DISTRIBUTED TESTING BY FUNKLOAD

Team 26
Project 6

FUNKLOAD

- Funkload is a functional and stress test tool that can be used on your web applications.
- Stress tests are implemented as PyUnit tests, so they can also be used as functional tests
- Funkload provides nice reporting features out of the box: you can do trends and diff reports - You can monitor the server being tested
- Performance testing: by loading the web application and monitoring your servers it helps you to pinpoint bottlenecks, giving a detailed report of performance measurement.

A SIMPLE TEST-CASE MODEL IN FUNKLOAD

```
import unittest
from  random  import random
from  funkload.FunkLoadTestCase  import FunkLoadTestCase

class  Simple(FunkLoadTestCase):
        def  setUp(self):
              self.server_url  =  self.conf_get('main', 'url')
        def  test_simple(self):
              server_url  =  self.server_url  res  =  self.get(server_url,  description='Get url')
              self.assertEqual(res.code, 200)
              self.assertEqual(res.body, "Hello World")

 if __name__ in ('main', '__main__'):
        unittest.main()
```

BENCH CONFIGURATION.

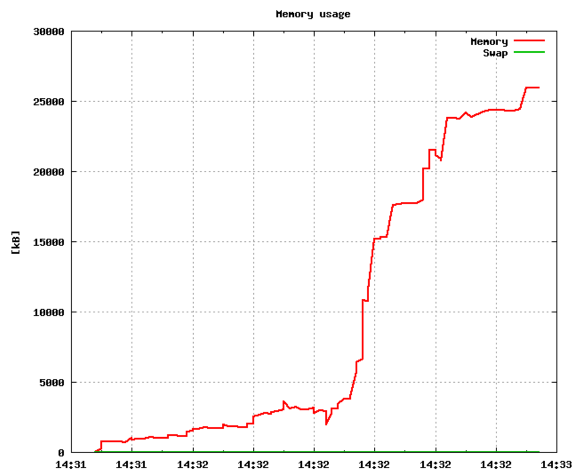
- The testing can be performed for various bench configurations.

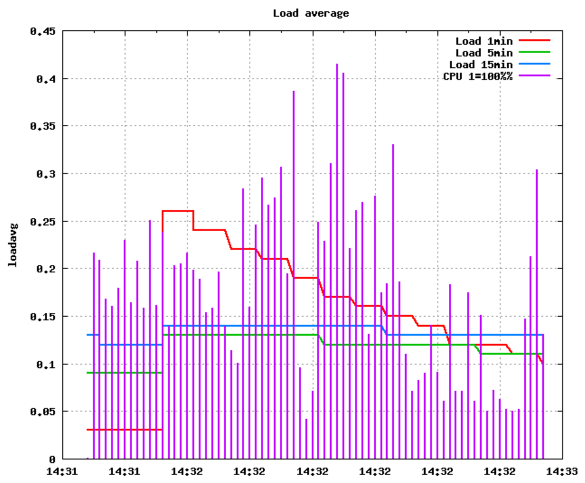
- The following tests are performed for these configurations.
  - Cycles of concurrent users: [1, 10, 20]
  - Cycle duration: 10s
  - Sleep time between requests: from 0.0s to 0.5s
  - Sleep time between test cases: 0.01s
  - Startup delay between threads: 0.01s

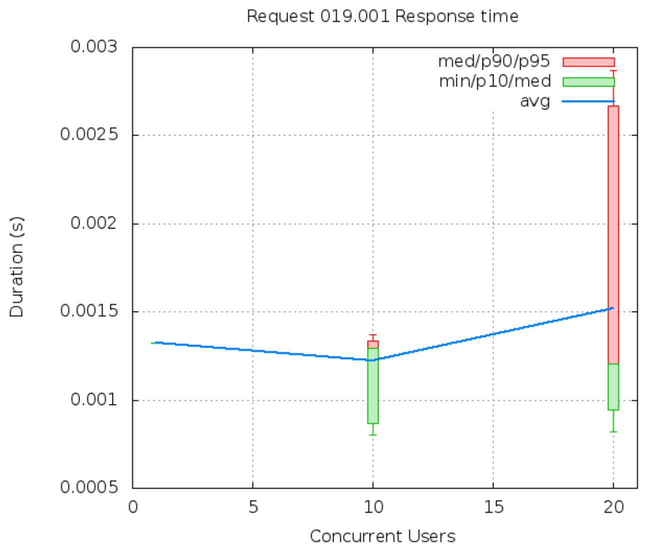
DISTRIBUTED TESTING

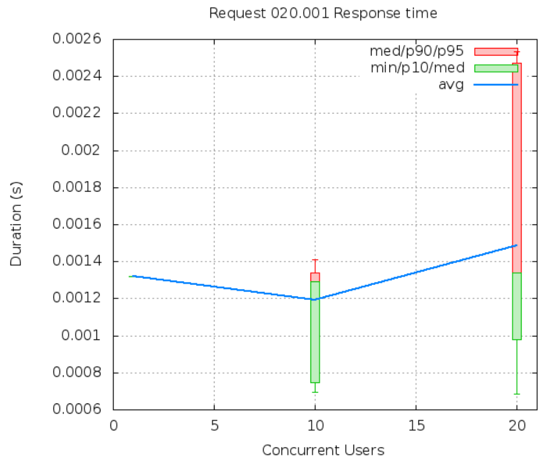
- Funkload has a built-in support to distribute the charge amongst several nodes.
- Funkload achieves this distributed testing by driving the nodes via SSH using Paramiko.
- Distributed bench testing:
  - fl-run-bench --distribute --distribute-workers=node1,node2 test_case.py
- Building Reports:
  - fl-build-report --html -o html distributed-simple-test.log/node1-simple-bench.xml
- Funkload with let you monitor:
  - fl-monitor-ctl monitor.conf start

