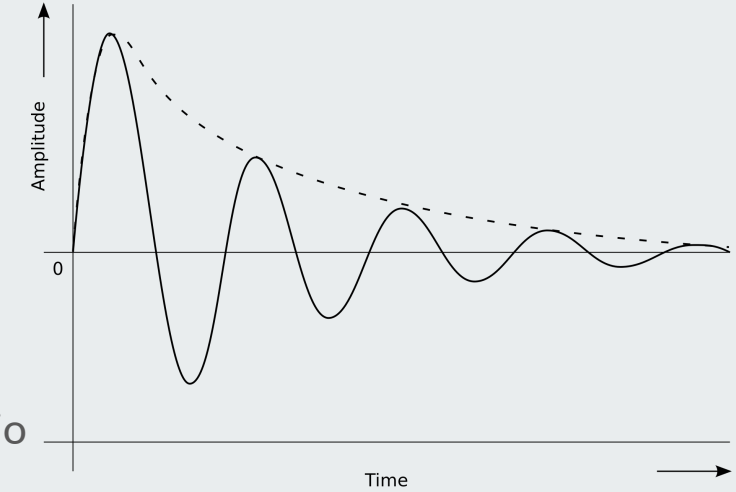
Playful Load Testing with Locust



Load Testing/Performance Testing with helps of Locust.io

Slides available at tinyurl.com/pyconid-locust
Codes available at github.com/mappuji/simple-todo-api

About me / my projects / my background

- Engineering Physics Graduate
- Context Driven School of Testing (github.com/Pendapa/buccaneer-tester)
- Software Development Engineer in Test at midtrans (midtrans.com)
- Open Source Contributor at Open Learning Exchange (github.com/ole-vi)
- Found me in
  - github.com/mappuji
  - BLOG mappuji.org

Disclaimer

The views or opinions expressed by me is solely my own and do not necessarily represent the view or opinions of PT. Midtrans

Load Test in a Nutshell

In what load the system will fails?

How the system fails?

Terminology

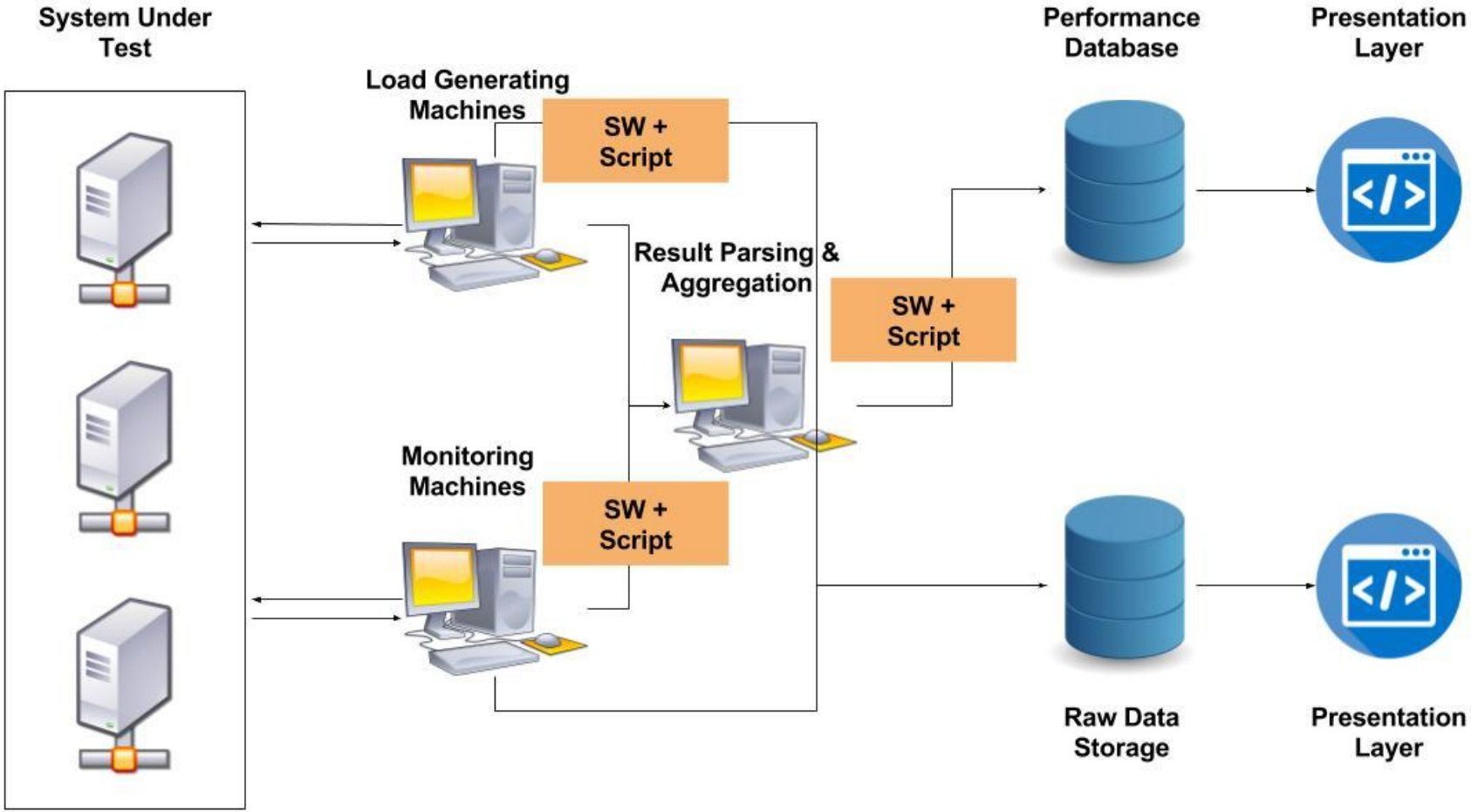
- ***Load Test*** – with certain load find out in over period of time what will the system behaves?
- ***Performance/Reliability Testing*** – how performant our system is? Usually involves load test.
- ***Performance Profiling*** – profiling performance in section/block of app building block.
- ***Benchmark Test*** – test an app with some benchmark and running it in every release/builds. (I currently develop the tool github.com/demingio)
- ***Scalability Test*** – if the resource increased, will the throughput increasing? How much?
- ***Availability Testing*** – tight to reliability, when the system fails how quick the system backup?

