CUSTOM LOAD GENERATION

Alexander Podelko, Arno Sokk and Leonid Grinshpan
Hyperion Solutions
{Alexander_Podelko, Arno_Sokk, Leonid_Grinshpan}@hyperion.com

Commercial load generation tools allow recording tier-to-tier communication and creating workloads via playback of the recorded script. Unfortunately, they do not work for all technologies. The alternative of creating a special test harness becomes more time-consuming when faced with complicated scenarios and a need to analyze results. This paper discusses our experience using an intermediate approach: developing a custom virtual client and running it using commercial tools (with powerful management and reporting features) to arrive at a technology-independent solution.

Load testing

Testing applications for multi-user performance and reliability is necessary to avoid failure in today's technology. It is rather special kind of testing, requiring special tools, knowledge, and in-depth analysis.

The most challenging task here, according to our experience, is to create a reproducible multi-user workload matching real-life user scenarios in a limited time.

This paper will discuss several approaches to the problem, summarizing the load testing experiences of the performance group at Hyperion Solutions Corp.

Hyperion Solutions is a vendor of business analysis software. The company's multidimensional database Essbase, packaged business analysis applications and tools are used by 6,000 organizations worldwide, including more than 60 of the Fortune 100.

The performance group is shared between all development teams and responsible for performance, scalability, and reliability testing. It supplements the functional testing done by the Quality Engineering group for each product.

In total, there are about two dozens of different products to test (each usually has different versions and builds and works on different platforms). Some of them work in combination with others as well as third-party products. Often a product has several kinds of clients using different protocols for communication (for example, a Web client using HTTP protocol or a Win32 client using Java RMI protocol).

So there are infinite combinations of products and workloads using different technologies and we need to create meaningful and realistic workloads in a timely manner (usually in the development cycle timeframe) in order to do comprehensive performance and reliability testing.

Originally we used the "record and playback" approach (load testing tools) or created a special program to generate workload (custom test harness) from scratch, but often found that neither of them is the best way.

"Record and playback" approach

The main idea of this approach is to record communication between two tiers of software and then playback an automatically created script (usually, of course, after proper parameterization).

There are several universal load testing tools on the market today that support such an approach (and are much more specialized, especially for testing web performance). The following features could be considered as standards for such tools:

- Ability to record scripts automatically for different protocols
- Simulating numerous users (limited mainly by hardware configurations)
- Centralized test management and result analysis
- Coordinated test execution from several computers
- Support for different environments
- Ability to simulate GUI users as well as virtual users

We successfully use two load testing tools in our group: Mercury LoadRunner and Rational Test (PerformanceStudio, preVue-C/S). The evaluation and choice were made in 1997. These tools are always changing, so there is not much sense to discuss why they were chosen at that time.

The choosing of the right load-testing tool is a separate large and interesting topic and is out of the range of this paper. All further examples are for Mercury LoadRunner and Rational Test because these are the tools with which we have experience.

I believe that everything discussed here could also be applied to other tools (for example, Segue SilkPerformer or Compuware QALoad). But, of course, the implementation details would be different.

We use the "record and playback" approach in most projects but, unfortunately, it doesn't work for all technologies.

For example, we were not able to use the "record and playback" approach for following protocols:

- SMB (Server Message Block) protocol, later succeeded by 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. We worked with Rational development and support for a long time but without success.

- Microsoft DCOM (Distributed Component Object Model). Used for communication between two remote COM components. At the moment of the evaluation (1999), only Mercury claimed DCOM support but it didn't work in our environment. After a couple of months of work with support without any success, we implemented another approach, described below. Since this approach worked fine we didn't return to the evaluation of DCOM "record and playback" support, although it was significantly improved in all load testing tools.

- Java RMI (Remote Method Invocation). Used for communication between two remote Java programs. At the moment of the evaluation (1999), nobody supported our environment (Microsoft JVM). Now, after migration to Sun JVM environment, we tried Mercury LoadRunner 7.0 and found that it works fine recording and replaying Java script. So now we are re-evaluating available approaches to RMI communications.

Custom test harness

Another straightforward way to create a multi-user workload is a custom test harness (special program to generate workload). It requires access to the API or source code and some programming 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 simple harness could spawn some threads and each thread will simulate real user.

This approach was used in several projects for component-level testing and was very useful sometimes. But as soon as a harness is developed, you need to add such features as, for example:

- Complex user scenarios
- Centralized test management and result analysis
- Coordinated test execution from several computers
- Ability to run GUI users as well as virtual ones

Efforts to update and maintain the harness increase drastically. If you have numerous products (as Hyperion does) you need really 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.

Custom load generation

Since we experienced numerous problems applying the two above-mentioned approaches to Hyperion's new products utilizing the latest technologies, we came to the idea of a mixed approach that in this paper is named "custom load generation".

The main idea of this approach is the development of lightweight custom software clients (client stubs) to create the correct workload but use powerful commercial tools to manage them and analyze the results.

The implementation of this approach depends on the particular load testing tool (this will be considered in details later). For Rational Test and Mercury LoadRunner, the more mature way is 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 (VU script for Rational Test, Vuser script for Mercury LoadRunner; both are C-like script languages).