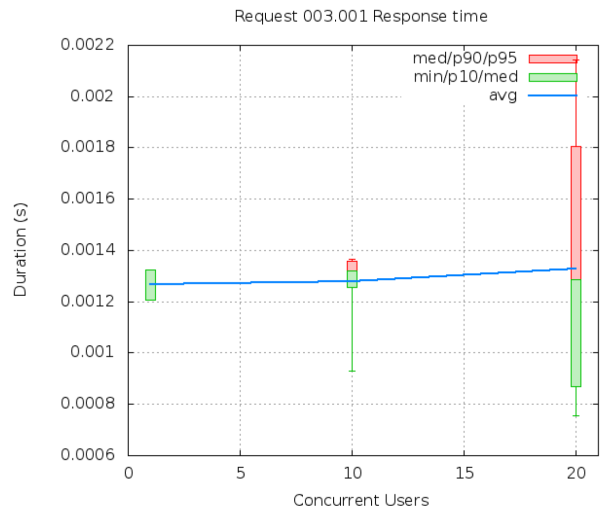
APACHE SERVER TESTING RESULTS

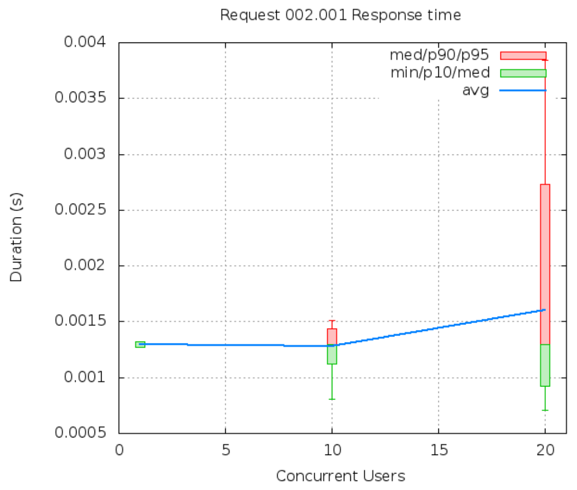


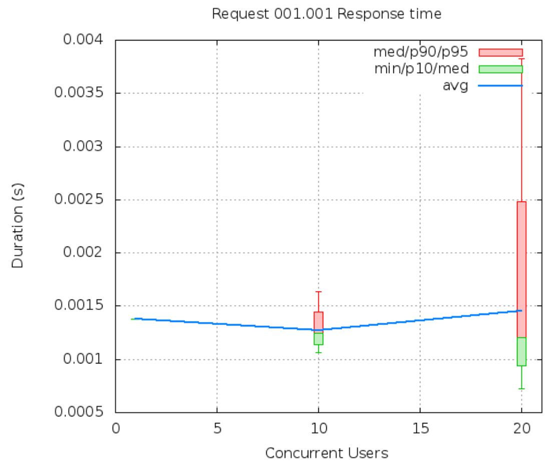


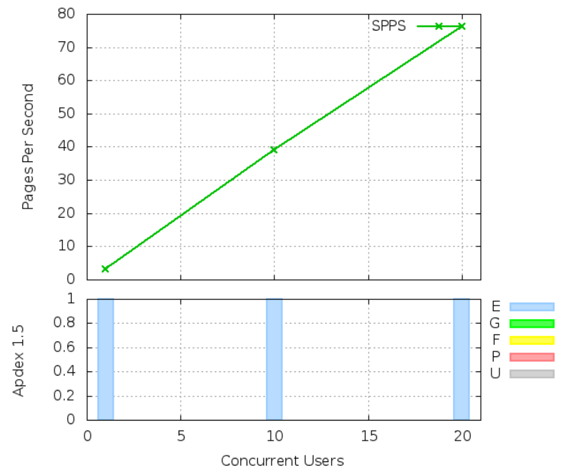


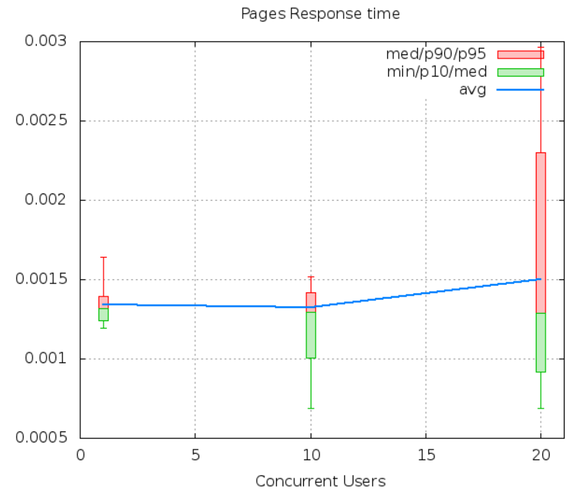


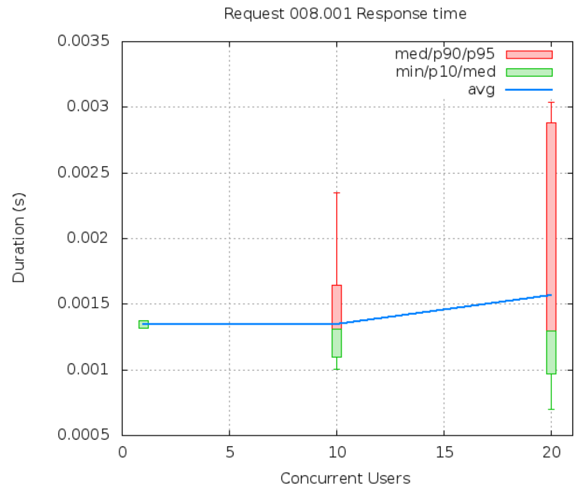

Request 003.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.0022
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0
5
10
15
20
Concurrent Users

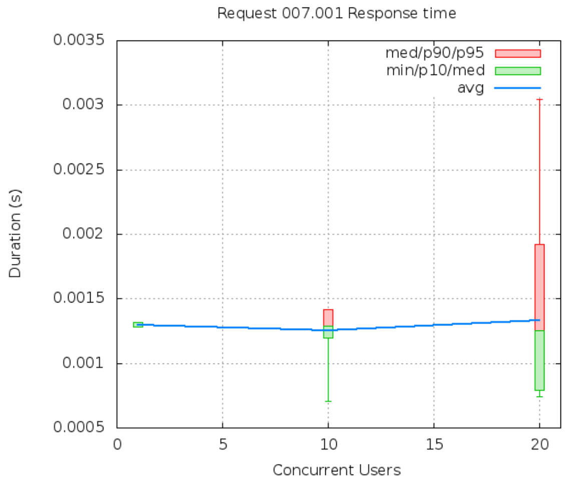

Request 002.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20
Concurrent Users

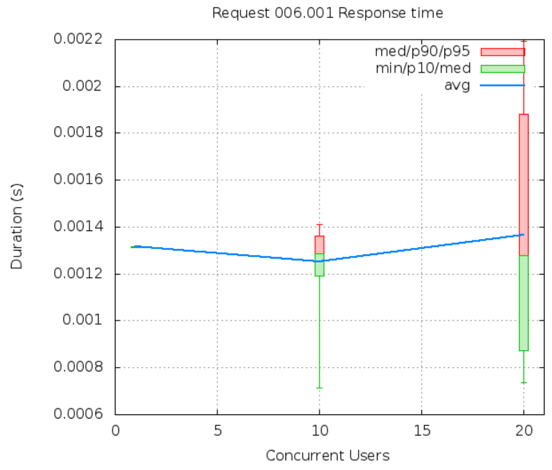

Request 001.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20
Concurrent Users

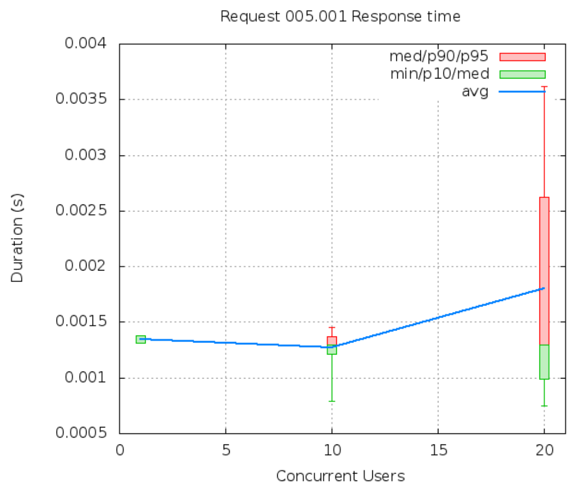

SPPS
Pages Per Second
80
70
60
50
40
30
20
10
0
Apdex 1.5
1
0.8
0.6
0.4
0.2
0
E
G
F
P
U
0
5
10
15
20
Concurrent Users

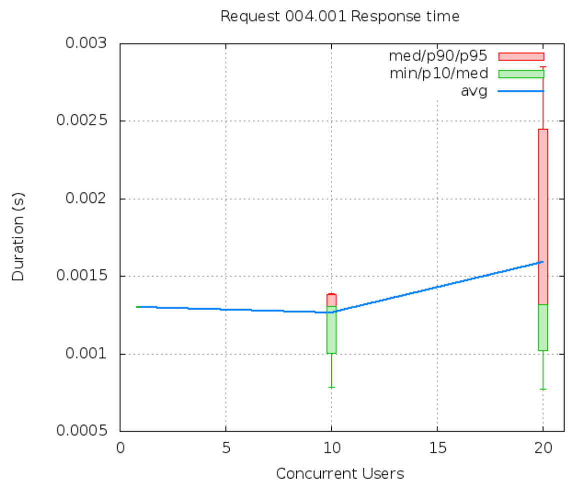

Pages Response time
med/p90/p95
min/p10/med
avg
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20
Concurrent Users

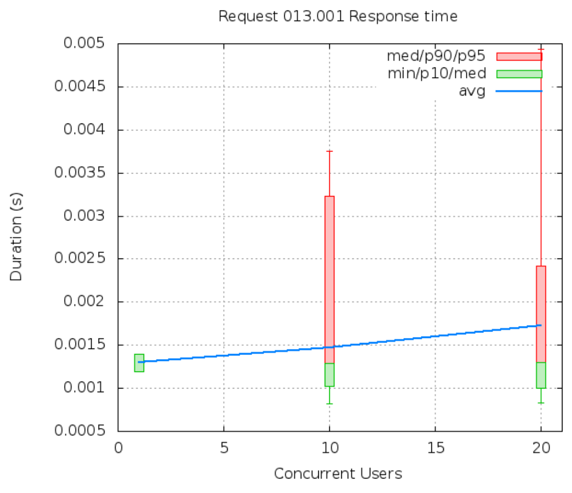

Request 008.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20
Concurrent Users

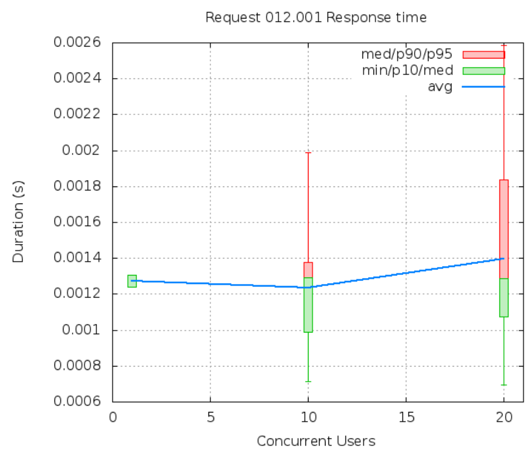

Request 007.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20
Concurrent Users

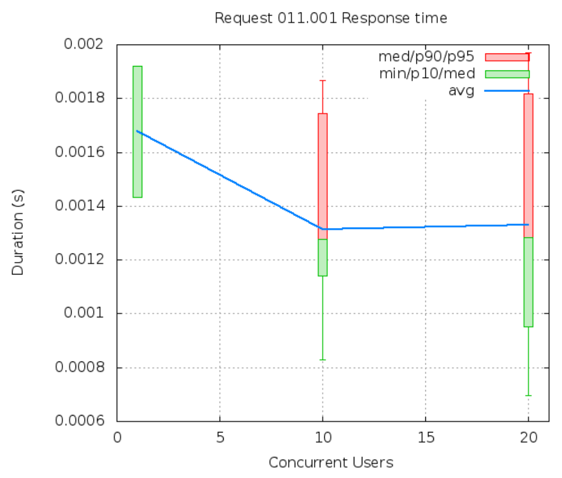

Request 006.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.0022
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0
5
10
15
20
Concurrent Users

