serious research task, often including many try-and-fail iterations. A good load testing tool can help if it supports your protocol.

The protocol level record and playback approach has several serious limitations:

- It usually doesn’t work for testing components and services.
- Each particular load testing tool supports a limited number of technologies.

- Some technologies require very time-consuming correlation and parameterization and some may be not supported at all.
- The workload validity in case of sophisticated logic on the client side is not guaranteed.

These limitations are usually not a problem in the case of simple web applications using a browser as a client, but they become a serious problem when you need to test different protocols across the whole software lifecycle.

Each load testing tool supports a limited number of technologies (protocols). New or exotic technologies are not usually on the list. Vendors of load test tools add new supported protocols continually, but we often do not have time to wait for the specific protocol to be added – as soon as we get a new product we need to test it.

For example, back in 1999, we were not able to use recording for the SMB (Server Message Block) protocol, later succeeded by the Common Internet File System (CIFS) protocol, Microsoft DCOM (Distributed Component Object Model), or Java RMI (Remote Method Invocation). While some vendors claimed that their products support these protocols, it didn't work in all environments.

Later there were issues with Java applets and ActiveX controls, which used serialization, encoding, or even proprietary protocols.

Today we are getting a new generation of Rich Internet Applications (RIA) and new web protocols, bringing these old challenges of protocol level recording back – so some authors started to talk about the crisis of performance testing (for example, [BUKSH12]). Still these issues don't look more challenging that the issues we had 10-15 years ago – especially considering that many still use underlying standard web protocols, so we at least are able to record the communication.

Even if the protocol is supported, script recording and parameterization often are far from being straightforward and often required a good knowledge of system internals. The question of workload validation is also opened. Here is a simple example illustrating possible issues.

One function of the product we were testing was financial consolidations, which can take a long time. The client starts the operation on the server, then waits for it to finish, as a progress bar is shown on screen. When recorded, the script looks like (in LoadRunner pseudo-code):

```
web_custom_request("XMLDataGrid.asp_7",
"URL={URL}/Data/XMLDataGrid.asp?Action=EXECUTE&TaskID=1024&RowStart=1&ColStart=2&RowEnd=1&ColEnd=2&SelType=0&Format=JavaScript",
                LAST);

web_custom_request("XMLDataGrid.asp_8",
"URL={URL}/Data/XMLDataGrid.asp?Action=GETCONSOLSTATUS",
LAST);

web_custom_request("XMLDataGrid.asp_9",
"URL={URL}/Data/XMLDataGrid.asp?Action=GETCONSOLSTATUS",
LAST);

web_custom_request("XMLDataGrid.asp_9",
"URL={URL}/Data/XMLDataGrid.asp?Action=GETCONSOLSTATUS",
LAST);
```

Each request's action is defined by the *?Action=* part. The number of *GETCONSOLSTATUS* requests recorded depends on the processing time. In the example above, the request was recorded three times; it means the consolidation was done by the moment the third *GETCONSOLSTATUS* request was sent to the server. If you play back this script, it will work this way: the script submits the consolidation in the *EXECUTE* request and then calls *GETCONSOLSTATUS* three times. If we have a timer around these requests, the response time will be almost instantaneous, while in reality the consolidation may take many minutes or even hours. If we have several iterations in the script, we will submit several consolidations, which continue to work in the background, competing for the same data, while we report sub-second response times.

Consolidation scripts require creating an explicit loop around GETCONSOLSTATUS to catch the end of consolidation:

```
web_custom_request("XMLDataGrid.asp_7",
"URL={URL}/Data/XMLDataGrid.asp?Action=EXECUTE&TaskID=1024&RowStart=1&ColStart=2&RowEnd=1&ColEnd=2&SelType=0&Format=JavaScript",
                LAST);

do {

        sleep(3000);

        web_reg_find("Text=1","SaveCount=abc_count",LAST);

        web_custom_request("XMLDataGrid.asp_8",
        "URL={URL}/Data/XMLDataGrid.asp?Action=GETCONSOLSTATUS",
        LAST);

        } while (strcmp(lr_eval_string("{abc_count}"),"1")==0);
```

Here the loop simulates the internal logic of the system, sending *GETCONSOLSTATUS* requests every three seconds until the consolidation is complete. Without such a loop, the script just checks the status and finishes the iteration while the consolidation continues for a long time after that.