When to do Performance/Load Testing

- Is my application can support what I think customer want?
- Data to backup sales & marketing or legal & certification if any.
- Hardware upgrades. Is the upgrade impact positively to performance? How much? Which manufacturer better?
- Performance test ≠ functionality test. It should done after functionality test.

Test Setup

1. System Under Test
2. Load generating machine with load generating software (***with proper use case scenarios***)
3. Monitoring machine with monitoring software
4. Data repository (detailed summary)
5. Data access and display


System Under Test
Load Generating Machines
SW + Script
Result Parsing & Aggregation
SW + Script
Monitoring Machines
SW + Script
Performance Database
Presentation Layer
Raw Data Storage
Presentation Layer

In House Tools

- Advantages
  - Frequently available as a side effect of development process
  - Easy to use (expertise exist in house)
  - Source code available - tools can be extended as necessary
- Disadvantages
  - Can not be shared with customers
  - Validity of results questionable outside of the company
  - Tool maintenance can be costly and tedious
  - Sometimes just create *yet another scripting tool*

Vendor Tools

- Advantages
  - Support many different protocols
  - Professionally developed and maintained
  - Results and scripts can be shared and verified
- Disadvantages
  - Extremely expensive
  - Proprietary scripting language
  - No access to source code for modifications | No money no feature
  - Put pretty graph but meaningless

Open Source Tools

- Advantages
  - Good price (purchase and maintenance)
  - Easy to share results with customers and or publications
  - Source code available - tools can be extended if necessary
  - Scripting in standard programming language
- Disadvantages
  - Support for different protocols limited
  - Steeper learning curve

Open Source Tools Comparison

Criteria	JMeter	Locust
Operating System	Any	Any
Open Source	Yes	Yes
GUI	Yes, with non-GUI mode available	No
Execution Monitoring	Console, File, Graphs. Desktop client, Custom plugins	Console, Web
Support of “Test as Code”	Weak (Java)	Strong (Python)
In-built Protocols Support	HTTP, FTP, JDBC, SOAP, LDAP, TCP, JMS, SMTP, POP3, IMAP	HTTP

Open Source Tools Comparison (cont.)