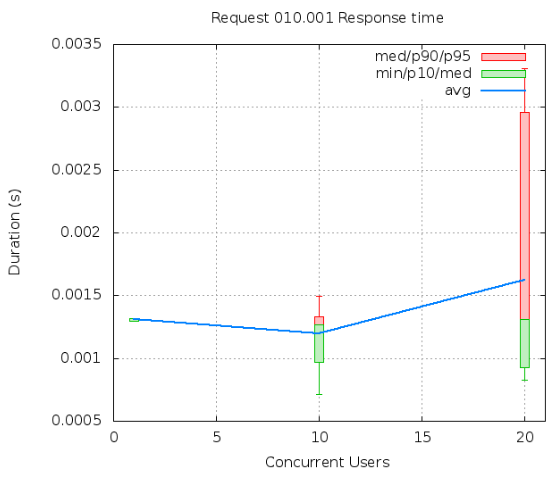

Request 005.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20

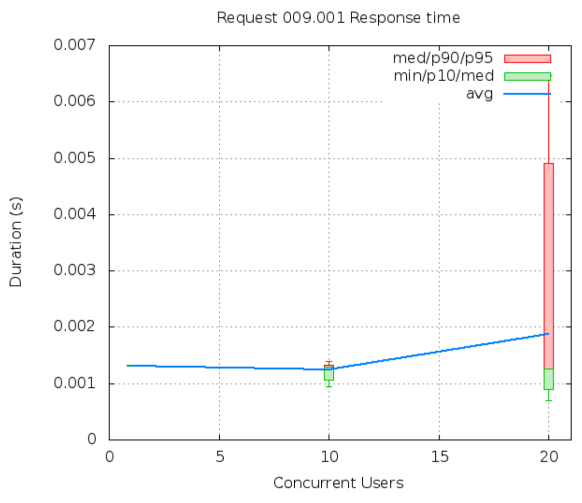

Request 004.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20

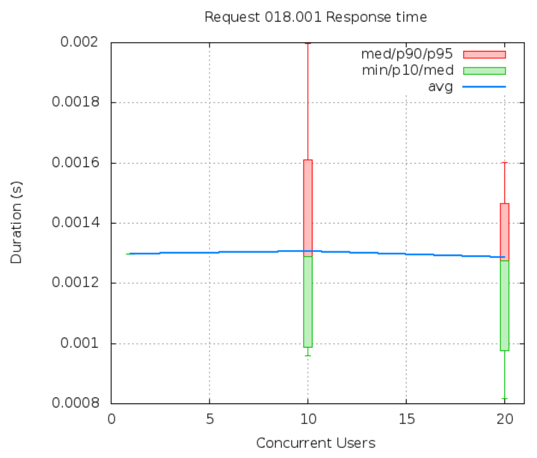

Request 013.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.005
0.0045
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20

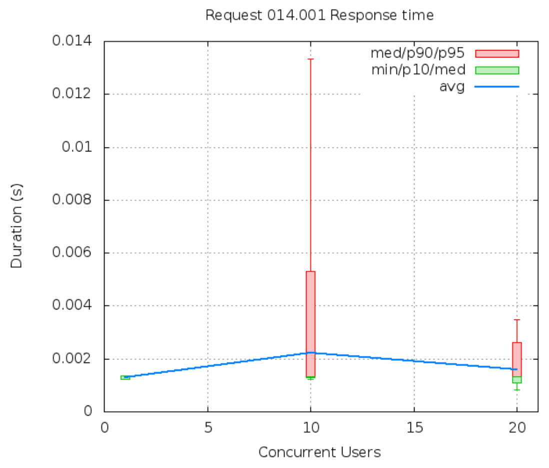

Request 012.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.0026
0.0024
0.0022
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0
5
10
15
20