Another way to implement this approach appeared in the latest versions of load testing tools: to create a script in a programming language (Visual Basic, C and C++ for Mercury LoadRunner; Java, Visual Basic and shell script for Rational Test) with the help of templates and special tool-supplied functions.

These are the advantages of this custom load generation approach:

- Eliminates dependency of the third-party tool's ability to support specific protocols
- Leverages all the features of commercial tools and allows using them as a test harness
- No need to implement multi-user support, data collection and analysis, reporting, scheduling, etc. This is inherent in the third-party tool.
- 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 quickly create a performance testing environment (as for SMB, DCOM and RMI, in our case) without developing a full-scale custom harness.

But this approach has one more advantage: it allows managing the workload in a more user-friendly way.

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, for example, could be read from an input file and 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 rarely change the API. 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:

- Requires access to API or source code
- Require additional programming work
- Requires commercial tool license for necessary number of virtual users
- Minimal transaction that could be measured is an external function
- Usually requires more resources on client machines (since there is some custom software)
- Results should be cautiously interpreted (insure that there is no contention between client stubs)

Implementations for Rational Test (PerformanceStudio)

The mature way to create a custom software client in Rational Test is to implement it as an external dll. It should be a set of C functions compiled as a dll (the functions could be written in C++ and declared as extern "C").

This external dll should then be placed in a specific directory and references to it are added to the script properties.

There are some limitations. Only a limited set of types can be function arguments:

C types	VU types
`Int`	`Int`
`char *`	`string /*read only */`
`char *`	`string:maxsize /*writable */`
`int *`	`Int[], int[][], int[][][]`
`char **`	`string[], string[][], string [][][]`

For example, let's create the external dll for C function void DoSomething (int n):

```
extern     "C"{__declspec(dllexport)     void
DoSomething(int n);}
#include "windows.h"

void DoSomething(int n)
{
       …    //some processing
}
```

VU script (C-like Virtual User language script using by Rational Test) to call this function would be:

```
#include <VU.h>
external_C proc DoSomething(n)
int n;
{}
int p=3000; //a parameter

{
  start_time ["T1"];
  DoSomething(p);
  stop_time ["T1"];
}
```

Since we use Rational Test as a framework to run this VU script we get a comprehensive set of available reports, abilities to simulate numerous users from several machines and create complex scenarios as well as other useful features of this load testing automatically.

Usually functions (like DoSomething in the example above) included in the dll are a wrapper around specific C or C++ API functions. In simpler cases the C API could be directly called from the VU script, without the creation of wrapping functions.

Unfortunately if the software is written in another language it isn't too simple to use this approach. For example, to run a custom client written in Java, the java virtual machine (JVM) was started each time with parameters to run the particular report by the Win32 CreateProcess function.

This probably wasn't the most elegant approach but it worked and allowed all necessary performance testing to be completed.

Rational 2001 TestManager can work not only with SQABasic (for GUI testing) and VU language scripts but also with Java, Visual Basic and shell scripts (test script services). It could significantly simplify using custom clients written in these languages.

For example, instead of the awkward starting of JVM by CreateProcess for each report for Java custom client as in the above example (or some other non-trivial transformation), it is possible to create a Java script that will do everything, something like (without exception handling):

```
import java.io.*;
import com.rational.test.tss.*;

public class hr extends
com.rational.test.tss.TestScript
{
  public void testMain(String args[])
  {
      int n=30000;
      myClass mc;

      new hr();
      mc = new myClass();

      TSSMeasure.timerStart("T1");
      mc.doSomething(n);
      TSSMeasure.timerStop("T1");
  }
}
```

This Java script is now just a standard script for Rational Test and utilizes all features of the load testing tool. Results could be seen through a set of reports available in Rational Test.

Implementations for Mercury LoadRunner

Using LoadRunner, a custom software module could be implemented as an external dll. LoadRunner could use the lr_load_dll function to load the external dll (shared library) in Vuser script or it could be defined globally in the vugen.dat file. Then functions defined in the dll could be used without declaration in the script.

For the external dll described above with void DoSomething(int n) C function, a simple Vuser script would look like:

```
#include "as_web.h"

Action1()
{

  int p=3000; //a parameter

  lr_load_dll("c:\\DoSomething.dll");
  lr_start_transaction("T1");
  DoSomething(p);
  lr_end_transaction("T1", LR_AUTO);
  return 0;
}
```

It is just a standard Vuser script that is running inside the LoadRunner framework. It could be a part of a complex scenario and LoadRunner, which produces a comprehensive set of various reports, would analyze its results.

Another way could be to program a script in Visual Basic, C or C++ using special templates and LoadRunner's functions. However, this was not attempted.

Summary

This paper described our experience of multi-user workload simulation using a mixed method (custom load generation): 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.

This approach eliminates technology limitations of the "record and playback" approach (it didn't work with SMB, DCOM and RMI for us) and in some cases facilitates the creation of a more flexible test harness.

Of course, it isn't a silver bullet. But sometimes (often enough in our case) it is a very good way to simplify the creation of a multi-user workload.

It also isn't something especially new. All these features have been available in commercial load testing tools for a long time. But usually they are

described at the very end of a user guide, somewhere in a section for advanced users.

The goal of this paper was to show the importance of these extendibility features of load test tools and the real benefits that could be gotten from them.