So, it is possible that the record and playback approach won't work in your environment, or that using the approach will be too time-consuming and inflexible (as it happened many times for us). When such problems are encountered, it is a good time to check other alternatives and add them to your arsenal.

Record and Playback: UI-Level

Another approach to simulating user activities is to record user interactions with Graphical User Interface (GUI) – such as keystrokes and mouse clicks - and then play them back. Users, simulated using such approach, are sometimes referred as GUI users. The tools using this approach simulate users in the most accurate way: they really take the place of a real user. You are supposed to get end-to-end response times identical to what users would see.

Originally such tools were mostly used for automated functional testing, although the option to use this approach for load testing was available for a long time. For load testing, these GUI tools were usually used in conjunction with the load testing tool from the same vendor, which coordinated execution of multiple GUI scripts and collected results.

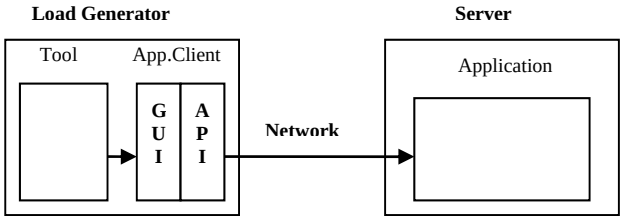

Fig.3 Record and playback approach, GUI users

The main problem with such tools is that these tools drive an instance of client software and require a machine for each user, so it is almost impossible to use them for a large number of simulated users – you need the same number of physical boxes as the number of users being simulated. Some tools have the ability to run one user per Windows Terminal Server session that significantly increases scalability of the solution (probably up to low hundreds of users from a practical point of view).

Another known option is, for example, using the low-level graphical Citrix or Remote Desktop protocols – which always was the last resort when nothing else was working, but were notoriously tricky to playback [PERF]. It works fine when you indeed use Citrix or Remote Desktop. But using it as a workaround means that you test a significantly different setup than you would use in real life (with multiple clients parts running on a server) that may undermine the value of testing.

Nowadays most applications have Web-based interface and a new generation of UI-level tools for browsers extend possibilities of the UI-level approach allowing to run multiple browsers per machine (so scalability is limited by the machine resources available to run browsers). Perhaps we can refer to users simulated by such tools as browser users (because more low-level browser control is usually used).

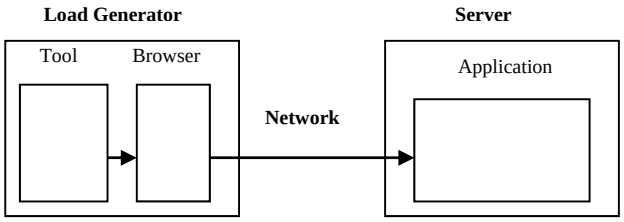

Fig.4 Record and playback approach, browser users

Moreover, UI-less browsers were created, such as HtmlUnit or PhantomJS, which require significantly less resources than real browsers. This drastically increased scalability of the UI-level approach and made it much more viable for load testing now, but the approach still remains less scalable than the protocol-level approach just because all these browsers (even the light-weight ones) still need to be run and all client-side application code be executed on the load generator machine.

Using the UI-level approach for load testing sounds very promising: we get end-user timing and do not depend on intricacies of the client-server communication. However questions of supported technologies, scalability, and timing accuracy remain largely undocumented, so the approach requires evaluation in every non-trivial case. So far the approach is mostly used to re-use existing functional testing scripts or when it is impossible to use protocol-level scripts.