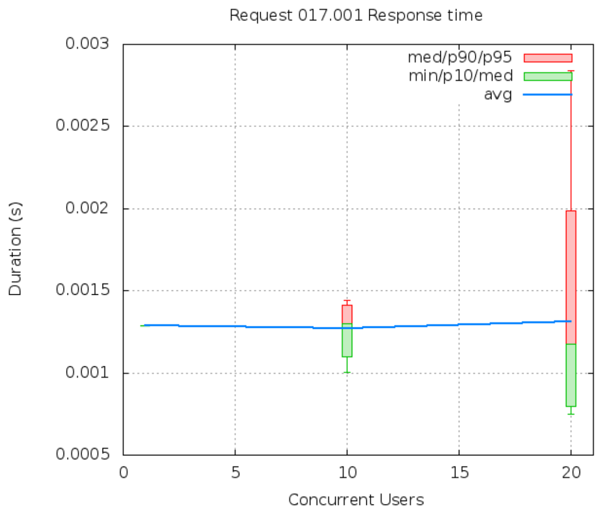

Request 011.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0
5
10
15
20

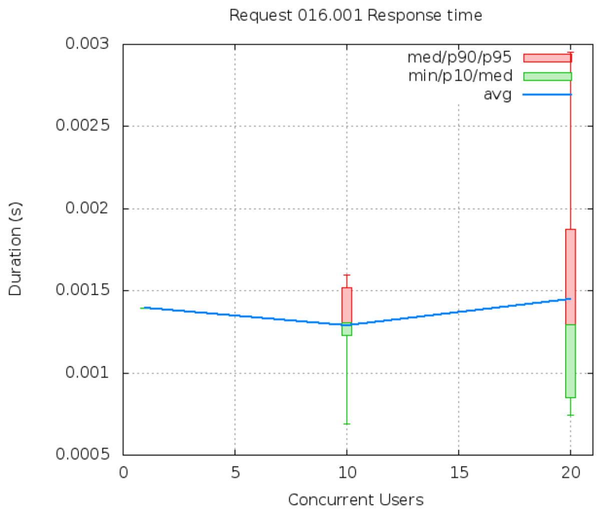

Request 010.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20

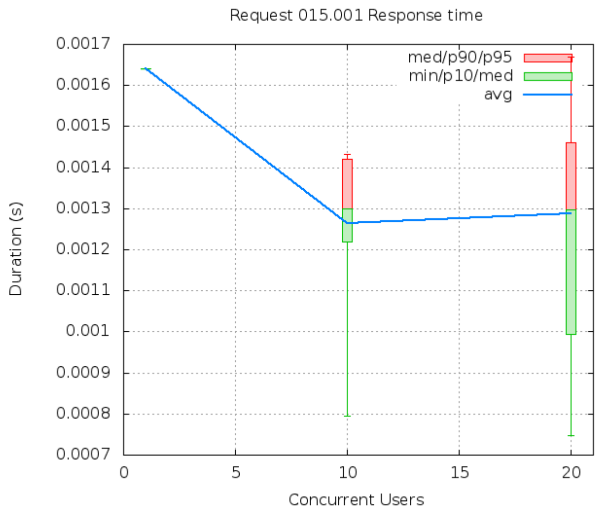

Request 009.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
5
10
15
20

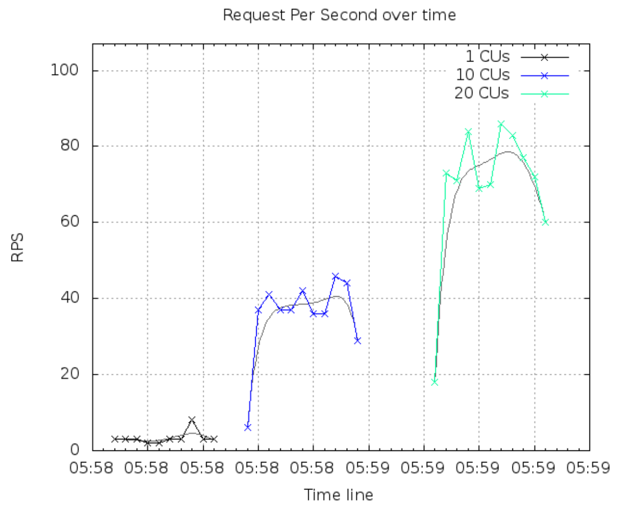

Request 018.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0
5
10
15
20

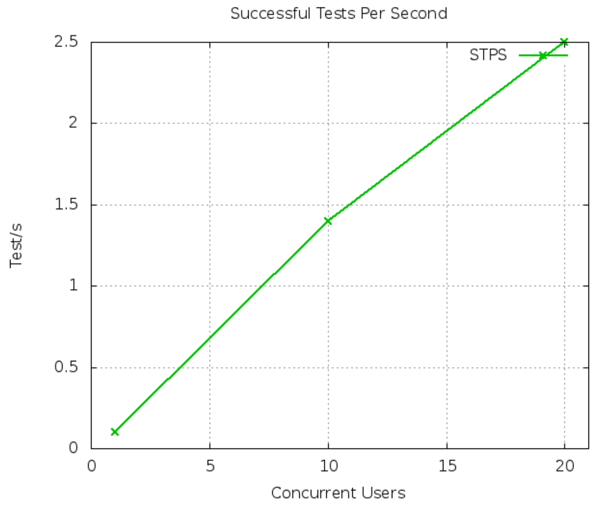

Request 014.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

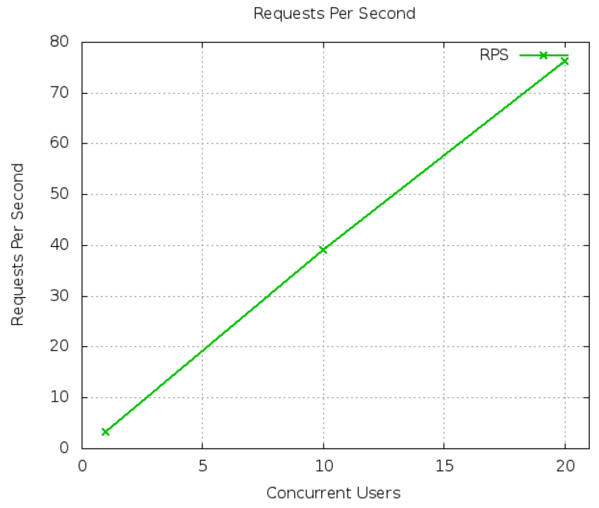

Request 017.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

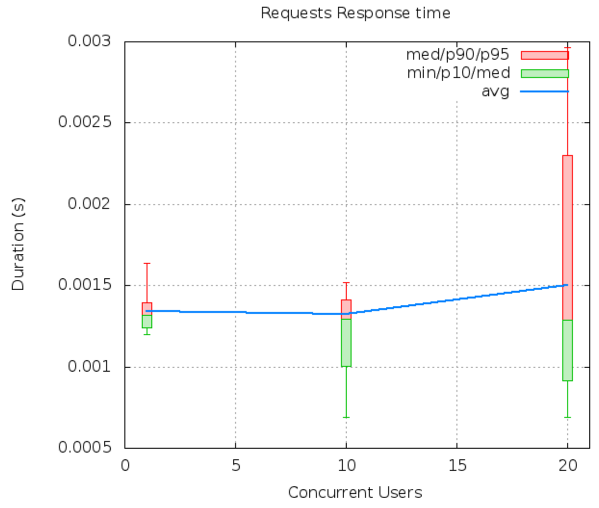

Request 016.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users


Request 015.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

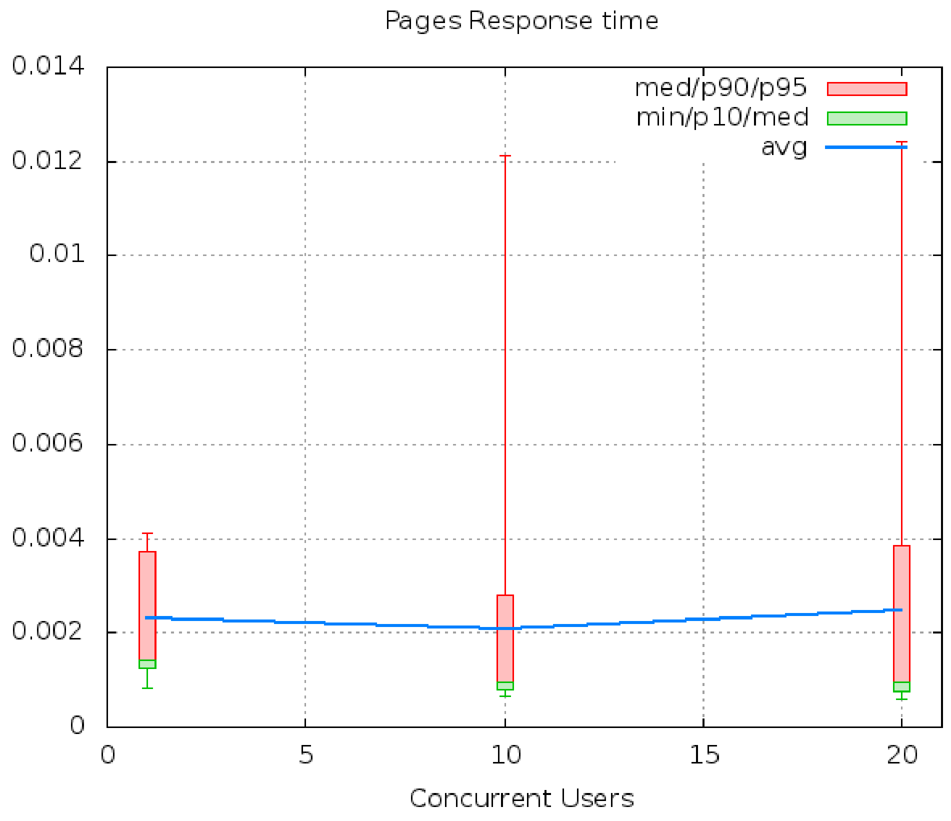

Request Per Second over time
1 CUs
10 CUs
20 CUs
RPS
Time line

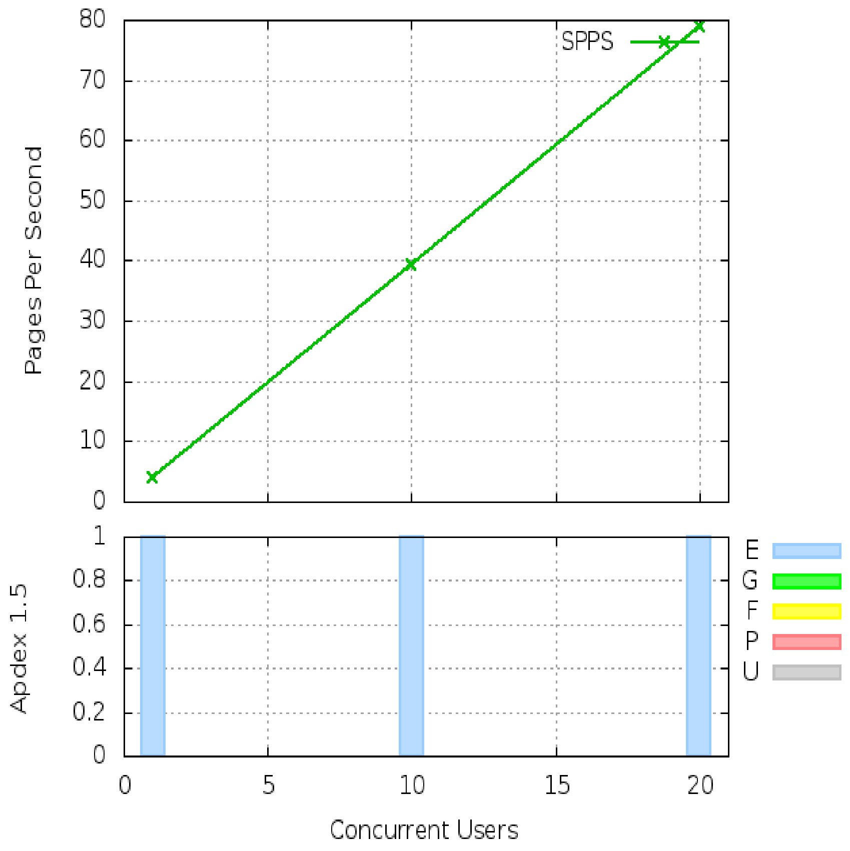

