The Association of System Performance Professionals

The **Computer Measurement Group**, commonly called **CMG**, is a not for profit, worldwide organization of data processing professionals committed to the measurement and management of computer systems. CMG members are primarily concerned with performance evaluation of existing systems to maximize performance (eg. response time, throughput, etc.) and with capacity management where planned enhancements to existing systems or the design of new systems are evaluated to find the necessary resources required to provide adequate performance at a reasonable cost.

This paper was originally published in the Proceedings of the Computer Measurement Group's 2005 International Conference.

For more information on CMG please visit www.cmg.org

WORKLOAD GENERATION: DOES ONE APPROACH FIT ALL?

Alexander Podelko
Hyperion Solutions
alexander_podelko@hyperion.com

A must task in load testing is workload generation: how to apply load to your system. 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, to use a load testing tool or to create a program to do it. Many tools allow you to use different ways of recording/playback and programming. This paper discusses pros and cons of each approach based mainly on experience with distributed business applications.

Much has been written about how to design scalable software, what best practices and design patterns to use, and even how to build models to predict performance (for example, [SMITH02] or [MICR04]). While these topics are very important to create scalable software, theories and best practices can't guarantee a required level of performance. Testing multi-user applications under realistic, as well as stress, loads remains the only way to ensure appropriate performance and reliability in production.

Many terms are used to describe such kinds of testing, for example: load testing, performance testing, stress testing, scalability testing, reliability testing. Despite many efforts to define clear distinctions between them, none of them are widely accepted [STIR02]. There are no clear distinctions because these terms describe testing from different points of view.

Without diving too deeply in details, we can define:

- load testing is testing when you apply load to the system,
- performance testing is testing the performance of the system,
- stress testing is testing how the system behaves under stress (heavy load),
- scalability testing is testing how the system scales with increasing load and/or resources.

Quite often, similar processes are used in all these kinds of testing, and a term can be chosen depending on what looks most important.

If you run a test simulating many users and measuring response times, what should you name your test? You can probably refer to it as the load test or the performance test, and both would be correct. They are not synonyms, they describe different sides of the test.

The term "load testing" is used in this paper because we are investigating ways to create load. Everything mentioned here applies to performance, stress, scalability, reliability and other kinds of testing - so long as the system is tested by applying load (while, for example, reliability testing by switching off the power is another story).

Based on classic functional testing from one side, and on system performance analysis from another side, load testing is emerging as an engineering discipline of its own. Quite often load testing is combined with tuning, diagnostics, and capacity planning. Sometimes it is difficult to separate them. For example, performance testing of a mistuned system isn't too meaningful. The typical load testing process is depicted on figure 1 (for variations see [BARB04], [MICR04]).

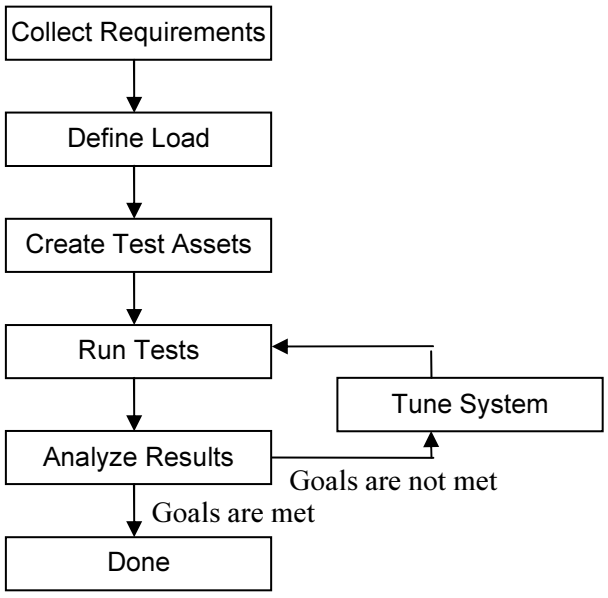

Fig.1 Load testing process

We explicitly define two different steps, "define load" and "create test assets" here. The "define load" step means the logical description of the load we want to apply (for example, a group of users that login, navigate to a random item in the catalog, add it to the

shopping cart, pay, and logout with an average 10 second think time between each pair of actions). The "create test assets" step means to convert the logical description into something that will physically create load during the "run tests" step. While for manual testing it can just be a description given to each tester, usually it is something else in load testing – a program or a script.

