Load Testing at Yandex

Alexey Lavrenuke
Load Testing at Yandex

What is Yandex
Yet another indexer
Yet another indexer
Yandex
Yandex’s mission is to help people discover new opportunities in their lives.
Yandex’s mission is to help people discover new opportunities in their lives.

Russian search engine

Started at 1997

Most popular search engine in Russia
Why do we need Load Testing
Once we accept our limits, we go beyond them.

Albert Einstein*

* some say Einstein didn’t say that but the expression is still beautiful
DevOps

Developers have never faced with performance problems before production

Sysadmins do not know service's architecture

Performance problems highlighted during load testing encourages people to work together solving them
Different kinds of problems

 › Stateless services. High throughput and velocity (banner system)

 › Scenario-based services (mail)

 › Different protocols (inter-service communication)

 › Batch systems (statistical services)
Load Testing at Yandex

Our main tool
Yandex.Tank: a ten-years history

Better software is produced by those forced to operate it*

phantom is a very fast web-server

phantom-benchmark is a plugin for it that acts as a client. And it is also fast

Yandex.Tank is built around phantom-benchmark. But today it is even more capable

* quoted from Theo Schlossnagle's "Operational Software Design" talk announce
Yandex.Tank today

- Yandex.Tank is an open source project
  - Primary language is **Python**
  - Default load generator is **phantom (C++)**
  - **JMeter** support
Yandex.Tank's internal architecture

Yandex.Tank is a meta-tool

- Tank provides common workflow for different load generators
- Generator should create sufficient load and measure response characteristics
- Tank has modular design
Load Testing at Yandex

Use Yandex.Tank
Load testing environment at Yandex

Cloud of tanks

Cloud of targets

A service that stores test results

DevOps, again: enable developers to perform load tests with minimum efforts
Use cases

Stateless: phantom + Yandex.Tank

Scenario-based: JMeter + Yandex.Tank

Different protocols: JMeter, phantom in some cases or custom solutions + Yandex.Tank

Batch systems: not using Yandex.Tank. Will discuss them later
How to install it

Yandex.Tank

Phantom

Jmeter
How to install it

Ubuntu PPA
Python package

Phantom
Jmeter

Yandex.Tank
How to install it

Ubuntu PPA
Python package

Yandex.Tank

Ubuntu PPA
From sources

Phantom

Jmeter
How to install it

- Ubuntu PPA
- Python package
- Yandex.Tank
- Phantom
- Jmeter

Ubuntu PPA
From sources
Any way you want
How to install it

- Ubuntu PPA
- Python package
- Jmeter
- Yandex.Tank
- Docker repository
- Phantom
- Ubuntu PPA
- From sources
- Any way you want
Configure your first load test

› .ini files
› good defaults
› redefine defaults on multiple levels

it is rather easy to use Yandex.Tank to make automated load tests
Configure your first load test

› .ini files
› good defaults
› redefine defaults on multiple levels

it is rather easy to use Yandex.Tank to make automated load tests
Configuration example, load.ini

Section header: for each plugin

[phantom]
Choose target

Target address: IP, IPv6 or domain name

address = my.service.com
What do we send

Ammo in one of possible formats

```python
[phantom]
address = my.service.com
uris = /
    /mypage.html
    /click/page?data=hello
headers = [Host: example.org]
    [Accept-Encoding: gzip, deflate]
```
Load type

Add schedule

[phantom]
address = my.service.com
uris = /
    /mypage.html
    /click/page?data=hello
headers = [Host: example.org]
    [Accept-Encoding: gzip, deflate]
rps_schedule = const(1, 40s)
First test

Save config as load.ini and shoot

```
yandex-tank -c ./load.ini
```
Load Testing at Yandex

Test types
**Open and closed systems**

Closed systems have negative feedback that makes it impossible to "bury" the service. Users wait for the responses before making new requests.

No negative feedback in **open** system. Internet is an open system.
Finding max performance

Closed model, gradually raise thread count
- each thread sends requests one-by-one

yandex-tank -c ./load.ini
- o "phantom.instances_schedule=line(1, 40, 10m)"
- o "phantom.rps_schedule="
Times distribution graph
Times distribution graph