Successful Tests Per Second
STPS
Test/s
Concurrent Users

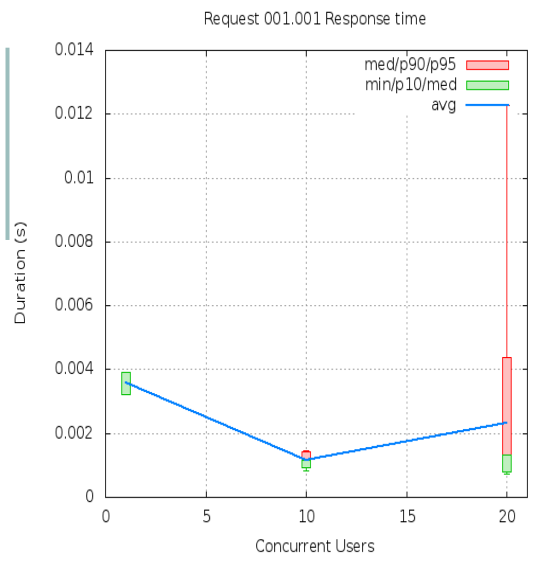

Requests Per Second
RPS
Requests Per Second
Concurrent Users

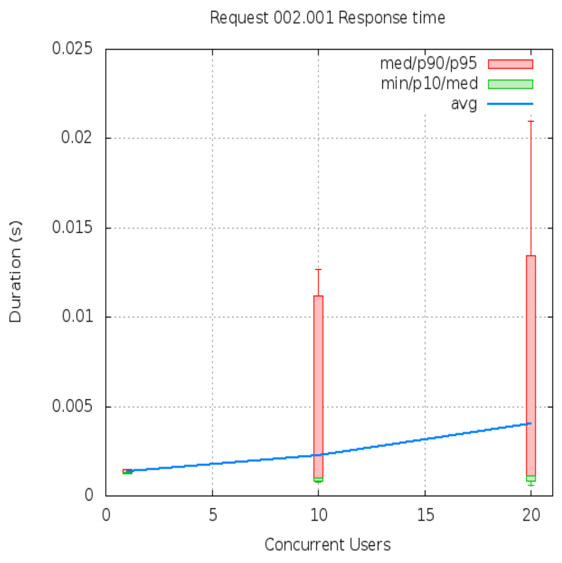

Requests Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

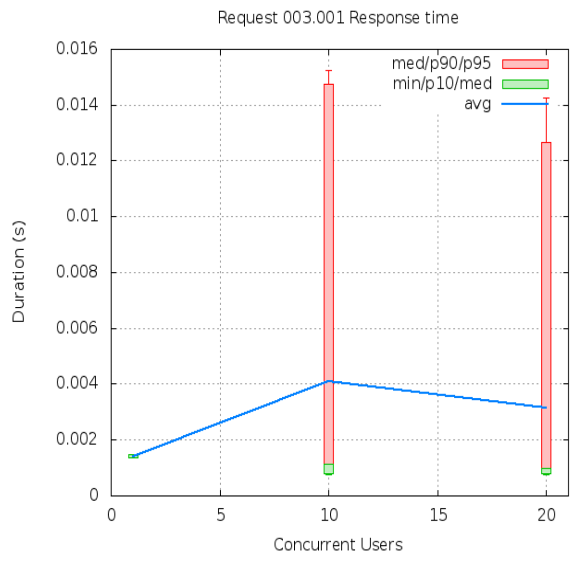
NGINX SERVER TESTING RESULTS

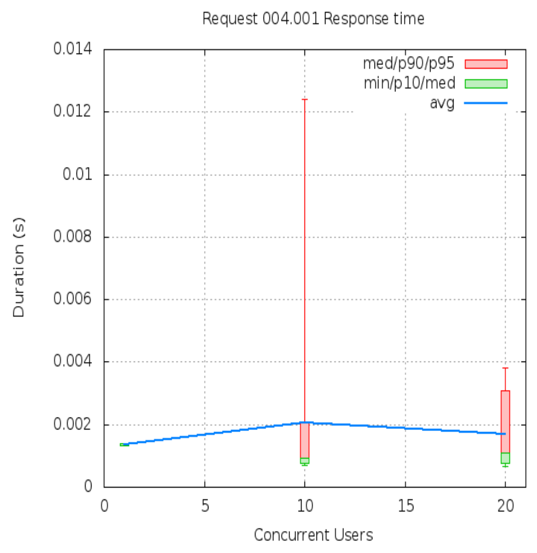


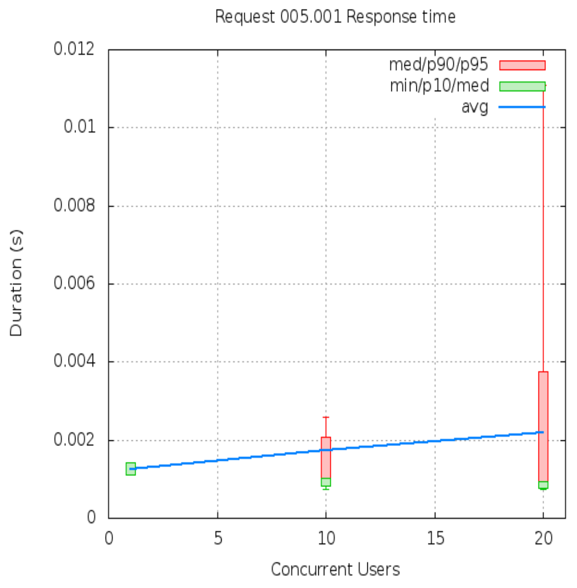


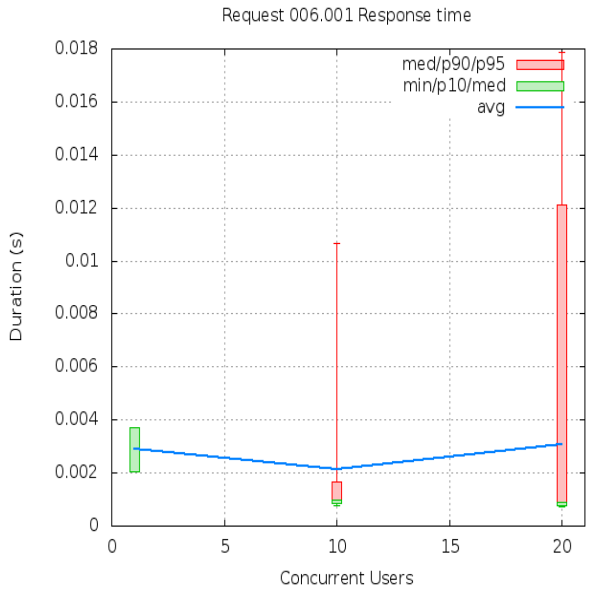

Request 001.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
5
10
15
20
Concurrent Users

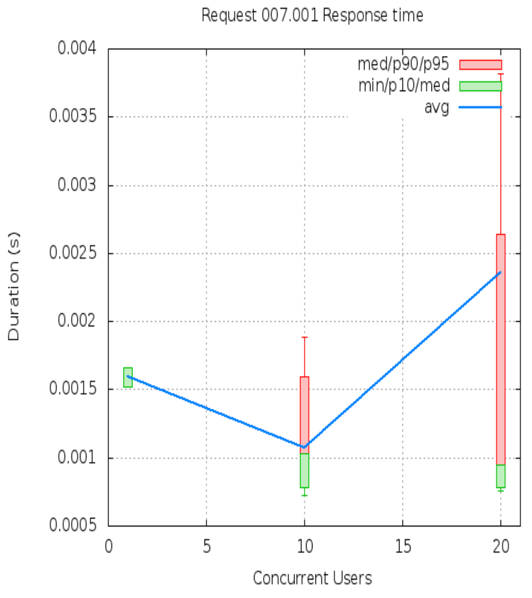

Request 002.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.025
0.02
0.015
0.01
0.005
0
0
5
10
15
20
Concurrent Users

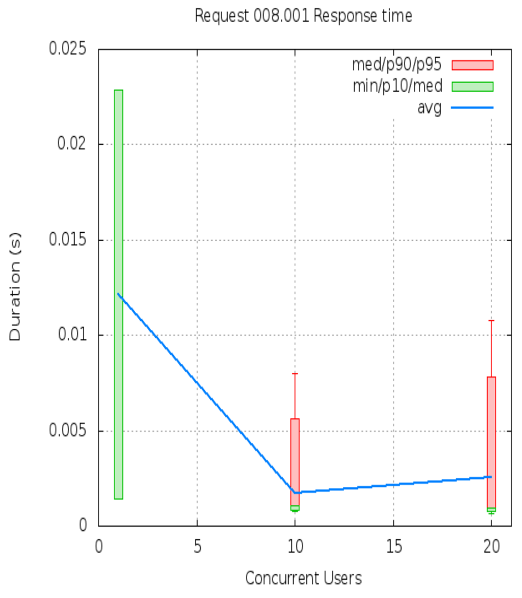

Request 003.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
5
10
15
20
Concurrent Users

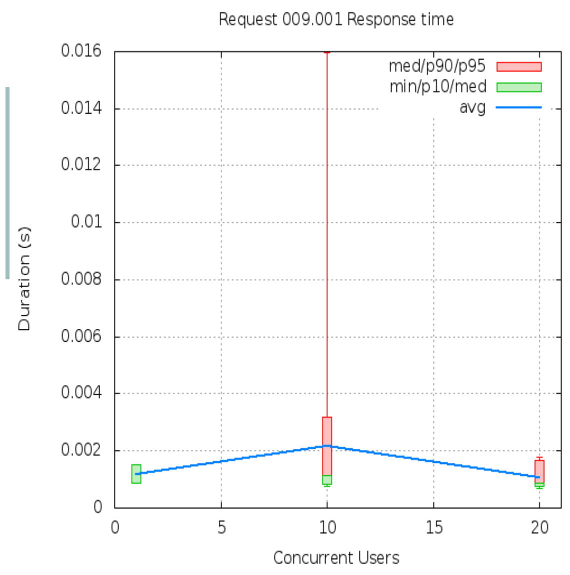

Request 004.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
5
10
15
20
Concurrent Users