Criteria	JMeter	Locust
Integrated Host Monitoring	PerfMon	No
Recording Functionality	Yes	No
Distributed Execution	Yes	Yes
Easy to use with VCS	No	Yes
Resources Consumption	More resources required	Less resources required
Number of Concurrent Users	Thousands, under restrictions	Thousands
Ramp-up Flexibility	Yes	No
Test Results Analyzing	Yes	Yes

When Locust.io fits your need

- Testing HTTP app
- High throughput less resource load generating machine.
- Distributed load generating machine
- Complex use case scenarios.
- Test as code

Locust.io in action

```
> pip install locustio

> touch todo_app_load.py
```

```python
# todo_app_load.py
from locust import HttpLocust, TaskSet, task
import uuid


class UserBehavior(TaskSet):
    @task(10)
    def get_tasks(self):
        self.client.get("/tasks", name="[get] /tasks")


    @task(1)
    def post_tasks(self):
        headers = {'content-type': 'application/x-www-form-urlencoded'}
        task_name = "PyCon" + str(uuid.uuid4())
        payload = {'name': task_name}
        self.client.post("/tasks", headers=headers, data=payload, name="[post] /tasks")


class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    min_wait = 900
    max_wait = 1000
```

Locust.io in action (cont)

```
> locust -f todo_app_load.py
--host=http://mappuji.org:3000

> #open browser localhost:8089

> #it will running single process
```

Locust.io in action (cont)

- **Number of user** – number of simulated user that will run the use case scenarios
- **Hatch rate** – rate of user hatching
- **MIN_WAIT** – minimum wait time before new request
- **MAX_WAIT** - maximum wait time before new request
- **If response time** > **MAX_WAIT** the system will wait until response received or timeout
- Locust use Python Requests library that by default has no timeout, so we should put time out explicitly
- How if we want to do request asynchronously, and not waiting for response to come in? Use greenlet!

```python
# todo_app_load_greenlet.py
from locust import HttpLocust, TaskSet, task
import uuid
import gevent

class UserBehavior(TaskSet):
    def _get_tasks(self):
        self.client.get("/tasks", name="[get] /tasks")
    @task(10)
    def get_tasks_async(self):
        gevent.spawn(self._get_tasks)
    def _post_tasks(self):
        headers = {'content-type': 'application/x-www-form-urlencoded'}
        task_name = "PyCon" + str(uuid.uuid4())
        payload = {'name': task_name}
        self.client.post("/tasks", headers=headers, data=payload, name="[post] /tasks")
    @task(1)
    def post_tasks_async(self):
        gevent.spawn(self._post_tasks)

class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    min_wait = 900
    max_wait = 1000
```

Locust.io in action (cont)

```
> locust -f todo_app_load_greenlet.py --master
--host=http://mappuji.org:3000

> locust -f todo_app_load_greenlet.py --slave
--host=http://mappuji.org:3000

> #open browser localhost:8089

> #it will running two processes
```

Things Outside Locust

- It is fact that locust reporting and test analysis is limited, but performance/load test produce lots of data.
- To better extract useful information we should record
  - App Logs
  - App Monitoring record: exception, error rate, success rate, etc
  - Resource utilization: CPU, Memory, I/O, Network, etc
- If possible record data in database

Summary

- Performance/Load Testing is important to find performance bottlenecks for system under test or softwares.
- There is several options for test tools including in-house, vendor, and open source tools.
- Open source tools has fairly good cost and other advantages.
- Locust is one of the performance testing tools that excels in high throughput less resource load generating machine with strong flexibility with full programming language support (Python)
- Indeed there is part of testing setup outside Locust, testing best done with proper setup, execution and analysis.

References

- Goranka Bjedov - Using open source tools for performance testing https://www.youtube.com/watch?v=335LKIXRauA
- Jad Meouchy - Load testing your node application https://www.youtube.com/watch?v=j7a-7cUvQ0c
- Blaze Meter - JMeter vs. Locust - Which One Should You Choose? https://loadstorm.com/2010/01/load-testing-vs-performance-testing/
- Locust Documentation - https://docs.locust.io/en/latest/
- pglass - How do I locust - https://github.com/pglass/how-do-i-locust

```
if questions:
 try:
   answer()
 except RuntimeError:
   pass
else
 print('Thank You!')
```

Bonus Slide: Starting Performance Test

1. Set up a realistic environment
    - Results from a small test environment cannot be simply extrapolated to predict behavior of production environment. This includes hardware and software (database in particular)
    - Modeling and profiling can help with extrapolation, but this is not always reliable

Bonus Slide: Starting Performance Test (cont.)

2. Do stress test
    - How is your system failing?
    - Find various failing
    - In unix machine don't go above 80%
    - Set performance test point

Bonus Slide: Starting Performance Test (cont.)

3. Execute performance test
    - Test time should long enough to reach steady state
    - Start each test with a reasonable "warm-up" period where the load is increasing linearly from none to maximum performance load (reasonable time 15-30 minutes)
    - Collect throughput and latency data during test execution time
    - Collect server performance data during test execution
    - System performance is the lowest throughput during test time
    - Repeat the test several times to assure statistical consistency