Manual

Manual load generation isn't a real option if we want to simulate a large number of users. Still, in some cases, it can be a good option when we need load from a few users and don't have proper tools available or face serious issues with scripting. Sometimes a manual test can be a good option on earlier stages of testing to verify that the system can support concurrent work or to diagnose, for example, locking problems.

One of the concerns with manual testing is that even when each user has an exact scenario, time variations can occur; so the tests are not exactly reproducible due to variations in human input times. Such an approach hardly can be recommended as a long term solution, even with few users.

It still could be useful to run one or few users manually in parallel to simulated virtual users' workload to better understand what real users would experience. That is a good way to verify test results: if manual response times match what you see for the scripts, it is an indication that your scripts are correct.

Programming: Custom Test Harness

Programming is another approach to load generation. A straightforward way to create a multi-user workload is to develop a special program to generate workload. This program requires access to the API or source code and some programming work. It is often used to test components. No special testing tool is necessary (although some tools are available that can simplify work).

In some simple cases it could be the best solution (from a cost perspective, especially if there is no purchased load testing tool). A starting version could be quickly created by a programmer familiar with the API. A simple test harness, for example, could spawn several threads and each thread, simulating a real user, could include the

same sequence of API calls as the real software for that use case. No need to worry about what protocol is used for communication.

We successfully used this approach for component load testing in several projects (and, of course, this approach is widely used by developers). However, efforts to update and maintain the harness increase drastically as soon as you need to add such features as, for example:

- Complex user scenarios
- Centralized test management and result analysis
- Coordinated test execution from several computers

If you have numerous products, you really need to create something like a commercial load testing tool to assure all necessary performance and reliability testing. It probably isn't the best choice for a small group of testers.

Programming: Using Load Testing Tools

Many advance load testing tools support one (or several) scripting languages allowing to program scripts in whatever way is necessary while using the tool to manage scripts executions, collect and analyze the results. It may be direct programming of server requests or using web services, or using Application Programming Interface (API). If using API, the approach may need lightweight custom software clients (client stubs) to create the correct workload.

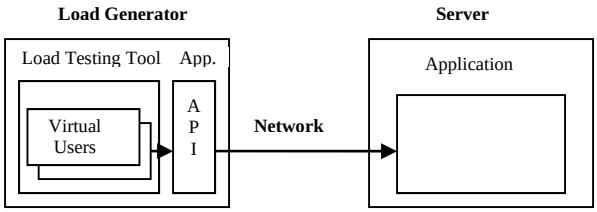

Fig 5. Programming API using a Load Testing Tool.

The implementation of this approach (we called it custom load generation) depends on the particular load testing tool. The original way was to create an external C dll (or shared library for UNIX) and then call functions defined in the dll from the tool's native script language.

Another way to implement this approach appeared in the later versions of load testing tools: creating a script in a programming language (such as Java or Visual Basic) with the help of templates and special tool-supplied functions.

These are the significant advantages of this custom load generation approach:

- It eliminates dependency on the third-party tool to support specific protocols.
- It leverages all the features of existing load testing tools and allows use of them as a test harness.
- It takes away the need to implement multi-user support, data collection and analysis, reporting, scheduling, etc. This is inherent in the third-party tool.
- It ensures that performance testing of current or future applications can be done for any protocol used to communicate among different tiers

Custom load generation may allow managing the workload in a more user-friendly way by simplifying parameterization.

For example, if you record socket-level traffic, recording and parameterization could take a lot of time. And if you need to change the workload (for example, use new queries), it is almost impossible to change the parameterized script to reflect the new workload. You probably need to re-record and re-parameterize the script.

When you implement custom load generation, the real query could be read from an input file. Changing the query becomes very easy: you just change the input file without any changes in the script.

The same is true if different builds of the software are tested. Small changes could impact a low-level protocol script, but the API is usually more stable. Just install the new build and run the test. There is no new recording and parameterization needed.

