UDC 004.422.81

UNIVERSAL TOOL FOR STRESS-TESTING A WEBSOCKET BACKEND WITH A BINARY PROTOCOL

Pigovskyy Y.R.[1], Yuzvin N.I.[2]
Ternopil National Economic University
[1] Ph.D; [2] graduate student

I. Introduction

Load testing is one of very important phases during delivering enterprise software to industry. It provides ability to discover how large amount of users will affect overall system quality of service.

It is of large practical value to have a domain-agnostic tool, which can be used to test backends which communicates through any request-response and transport protocols.

There are a lot of different software devoted to load testing of enterprise systems, for example: Apache JMeter and gatling.

Apache JMeter can be used to measure latency of processing general requests by a server. JMeter may be used to test performance both on static and dynamic resources (Webservices (SOAP/REST), Web dynamic languages - PHP, Java, ASP.NET, Files, etc. -, Java Objects, Data Bases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server/script/object behavior under heavy concurrent load [1].

JMeter documentation on distributed testing [2] discusses following terms:

• Master — the system running Jmeter GUI, which controls the test

• Slave — the system running jmeter-server, which takes commands from the master GUI and send requests to the target system(s)

• Target — the webserver we plan to stress test

JMeter supports many different protocols, in particular: Web (HTTP/HTTPS), FTP, JDBC, LDAP, JMS. But its support of Websocket is still under development. Therefore we need to implement a testing tool of our own or to create a JMeter plugin.

But these software cannot be aplied to stress-test of a server communicating using non-textual, binary protocol.

One of a modern approaches to effectively create such binary protocols is Google's protocol-buffers. According to an official document [3] protocol buffers in comparison with textual serialization protocols such as XML or JSON:

• are simpler

• are 3 to 10 times smaller

• are 20 to 100 times faster

• are less ambiguous

• generate data access classes that are easier to use programmatically

II. Domain-agnostic requests

In order to develop a universal stress-test tool it is necessary to define server requests in domain-agnostic manner. Let's split up all possible requests on two groups:

• on demand,

• instant.

On demand requests are those, which change data at server and these data can be obtained by system users in some moment in future, while instant requests not only change data at server, but also notify other users about the change immediately, i.e. when a user sends an instant request, server notifies other user(s) about it as soon as possible.

Taking this description of abstract requests into account, load tests can be split on two categories:

• extreme number of requests (performance testing),

• extreme number of online users

In current study, tests of both these categories will be carried out.

III. Requirements

Enterprise software is build up from a number of cooperating parts:

• application container,

• database,

• caching engine,

• load balancer.

All these parts take their role in processing requests and will be a bottleneck in one or other case. Enterprise software requires significant hardware resources to fulfil needs of large amount of users making huge number of simultaneous requests.

From other side, testing extreme load of an enterprise software requires appropriate amount of hardware and software resources to run tool with bots, which simulate user activity. So let's count required resources from both theoretical and practical points of view.

To discover number of ports, required to run loading test bots, a black-box approach has been applied.

Using command

Listing 1. netstat

sudo netstat -natp | grep -e slave_port -e master_port -e backend_domain | wc – l

where slave_port and master_port are numbers specifying akka ports, which were used to execute slave and master instances, backend_domain is the targed url being tested.

It is possible to count number of ports which are in use by slave and master akka actor systems and by user websocket connections to backend_domain.

After slave is running on a computer, the netstat command outputs just one line, meaning that only one port is busy.

But when master running test-plan with 10 online users is executed netstat prints 14 lines: 2 akka ports in state of listening, 2 akka ports in established state and 10 websocket ports. The two akka ports in established state are used to transfer data between master and slave actors. Therefore it can be said, that running loadtest with N users using single slave requires P ports:

$$P = N + 2 + m \cdot 2$$

where m is a coefficient which takes value of one or zero, depending on if master actor is run or not on the same computer.

So P for N = 10 and m =1 equals to 14.

The tool also creates a number of threads. Every thread executes an actor instance actions simulating user behavior.

The tool is executing at Java virtual machine under a host operating system. So OS constrain the JVM by its limit of maximal thread number.

Modern desktop OSes support up to hundreds of thousands threads simultaneously. The actual limit for an OS can be obtained using:

Listing 2. Read limit for maximal number of threads in an OS

cat / proc / sys / kernel / threads –max

Practical experiment shown that executing test simulating 1000 online users leads to an out of thread limit exception.

References

1. "Apache jmeter official site," http://jmeter.apache.org/.
2. "Jmeter distributed testing step-by-step," http://jmeter.apache.org/usermanual/jmeter_distributed_testing_step_by_step.pdf.
3. Google, "Protocol buffers overview", https://developers.google.com/protocol-buffers/docs/overview.