Before you can move forward from "define load" to "create test assets" you need to decide how you 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 have few users), use a load testing tool (software or hardware), or create a program to do it. Many tools allow you to use different ways of recording/playing back and programming.

The following provides a description of different approaches to aid in making realistic decisions about which approach and which tool may be most appropriate. The material is based on experience with business applications, so limitations may exist for dealing with other environments.

Record and Playback: Virtual Users

The mainstream approach of load testing (at least for distributed 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 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 the 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).

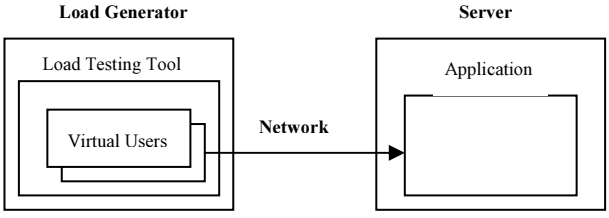

Fig.2 Record and playback approach, virtual users

Both recording and playback happen between the tiers, so the protocol used between the client and the server is extremely important. Other factors, like 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 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.

Load Testing Tools

A few tools support the recording and playback approach for a variety of protocols. Usually they are the most mature commercial products. Such enterprise-level load testing tools have many important features. The following features could be considered typical for such tools:

- Ability to record scripts automatically for different protocols
- Powerful scripting language
- Simulating numerous users (limited mainly by hardware)
- Coordinated test execution from several computers
- Centralized test management and result analysis
- Support for different environments
- Ability to monitor environments
- Ability to use other approaches to load generation (considered in detail below)
    - Ability to simulate GUI users as well as virtual users
    - Ability to extend scripting language and make external calls
- Interface with other development and test software: requirements gathering, test

management, defect tracking, configuration management, etc.

The list of supported features differs from tool to tool. Examples of powerful multi-protocol tools are Mercury LoadRunner (www.mercury.com), Segue SilkPerformer (www.segue.com), IBM Rational Performance Tester (www.ibm.com/software/rational), and Compuware QALoad (www.compuware.com). For a Web-only commercial tool, Empirix e-Load (www.empirix.com), having some features of enterprise-level load testing tools, probably is best known.

The five above-mentioned vendors accounted for 95% of the worldwide distributed automated software quality commercial tools market in 2003 according to IDC: Mercury 55.6%, IBM/Rational 22.5%, Compuware 9.7%, Segue 4.1%, and Empirix 3.1% [IDC04]. These numbers are not from load testing tools alone, but the statistics still give an idea about the market.

Many other specialized tools are available, especially for Web technologies. If the number of technologies you'll use is limited, it makes sense to check out such tools. Specialized tools weren't a real option for us because of the multiple technologies we have been working with. Most specialized tools can be found in these two lists:
www.softwareqatest.com/qatweb1.html
www.testingfaqs.org/t-load.html
Not all listed tools support the record and playback approach; some require programming.

Recording abilities of tools differ significantly. Enterprise-level load testing tools usually can work in more sophisticated environments and do more correlation automatically (like getting real cookies, session ids, etc. from the server instead of recorded values).

Another area of differentiation is infrastructure (test coordination, results analysis, monitoring, integration with other tools, etc.). Most inexpensive or free tools, unfortunately, are weak in this regard.

One more tool worth mentioning is Microsoft Application Center Test (ACT) coming with Visual Studio .Net, although it is rather limited in functionality. The Visual Studio 2005 Team System for Software Testers will include a much more powerful load testing tool.