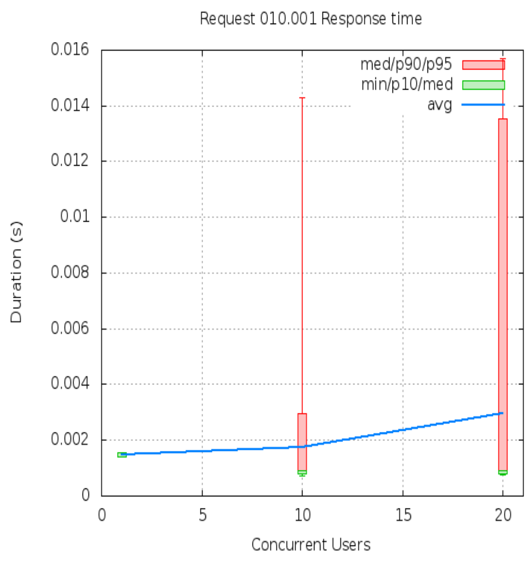

Request 005.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.012
0.01
0.008
0.006
0.004
0.002
0
0
5
10
15
20
Concurrent Users

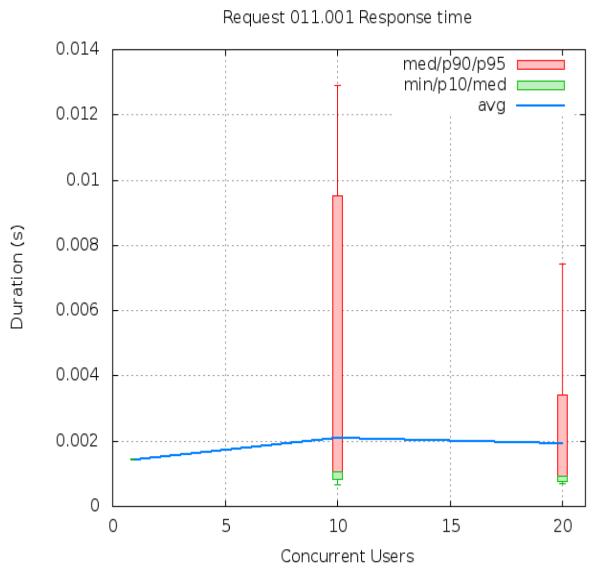

Request 006.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
5
10
15
20
Concurrent Users

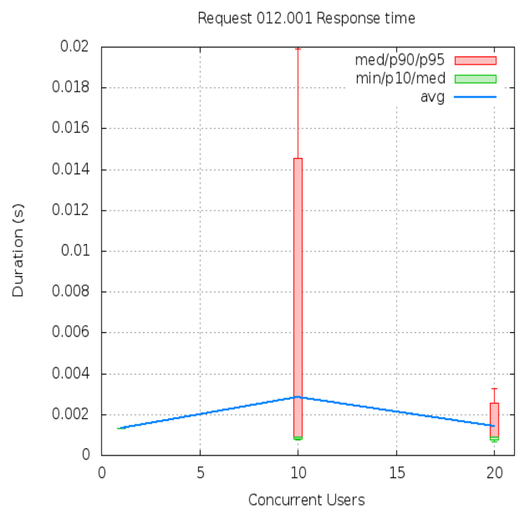

Request 007.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
5
10
15
20
Concurrent Users

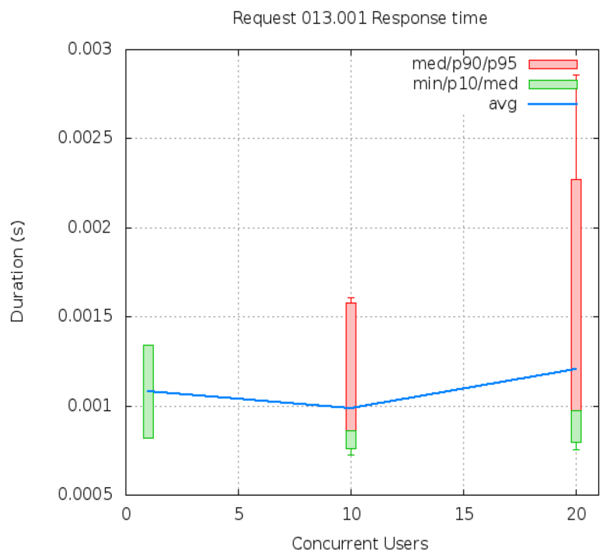

Request 008.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
0.025
0.02
0.015
0.01
0.005
0
0
5
10
15
20
Concurrent Users

