Load Testing using Tsung

By Swaroop C H & Divyanshu

Helpshift Engineering Team Follow
May 1, 2014 · 5 min read

Background

Elasticsearch is one of the pillars of the Helpshift platform. We were adding a new feature related to live notifications for new issues, so we decided to use the new distributed percolator engine introduced in Elasticsearch 1.0. Before jumping into using percolators, we wanted to do load-testing to get an idea of the performance.

The best load-testing tool that our colleagues recommended was Tsung — it is an open source tool written in Erlang.

Note: Percolator engine feature is an "inverse" of a search engine. Usually, you register data and send search queries and you get back results containing data. In a percolator engine, you register queries and send data and you get back results containing queries. This is useful when you want to trigger events for new data that match queries of interest. For more information, see the Elasticsearch reference on percolators.

Installation

Installing Tsung on Linux is fairly straight-forward.

The installation steps we followed for Tsung version 1.5.0 are:

On Ubuntu Linux:

```
# Ubuntu's tsung package version is 1.4.2,
# so we have to compile it to get version 1.5.0
sudo apt-get install build-essential debhelper \
```

```
    erlang-nox erlang-dev \
    python-matplotlib gnuplot \
    libtemplate-perl
wget https://github.com/processone/tsung/archive/v1.5.0.tar.gz
tar -xvzf v1.5.0.tar.gz
cd tsung-1.5.0
./configure
make
make deb
cd ..
sudo dpkg -i tsung_1.5.0-1_all.deb
```

On Arch Linux:

```
sudo pacman -S --needed base-devel

# https://aur.archlinux.org/packages/tsung/
wget https://aur.archlinux.org/packages/ts/tsung/tsung.tar.gz
tar -xvzf tsung.tar.gz
cd tsung
makepkg -si

# Install dependencies for tsung_stats.pl
sudo pacman -S perl-template-toolkit
```

Run `tsung -v` to check that tsung is indeed installed.

Getting started with Tsung

Note: Do keep the Tsung user manual open for reference when you are trying this out.

Creating a basic Tsung load-test is just creating a XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE tsung SYSTEM "/usr/share/tsung/tsung-1.0.dtd" []>
<tsung loglevel="warning">

<clients>
    <client host="localhost" cpu="2" maxusers="30000000"/>
  </clients>

<servers>
    <server host="localhost" port="9200" type="tcp"/>
```

```
  </servers>

<load>
    <arrivalphase phase="1" duration="1" unit="minute">
      <users arrivalrate="5" unit="second"/>
    </arrivalphase>
  </load>

<sessions>
    <session name="es_load" weight="1" type="ts_http">
      <request>
      <http url="/fooindex/issue/_search"
              method="GET"
              contents_from_file="query.json" />
      </request>
    </session>
  </sessions>
</tsung>
```

If you want to sample the actual traffic being generated to make sure it looks correct, use `dumptraffic="true"` attribute on the top-level `tsung` tag, **but** do not use this for actual testing, because it slows down Tsung to a crawl.

Contents of `query.json` is:

```
{"size":10,"query":{"filtered":{"query":{"match_all":{}}}}}
```

Running the `tsung` is:

```
tsung -f basic.xml start
```

It takes some time to generate reports after the load is done, so be patient — for example, if you run the actual load-testing for 1 minute, then tsung finishes running in about 2.5 minutes.

You can watch it's progress by tailing the `tsung.log` file in the output directory it mentions when you start it:

```
$ tsung -f basic.xml start
Starting Tsung
"Log directory is: /home/helpshift/.tsung/log/20140430-1126"
# In another shell:
$ tail -f /home/helpshift/.tsung/log/20140430-1126/tsung.log
# stats: dump at 1398838905
stats: users 0 1
stats: session 46 9.011442764945654 20.867107440483167
148.7548828125 4.35986328125 0 0
stats: users_count 46 46
stats: finish_users_count 46 46
stats: request 44 7.090836958451704 20.332727527523975
140.344970703125 2.416015625 0 0
stats: page 44 7.090836958451704 20.332727527523975 140.344970703125
2.416015625 0 0
stats: connect 44 0.9098455255681818 0.21548106577396298
1.320068359375 0.364013671875 0 0
stats: size_rcv 9328 9328
stats: size_sent 13992 13992
stats: connected 0 0
stats: 201 44 44
#...
```

Once `tsung` finishes running, you can generate a report using:

```
mkdir basic_output
cd basic_output
/usr/lib/tsung/bin/tsung_stats.pl --stats
/home/helpshift/.tsung/log/20140430-1126/tsung.log
chromium graph.html  # Or your favorite browser
```

You should see some pretty graphs in the html page. Take some time to become familiar with them. Example graphs are:

Dynamic requests

If you want to generate some random input with each query, you can use variables.

For example, we wanted to register a lot of queries with the percolator engine under different document-ids:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE tsung SYSTEM "/usr/share/tsung/tsung-1.0.dtd" []>
<tsung loglevel="warning">

<clients>
    <client host="localhost" cpu="2" maxusers="30000000"/>
  </clients>

<servers>
    <server host="localhost" port="9200" type="tcp"/>
  </servers>

<load>
    <arrivalphase phase="1" duration="1" unit="minute">
      <users arrivalrate="5" unit="second"/>
    </arrivalphase>
  </load>

<sessions>
    <session name="es_load" weight="1" type="ts_http">
      <setdynvars sourcetype="random_string" length="20">
        <var name="docid"/>
      </setdynvars>
```

```
      <request subst="true">
        <http url="/fooindex/.percolator/%%_docid%%"
              method="PUT"
              contents_from_file="variable.json" />
      </request>
      </session>
    </sessions>
  </tsung>
```

The queries JSON file is:

```
{
  "query":
  {
    "filtered":
    {
      "filter":
      {
        "and": [
        {
          "term":
          {
            "tags": "ios"
          }
        },
        {
          "term":
          {
            "tags": "high_paying_customer"
          }
        }]
      },
      "query":
      {
        "match_all":
        {}
      }
    }
  }
}
```

As before, run tsung and generate stats file:

```
$ tsung -f variable.xml start
Starting Tsung
"Log directory is: /home/helpshift/.tsung/log/20140430-1151"

$ mkdir variable_output && cd variable_output
$ /usr/lib/tsung/bin/tsung_stats.pl --stats
/home/helpshift/.tsung/log/20140430-1151/tsung.log
chromium graph.html  # Or your favorite browser
```

Distributed Tsung

To run Tsung across machines, you have to:

1. Install Erlang and Tsung on all the machines — make sure all machines have the same versions of both Erlang and Tsung
2. Open ports for access in the machines to each other : port range 0–65535
3. Configure hostname on both machines for each other, because the tsung configuration demands host names and not ips, and those machines need to know how to talk to each other using those host names.

Now expand the client list in the tsung configuration xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE tsung SYSTEM "/usr/share/tsung/tsung-1.0.dtd" []>
<tsung loglevel="warning">

<clients>
    <client host="t1" cpu="2" maxusers="30000000"/>
    <client host="t2" cpu="2" maxusers="30000000"/>
  </clients>

<servers>
    <server host="localhost" port="9200" type="tcp"/>
  </servers>

<load>
    <arrivalphase phase="1" duration="1" unit="minute">
      <users arrivalrate="5" unit="second"/>
    </arrivalphase>
  </load>
```

```
    <sessions>
        <session name="es_load" weight="1" type="ts_http">
          <request>
          <http url="/fooindex/issue/_search"
                  method="GET"
                  contents_from_file="basic.json" />
          </request>
        </session>
      </sessions>
    </tsung>
```

Running and generating graphs is as same as before.

Additional Notes

If you want to generate more than one kind of query, then use sessions and specify multiple requests.

If you're generating a large number of requests, ensure that `maxusers` attribute of `client` is high (see above). Relatedly, ensure that the `ulimit` for file descriptors is high on the client machines as well.

If you want more flexibility in generating the body of the requests, then you will need to write Erlang code and use the `%%` sigil, example:

```
<sessions>
  <session name="app_event" weight="9" type="ts_http">
    <request name="start" subst="true">
    <http url="/some/api/"
            method="POST"
            contents="%%api_payloads:something_happened%%" />
    </request>
  </session>
</sessions>
```

Postscript

In the end, we realized that adding a lot of percolators has a performance hit on regular search queries, so we decided to have a separate percolator cluster where we only register our search queries and make percolator api calls, no data is stored. Having

separate clusters ensures the performance is not degraded and ensures scalability of speed for both searches and live notifications.

If you're interested in working with Erlang, Tsung, Elasticsearch and related topics, we are hiring, join us!

Elasticsearch Benchmark