There are many open source tools. For example, the following link included 21 tools at the moment of writing:
www.opensourcetesting.org/performance.php

Unfortunately most tools have limited functionality. Probably OpenSTA and Apache JMeter are the best known and most mature open source tools.

OpenSTA (www.opensta.org) is a web load testing tool originally developed as a commercial tool by Cyrano. OpenSTA stands for Open Systems Testing Architecture. Another branch of the Cyrano code is a commercial tool QuotiumPRO from Quotium (www.quotium.com).

Apache JMeter (jakarta.apache.org/jmeter) is a 100% pure Java tool for load and performance testing HTTP and FTP servers as well as arbitrary database queries (via JDBC).

Probably the most ambitious open source project is the Eclipse Test & Performance Tools Platform (http://www.eclipse.org/tptp/index.html), but it isn't quite clear what load testing functionality is available right now.

Load testing appliances (for example, Spirent Avalanche) can be useful for simulating a large number of simple Web users. Usually, scripting is limited. It is interesting that Spirent is a partner of Mercury and they position their hardware load generator as a complement to LoadRunner to create heavy, but simple, background load.

Choosing a Load Testing Tool

Generally, it would be wrong to say that one tool is better than another, but one tool can fit better in a particular environment than another. Many factors beyond functionality can impact the choice. Here are some:

- familiarity with the tool and other tools from that vendor
- familiarity with languages the tool uses (many are based on standard languages such as C, Basic, or Java)
- support
- price
- vendor's prospective

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.

Limitations

We have been using the record and playback approach in most projects, but, unfortunately, it has several serious limitations:

- It usually doesn't work for testing components.
- Each particular load testing tool supports a limited number of technologies.
- 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, we were not able to use recording for the SMB (Server Message Block) protocol, later succeeded by the Common Internet File System (CIFS) protocol. It is used when two Microsoft network systems communicate over a network. Its commands are embedded within the transport protocols like TCP/IP.

Back in 1999, we weren't able to use recording for Microsoft DCOM (Distributed Component Object Model); it is used for communication between two remote COM components. Nor we were able to use recording for Java RMI (Remote Method Invocation); it is used for communication between two remote Java programs.

Although some vendors claim their products support these protocols, they cannot work in all environments. Script recording and parameterization are still far from being straightforward and often require a good knowledge of system internals. The question of workload validation is also opened. A good illustration of possible problems is the code below.

Here is an example of recorded RMI protocol:

```
 _integer =
    _ireportserver.executeJob(_designjobobject);
_ireportserver.getStatus(new Integer(3));
_ireportserver.getStatus(new Integer(3));
_ireportserver.getStatus(new Integer(3));
_iinstance = _ireportserver.getInstance
    (new Integer(3));
```

Here is the real code producing this RMI communication:

```
   joID = poReportServer.executeJob(djo);
   bStatus = true;
  while (bStatus) {
     bStatus = poReportServer.getStatus (joID);
     Thread.sleep(300); }
   poReportServer.getInstance(joID);
```

The client polls the server each 300 ms to check the status and get the result as soon as it is ready. Without knowledge of the real code it is almost impossible to parameterize the script properly – it just calls getStatus three times and then calls getInstance even if the result won't be ready yet.

So, it is possible that the record and playback approach won't work in your environment, or that using the approach is 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: GUI Users

Another type of tools uses the recording approach. These tools record all actions of a real user: mouse moving and clicking, keystrokes. These tools are usually used for functional and regression testing. Examples are Mercury WinRunner, Mercury QuickTest Professional, and Rational Robot. They record and playback communication between the user and client GUI. Users, simulated using such tools, are often referred as GUI users.

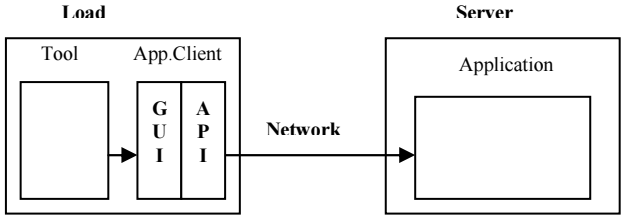

Fig.3 Record and playback approach, GUI users

These tools simulate users in the most accurate way; they really just take the place of a real user. You get end-to-end response times identical to what users would see.