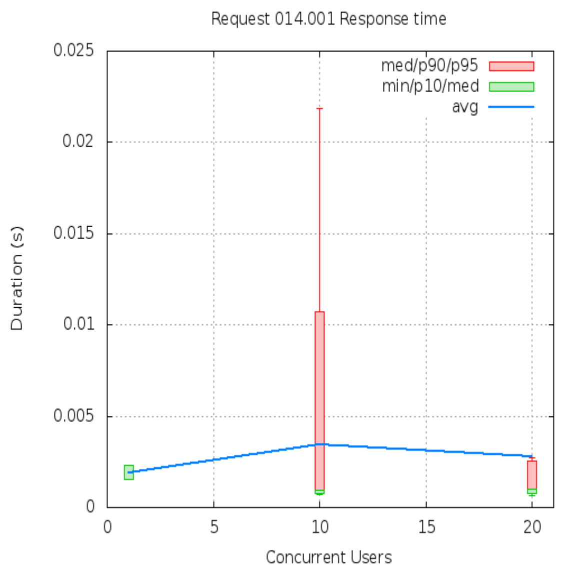

Request 009.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg

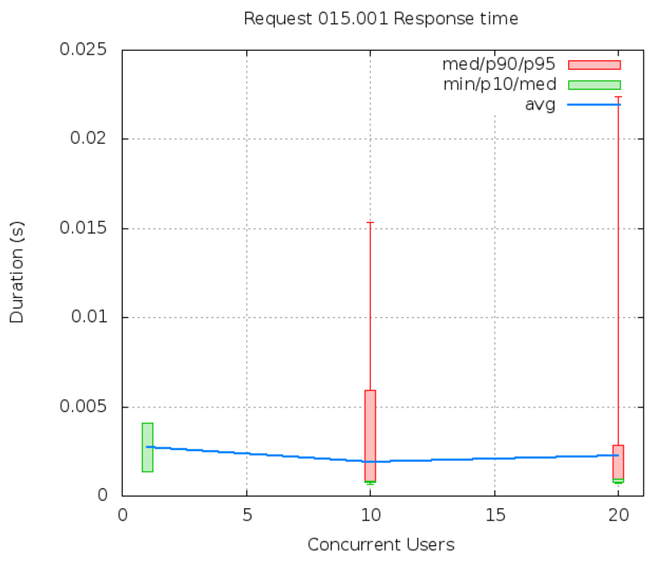

Request 010.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg

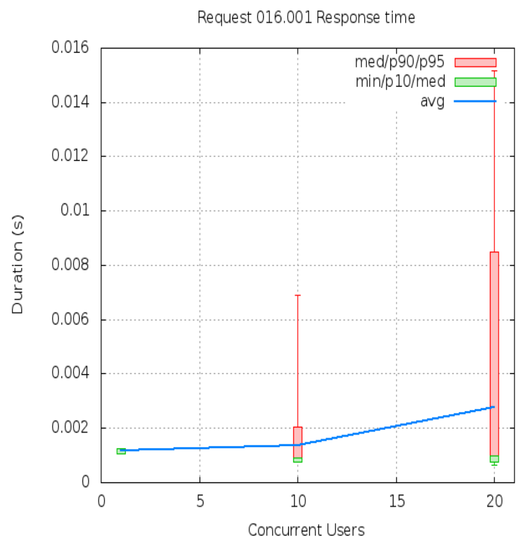

Request 011.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg

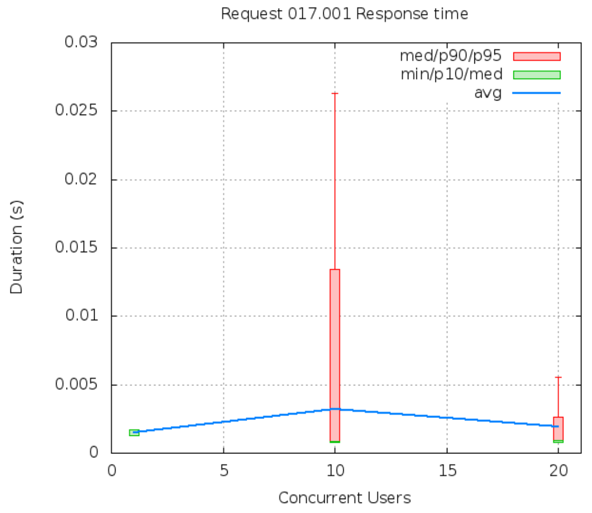
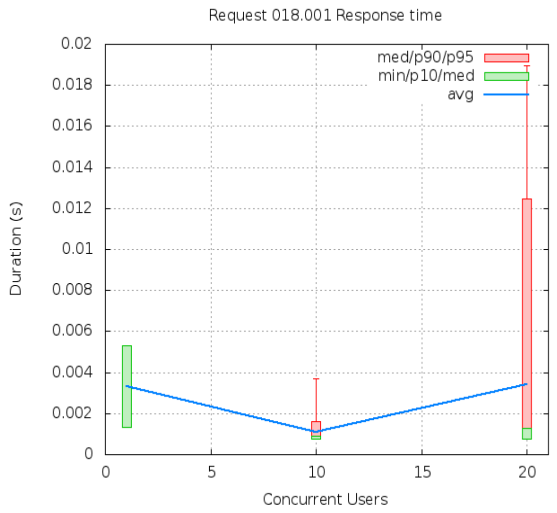
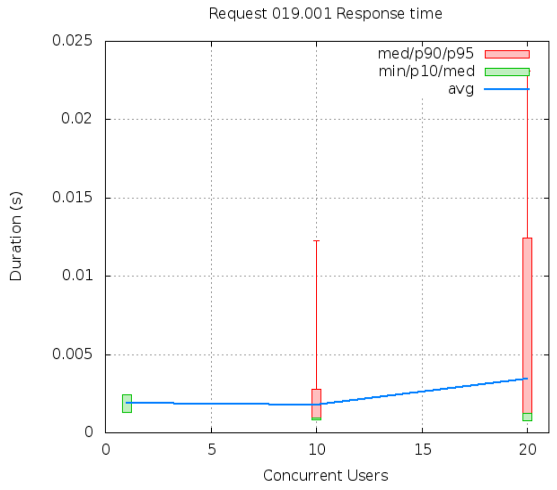
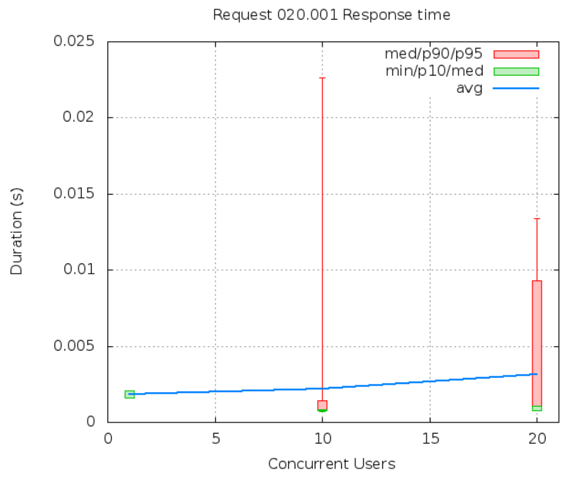
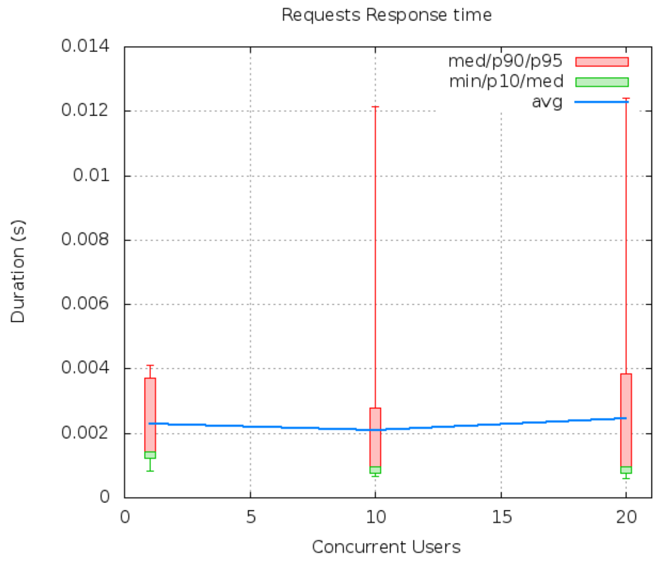
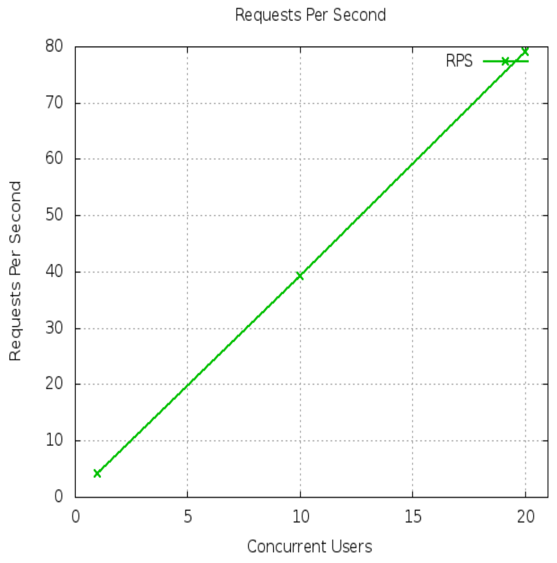
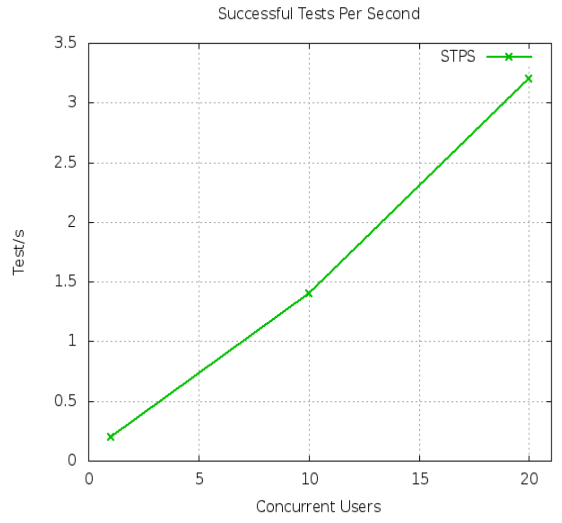
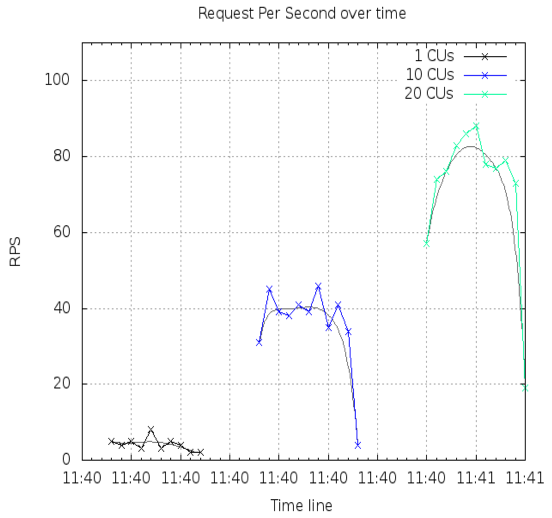

Request 012.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg


Request 013.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg


Request 014.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg


Request 015.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg


Request 016.001 Response time
Duration (s)
Concurrent Users
med/p90/p95
min/p10/med
avg


Request 017.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

Request 018.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

Request 019.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

Request 020.001 Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

Requests Response time
med/p90/p95
min/p10/med
avg
Duration (s)
Concurrent Users

Requests Per Second
RPS
Requests Per Second
Concurrent Users

Successful Tests Per Second
STPS
Test/s
Concurrent Users

Request Per Second over time
1 CUs
10 CUs
20 CUs
RPS
Time line

KEYWORDS

CUs: Concurrent users or number of concurrent threads executing tests.

Request: a single GET/POST/redirect/XML-RPC request.

Page: a request with redirects and resource links (image, css, js) for an HTML page.

STPS: Successful tests per second.

SPPS: Successful pages per second.

RPS: Requests per second, successful or not.

maxSPPS: Maximum SPPS during the cycle.

maxRPS: Maximum RPS during the cycle.

MIN: Minimum response time for a page or request.

AVG: Average response time for a page or request.

MAX: Maximum response time for a page or request.

P10: 10th percentile, response time where 10 percent of pages or requests are delivered.

MED: Median or 50th percentile, response time where half of pages or requests are delivered.

P90: 90th percentile, response time where 90 percent of pages or requests are delivered.

P95: 95th percentile, response time where 95 percent of pages or requests are delivered.

MYSQL BENCHMARKING

Using Sysbench OLTP

SYSBENCH

- SysBench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load.

- The idea of this benchmark suite is to quickly get an impression about system performance without setting up complex database benchmarks or even without installing a database at all.

- Current features allow to test the following system parameters:
  - File I/O performance
  - Scheduler performance
  - Memory allocation and transfer speed
  - POSIX threads implementation performance
  - Database server performance

INSTALLATION

```
sudo apt-get install sysbench
```

For complete reference, see man page of sysbench

OLTP MODE

- This test mode was written to benchmark a real database performance. At the prepare stage the following table is created in the specified database (sbtest by default):

```
CREATE TABLE `sbtest` (    `id` int(10) unsigned NOT NULL auto_increment,
                              `k` int(10) unsigned NOT NULL default '0',
                      `c` char(120) NOT NULL default '',
                              `pad` char(60) NOT NULL default '',
                      PRIMARY KEY  (`id`),
                              KEY `k` (`k`);
```

Then this table is filled with a specified number of rows.

EXECUTION MODES

Simple

- In this mode each thread runs simple queries of the following form:
  - SELECT c FROM sbtest WHERE id=N ; where N takes a random value in range 1..<table size>

Advanced Transactional

- Each thread performs transactions on the test table. If the test table and database support transactions (e.g. InnoDB engine in MySQL), then BEGIN/COMMIT statements will be used to start/stop a transaction. Otherwise, SysBench will use LOCK TABLES/UNLOCK TABLES statements (e.g. for MyISAM engine in MySQL). If some rows are deleted in a transaction, the same rows will be inserted within the same transaction, so this test mode does not destruct any data in the test table and can be run multiple times on the same table.