But, of course, there are some considerations to keep in mind for the custom load generation approach:

- It requires access to API or source code.
- It requires additional programming work.
- It requires an understanding of internals (to re-create the sequence used by real users).
- The client environment should be set up on all load generator machines.
- It requires commercial tool licenses for the necessary number of virtual users.
- The lowest level transaction that can be measured is an external function.
- It usually requires more resources on client machines (since there is some custom software).
- The results should be carefully interpreted (to insure that there is no contention between client stubs).

Programming may be a better solution in many cases, but it is not a full replacement of recording approaches. In cases when recording works well, it usually provides better and more efficient solutions. One of important advantages of recording is that that the tool records exactly whatever communication happens between user and server – while with programming it is often what the person creating script *think* the communication is. Unfortunately communication between user and server is often very complicated and difficult to reproduce programmatically. So the tools that support only programming (and not supporting recording) have rather limited area of application.

Custom Load Generation Examples

Two examples below are for LoadRunner - just because it is the tool we use most. Similar things can be done other tools.

The first example is a multi-dimensional analytical engine. Originally the main way to access it was through the C API; many products use it, including Excel Add-in. It is possible to record a script using the Winsock protocol (a low-level protocol recording all network communication); Winsock scripts are quite difficult to parameterize and verify.

Here is a small extract of a correlated Winsock script:

```
lrs_create_socket("socket0", "TCP", "LocalHost=0",
        "RemoteHost=ess001.hyperion.com:1423",  lrsLastArg);
lrs_send("socket0", "buf0", LrsLastArg);
lrs_receive("socket0", "buf1", LrsLastArg);
lrs_send("socket0", "buf2", LrsLastArg);
lrs_receive("socket0", "buf3", LrsLastArg);
lrs_save_searched_string("socket0",      LRS_LAST_RECEIVED, "Handle1",
        "LB/BIN=\\x00\\x00\\v\\x00\\x04\\x00",     "RB/BIN=\\x04\\x00\\x06\\x00\\x06", 1, 0, -1);
lrs_send("socket0", "buf4", LrsLastArg);
lrs_receive("socket0", "buf5", LrsLastArg);
lrs_close_socket("socket0");
```

Another part of the script includes the content of each sent or received buffer:

```
send  buf22 26165
"\xff\x00\xf0\a"
"\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x03\x00"
"d\x00\b\x00"
"y'<Handle1>\x00"
"\b\r\x00\x06\x00\f\x00\x1be\x00\x00\r\x00\xd6\aRN"
"\x1a\x00\x06\x00\x00\x00\x00\x00\x00\x00\x00\b"
"\x00\x00\x00\xe7\x00\x00\x01\x00\x03\x00\x04\x00"
"\x10\x00\xcc\x04\x05\x00\x04\x00\x80\xd0\x05\x00\t"
"\x00\x02\x00\x02\x00\b\x00<\x00\x04"
```

```
"FY04\aWorking\tYearTotal\tELEMENT-F\tProduct-P"
"\x10<entity>\t\x00\x02\x00"
…
```

The script consists from many pages of such binary data. Correlating such scripts is very time-consuming and the resulting scripts are almost impossible to parameterize – if you need to change anything in the query (for example, run it for another city) you need to start from a scratch.

An external dll was made for major functions. Below is a script using this external dll:

```
lr_load_dll("c:\\temp\\lr_ess.dll");
pCTX = Init_Context();
hr = Connect(pCTX, "ess01", "user001","password");
…
lr_start_transaction("Mdx_q1");
sprintf(report, "SELECT %s.children on columns,
  %s.children on rows FROM Shipment WHERE
  ([Measures].[Qty Shipped], %s, %s)",
  lr_eval_string("{day}"),  lr_eval_string("{product}"),
  lr_eval_string("{customer}"),
  lr_eval_string("{shipper}"));
hr = RunQuery(pCTX, report);
lr_end_transaction("Mdx_q1",LR_AUTO);
```

The lines above are almost the whole script (except a few technical lines) instead of many pages of binary data. An MDX query is generated using day, product, customer, and shipper as parameters, so we hit the different spots of the database and avoid artificial caching effects.