For load testing, these GUI tools are usually used in conjunction with the load testing tool from the same vendor, which coordinates execution of multiple GUI scripts and collects results.

The main problem with such tools is that they 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, it significantly increases scalability of the solution (probably up to low hundreds of users

from a practical point of view). Another workaround from Mercury, for example, is using the low-level graphical Citrix protocol. Still, it is a significantly less scalable approach than record and playback with virtual users because you need to have full working client software (which adds significant overheads on load generating machines).

These tools also could be useful in combination with virtual users to verify VU scripts, get end-to-end timing, or increase the number of use-cases during load testing re-using functional testing scripts (of course, if the functional testing tool matches the load testing tool).

Manual

Manual load generation isn't a real option if you want to simulate a large number of users. Still, in some cases, it can be a good option when you need load from a few users and don't have proper tools available or you face big problems 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 scripts (keep in mind that virtual users don't have client-side overheads) it is one more indication that your scripts are correct.

Programming

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 your 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 some threads and each thread, simulating a real user, could include the same sequence of API calls as the real software for that use case. Such a harness should work if the API works. You don't 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 (as was true in our case) 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.

Custom Load Generation

Originally we used the record and playback approach (load testing tools) or created special programs to generate workload (custom test harnesses) in cases where recording didn't work. Since we experienced numerous problems applying the two above-mentioned approaches to new products utilizing the latest technologies, we came to the idea of a mixed approach. This mixed approach involves developing lightweight custom software clients (client stubs) to create the correct workload but use powerful commercial tools to manage them and analyze the results [PODE01].

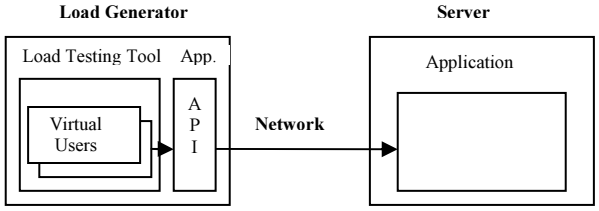

Fig 4. Custom load generation.

The implementation of this approach (we called it custom load generation) depends on the particular load testing tool. For the Rational load testing tool and Mercury LoadRunner, 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 (like 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 commercial 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. In some instances, it is the only way to generate load (as it was for SMB, DCOM, and RMI in our case) without developing a full-scale custom harness.

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.
- 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).

Custom load generation has one more advantage: it may allow managing the workload in a more user-friendly way while 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.

Custom Load Generation Examples

All examples below are for Mercury LoadRunner - just because it is the tool we use most. Similar things can be done with the Rational performance tool and probably some 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. We have a full methodology on how to correlate such scripts, but it is very time-consuming (you should go through all pages of the binary data and replace hard-recorded handles with parameters). 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 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. We can create scripts for each function that was included into the dll (that cover the main functionality of the product).

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);
…
```

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

Summary

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. You need to dig deeply into details of particular tools for a particular project, but it is good to see the big picture of what is available and what can be used for that and other projects.

This paper describes our experience of multi-user workload simulation using different methods of load generations. These included recording/playback, programming, and a mixed method (custom load generation). Custom load generation involves implementing low-weight custom client software and running it with a commercial load testing tool which is used as a harness to collect, analyze and report results, as well as manage test execution. Select the set of methods that seem most appropriate to you, and then evolve your approach to yield the best results.

References

[BARB04] S.Barber, "Beyond Performance Testing" (2004). http://www.perftestplus.com/pubs.htm

[IDC04] “Worldwide Distributed Automated Software Quality Tools 2004-2008 Forecast and 2003 Vendor Shares”, IDC (2004).

[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/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp

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

[SEGUE] “Choosing a Load Testing Strategy”. Segue white paper. https://www.segue.com/_files/choosing_a_load_testing_strategy.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