- RPS
- Times distribution (each second)
- Thread count (schedule)
Finding max performance on times dist graph

response times distribution graph
Finding max performance on times dist graph

response times distribution graph
Finding max performance on times dist graph

response times distribution graph
Finding max performance on times dist graph

top performance

linear growth

response times distribution graph
Finding max performance on times dist graph

- Linear growth
- Top performance
- Degraded!
Finding point of failure

Open system. Hard schedule

- emulate open system with multitude of threads

```
yandex-tank -c ./load.ini
   -o "phantom.rps_schema=range(1, 1000, 10m)"
   -o "phantom.instances=10000"
```
Response times quantiles graph
Response times quantiles graph

load schedule (expected RPS)

quantiles

response times
Finding point of failure on RT quantile graph

response times quantiles
Finding point of failure on RT quantile graph

imbalance started

response times quantiles
Finding point of failure on RT quantile graph

imbalance started

RPS of failure

response times quantiles
Measuring response times

Open system, constant load. Load level from SLA or from previous tests

- don't forget about warming up

`yandex-tank -c ./load.ini
- o "phantom.rps_schedule=line(1, 300, 30s) const(300, 5m)"
- o "phantom.instances=10000"`
Mind your spikes

spikes on quantile graph
Mind your spikes

spikes on quantile graph
Common spikes reasons

"Heavy" requests in your ammo. See if spikes become more often on higher load

Periodical processes on your server. Cron job or cache synchronization. Garbage collector

Someone else queries your server periodically
Investigating reasons for spikes

Service downloads something periodically:

- memory consumption
- network send/receive
Investigating reasons for spikes

Service downloads something periodically:

memory consumption

network send/receive
Finding leaking resources

Open system, constant load, long period

set your load level at 80-90% from maximum level you've found before

```
yandex-tank -c ./load.ini
  -o "phantom.rps_schedule=line(1, 700, 30s) const(700, 1h)"
  -o "phantom.instances=10000"
```
Testing methodology

› Smoke test. Ensure your system is working and you have all the metrics needed
› Performance test. Closed system, growing number of threads
› Imbalance test. Open system, hard schedule, linear growth
› Measuring timings. Open system, constant load level from SLA
› Find leaking resources. High load level.
› Any other test you need.
Load Testing at Yandex

Approaching batch systems
Machine learning system

Ordinary people

An interesting person

Event log

The machine
codependent filters with history

Mark relevant events
Machine learning system

Mark relevant events
Machine learning system

Mark relevant events

Billing

Statistics

Targeting
Living with a batch system

Millions of events in one log. Thus, no RPS

Log sizes varied over the day. Can't change load level. Can't make it "flat" or linear

Can't use Yandex.Tank or any other load generator. Still want to see performance limits and trends
Why having X2 server is never enough

Assumption: if we have an X2 server and it works then we have enough time to do something before we reach our performance limit in production

But how much time do we have and what exactly should we do?
First step: collecting metrics

We collected different kind of metrics from all our servers.

Some of them were useful in understanding the results of experiments

But they work only if the change is instant and big enough
Problem: too many metrics

There are a lot of metrics. Really a lot.

And we don't know for sure which of them are the reasons and which are the consequences.

Lower the number of dimensions

\[(y_1, \cdots, y_n) = F(x_1, \cdots, x_n)\]
Solution: scatter plots

Uncorrelated

Linear dependency
What scatter plot can tell us

Non-zero process time for zero-sized logs

Non-linear tail

Outliers
Compare observations
Compare observations

Test vs Prod

New release vs current

Period in the past with today
Find trends

Basic idea: build linear model on each release and compare the coefficients

But the data is too noisy and the model is unstable
Making it better

Clean the dataset by using density-based clusterization

Use points from the biggest cluster to build the linear model

Investigate the reasons for outliers and smaller clusters
Dig deeper: components dependencies

Ordinary people → Event log → The machine

Event log: codependent filters with history

Mark relevant events

An interesting person
Dig deeper: components dependencies

Ordinary people

An interesting person

Event log

codependent filters with history

Mark relevant events
Dig deeper: components dependencies

Investigated component dependencies

Extracted data flow from code

Converted them into graph diagram

Found critical path: the longest path in that graph
Critical path visualization

Collected data about each component work times and wait times for each processed log

Visualize them on critical path

Now we can find bottlenecks and see if they migrate in new releases
Testing the batch system

› Learn about the architecture

› Collect a lot of metrics. Write tools to collect additional metrics

› Find correlations

› Automate trend detection

› Find the critical path

› Investigate outliers
Load Testing at Yandex

Summary, links and contacts
What did we learn today

› Yandex.Tank: a universal load testing tool
› Load testing methodology
› How to approach batch systems
Useful links

**Our chat room:**  [gitter.im/yandex/yandex-tank](https://gitter.im/yandex/yandex-tank)

About Yandex.Tank project:  [yandex.github.io/yandex-tank](https://yandex.github.io/yandex-tank)

Yandex.Tank on github:  [github.com/yandex/yandex-tank](https://github.com/yandex/yandex-tank)

Yandex Tank API on github:  [github.com/yandex-load/yandex-tank-api](https://github.com/yandex-load/yandex-tank-api)

phantom on github:  [github.com/mamchits/phantom](https://github.com/mamchits/phantom)

Read the docs on ReadTheDocs:  [yandextank.readthedocs.org](https://yandextank.readthedocs.org)
Contacts

Alexey Lavrenuuke
testing engineer

direvius@yandex-team.ru
@direvius, #yandextank
Let’s go beyond our limits!