Another example is a middleware product (without GUI interface, only an administrative console). We were given functional test scripts in Java. The product can use HTTP (with major application servers) or TCP/IP (as a stand-alone solution). It is possible to run a test script and record HTTP traffic between the script and the server. It is HTTP, but it is just binary data inside the HTTP request body. You can't do anything with them; you can only play them back as is. You need start from a scratch if you want to make a small change.

The solution that we finally used was the creation of LoadRunner scripts from the test script directly. Just put Java code inside the template and add tool-specific statements (like lr.start_transaction and lr.end_transaction). Here is how the beginning of the script looks:

```
import lrapi.lr;
import com.essbase.api.base.*;
import com.essbase.api.session.*;
…
public int action() {
String s_userName = "system";
String s_password = "password";
lr.enable_redirection(true);
try {
lr.start_transaction("01_Create_API_instance");
ess = IEssbase.Home. create
  (IEssbase.JAPI_VERSION);
lr.end_transaction
  ("01_Create_API_instance", lr.AUTO);
lr.start_transaction("02_SignOn");
IEssDomain dom = ess.signOn(s_userName,
  s_password, s_domainName, s_prefEesSvrName,
  s_orbType, s_port);
lr.end_transaction("02_SignOn", lr.AUTO);
…
```

It is possible, of course, to create a simple program that will start many such scripts in parallel, but you need to implement all the infrastructure (coordination, results analysis, monitoring, etc.) yourself. Such work is usually not a good option for a small group working with many different products. It makes much more sense when an existing tool provides this infrastructure. However most inexpensive or free tools, unfortunately, are weak in providing such functionality.

Selecting Load Testing Tools

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 criteria allowing to differentiate load testing tools and it is probably better to evaluate tools on each criterion separately.

First, as we discussed above, there are three main approaches to workload generation and every tool may be evaluated on which of them it supports and how exactly:

Protocol-level recording and the list of supported protocols. Does the tool support protocol-level recording and, if it does, what protocols it supports. With quick Internet growth and popularity of browser-based clients, most products support HTTP only or a few Web-related protocols. According to my knowledge, only HP LoadRunner and Microfocus SilkPerformer try to keep up with support of all popular protocols. So if you need recording of a special protocol, you probably end up into looking at these two tools (unless you find a special niche tool supporting your specific protocol). That somewhat explains the popularity of LoadRunner at large corporations where you probably have almost all possible protocols used. 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 you may look for that kind of specific support. For example, Oracle Application Testing Suite may have better support of Oracle technologies.

UI-level recording. The option was available for a long time, but it is much more viable now. For example, there was a possibility to use Mercury/HP WinRunner or QuickTest Professional (QTP) scripts in load tests, but you needed a separate machine for each virtual user (or at least a separate terminal session). That limited the level of load you may achieve drastically. Other known options were, for example, Citrix and RDP (Remote Desktop Protocol) protocols in LoadRunner – which always were the last resort when nothing else was working, but were notoriously tricky to playback. New UI-level tools for browsers, such as Selenium, extended possibilities of the UI-level approach allowing to run multiple browser per machine (so scalability is limited by resources available to run browsers). Moreover, there are UI-less browsers, such as HtmlUnit or PhantomJS, which require significantly less resources than real browsers. There are multiple tools supporting this approach now – such as PushToTest directly harnessing Selenium and HtmlUnit for load testing or LoadRunner TruClient protocol and SOASTA CloudTest using more proprietary solutions to achieve low-overhead playback. Still questions of supported technologies, scalability, and timing accuracy remain largely undocumented, so the approach requires evaluation in every specific non-trivial case.

Programming. There are cases when you can't (or can, but it is more difficult) use recording at all. In such cases using API calls from the script may be an option. Other variations of this approach are web services scripting and using of unit testing scripts for load testing. And, of course, you may need to add some logic to your recorded script. You program the script using whatever way you have and use the tool to execute scripts, coordinate their executions, report and analyze results. To do this, the tool should have ability to add code to (or invoke code from) your script. And, of course, if tool's language is different from the language of your API, you would 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 you should know all details of the communication between client and server that is often very challenging.

Other important criteria are related to the environment:

Deployment Model. There were a lot of discussions about different deployment models: lab vs. cloud vs. service. There are some advantages and disadvantage of each model. Depending on your goals and systems to test you may prefer one deployment model over another. But for comprehensive performance testing you may really need both lab testing (with reproducible results for performance optimization) and realistic outside testing from around

the globe (to check real-life issues that you can't simulate in the lab). Doing both would be expensive and makes sense when you really care about performance and have a global system – but it is not rare and if you are not there yet, you can get there eventually. If there are such chances, it would be better to have a tool which supports different deployment models.

Either it is lab or cloud, an important question would be what kind of software / hardware / cloud the tool requires. Many tools use low-level system functionality, so it may be unpleasant surprises when the platform of your choice or your corporate browser standard is not supported.

Scaling. When you have a few users to simulate, it usually is not a problem. The more users you need to simulate, the more important it becomes. The tools differ drastically on how many resources they need per simulated user and how well they may handle large volumes of information. It may differ significantly even for specific tool depending on protocol used and specifics of your script. As soon as you get to thousands of users, 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. Load testing appliances (for example, Spirent Avalanche) can be useful for simulating a large number of simple Web users, but scripting is usually limited.

Two other important sets of functionality are **monitoring** of the environment and **result analysis**. While theoretically it is possible to do it using other 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 very important. And the more complex system and tests, the more important they are.

Of course, non-technical criteria are important too:

Cost/Licensing Model. There are commercial tools (and license costs differ drastically) and free tools. And there are some choices in between: for example SOASTA has the CouldTest Light edition free up to 100 users. There are many free tools (some, as JMeter, are mature enough and well-known) [OPEN] and many inexpensive tools, but most of them are very limited in functionality.

Skills. Considering a large number of tools and a relatively small number of people working in the area, there is a kind of labor market only for the most popular tools. Even for the second-tier tools there are few people around and few positions available. So if you don't choose the market leaders, you can't count that you find people with this tool experience. Of course, an experienced performance engineer will learn any tool – but it may take some time until productivity will get to the expected level.

Support. Recording and load generation has a lot of sophistication in the background and issues may happen in every area. Availability of good support may significantly improve productivity.

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

And while you may evaluate tools with above mentioned criteria, 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 system 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, of course, check it with at least the most important systems).

On the other hand, it is always good to keep in mind that a load testing tool is only a tool. While you probably need a sophisticated set of tools to create a luxury furniture set, you need only a hammer to nail a picture to the wall.

Summary

Load testing is an important integral part of the performance engineering process. Maybe it would be less need for "performance testers" as it was at the load testing heyday due to better instrumenting, APM tools, continuous integration, etc. – but we may expect more need for performance experts that would be able to see the whole picture using all available tools and techniques.

There is no best approach to load generation or, moreover, best load testing tool. Some approaches or tools may be better in a particular context. It is quite possible that a combination of tools and approaches would be necessary in complex environments. Choosing the right strategy in load generation can be a challenging task. While digging deeply into details of particular projects and tools may be needed, it is good to see a bigger picture of what approaches and tools are available and what are their advantages and disadvantages.

References

[BARB06] S.Barber, “User Experience, Not Metrics” (2006). http://www.perftestplus.com/resources/UENM1.pdf

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

[JAIN91] R. Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling", Wiley (1991).

[MICR04] Improving .NET Application Performance and Scalability, Microsoft Press (2004).
http://msdn.microsoft.com/en-us/library/ff649152.aspx

[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/

[PODE01] A.Podelko, A.Sokk, L.Grinshpan, Custom Load Generation, CMG (2001).

[PODE05] A.Podelko Workload Generation: Does One Approach Fit All?, CMG(2005).

[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, September (2002). http://www.soft.com/News/QTN-Online/qtnsep02.html