Non-Transactional

- This mode is similar to Simple, but you can also choose the query to run. Note that unlike the Advanced transactional mode, this one does not preserve the test table between requests, so you should recreate it with the appropriate cleanup/prepare commands between consecutive benchmarks

PREPARE DATA

Prepare Table and data:

- sysbench --test=oltp --oltp-table-size=1000 --mysql-db=dbtest --mysql-user=root --mysql-password=root prepare

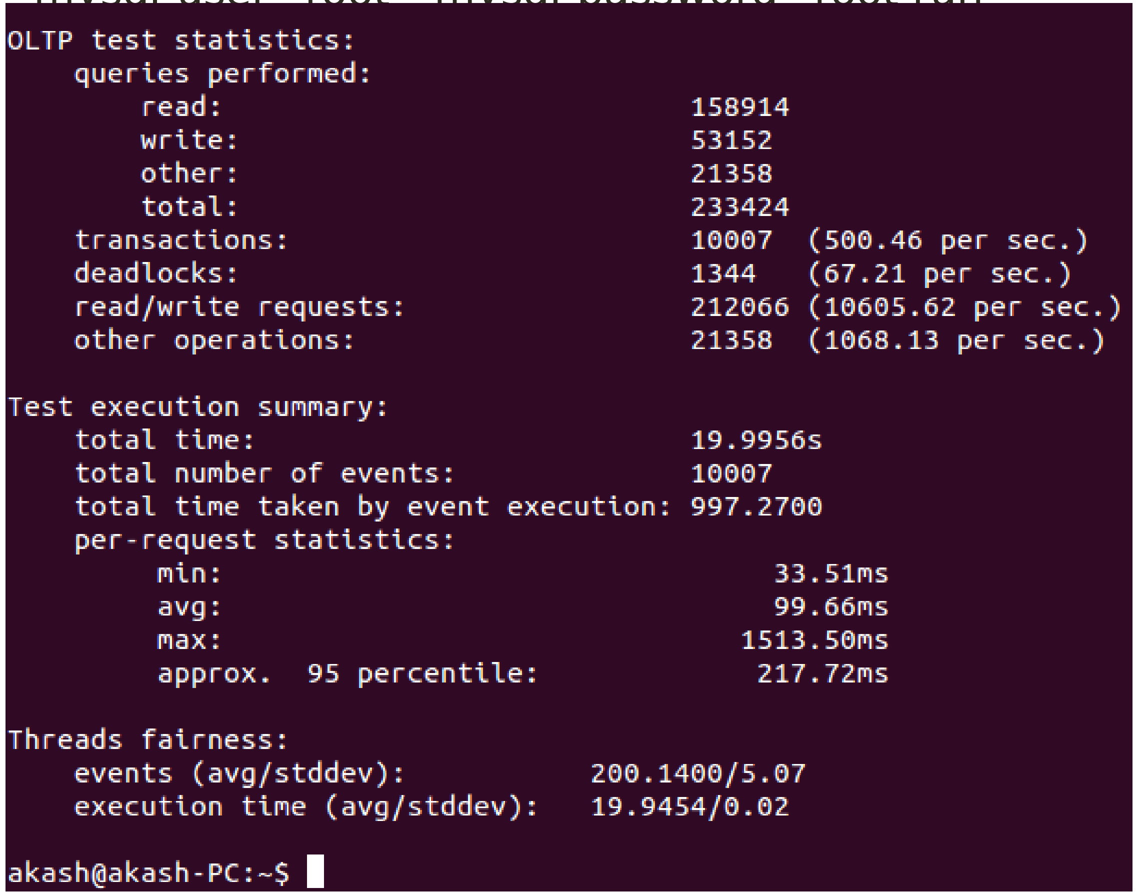
Output:

- sysbench 0.4.12: multi-threaded system evaluation benchmark
- No DB drivers specified, using mysql
- Creating table 'sbtest'...
- Creating 1000 records in table 'sbtest'...

TEST RUN

sysbench --test=oltp --oltp-table-size=1000 --oltp-test-mode=complex --num-threads=50 --max-time=60 --mysql-db=dbtest --mysql-user=root --mysql-password=root run

Output:



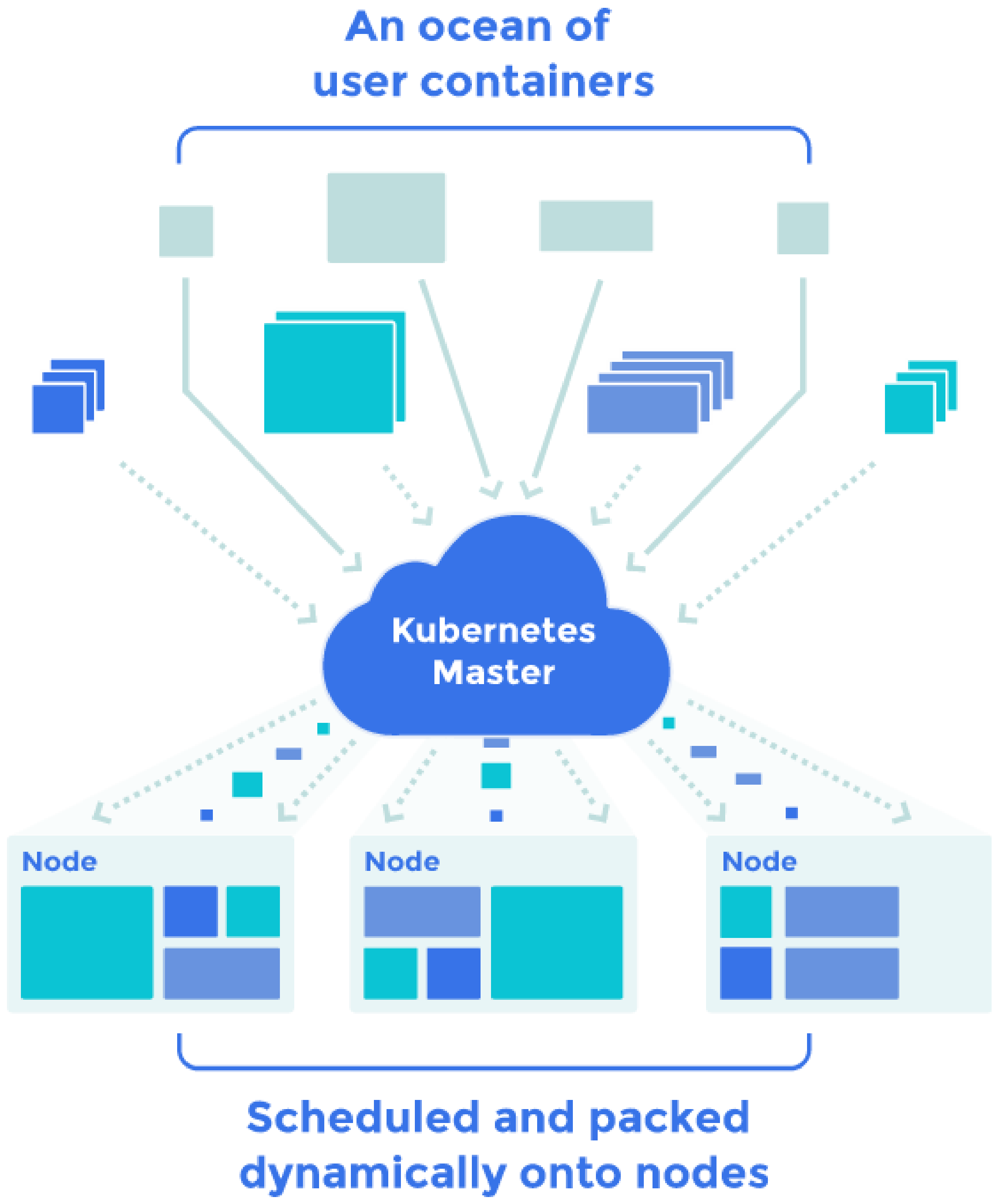
```
OLTP test statistics:
    queries performed:
        read:                            158914
        write:                           53152
        other:                           21358
        total:                           233424
    transactions:                        10007  (500.46 per sec.)
    deadlocks:                           1344   (67.21 per sec.)
    read/write requests:                 212066 (10605.62 per sec.)
    other operations:                    21358  (1068.13 per sec.)

Test execution summary:
    total time:                          19.9956s
    total number of events:              10007
    total time taken by event execution: 997.2700
    per-request statistics:
         min:                                 33.51ms
         avg:                                 99.66ms
         max:                               1513.50ms
         approx.  95 percentile:             217.72ms

Threads fairness:
    events (avg/stddev):           200.1400/5.07
    execution time (avg/stddev):   19.9454/0.02

akash@akash-PC:~$ 
```

Kubernetes


An ocean of user containers
Kubernetes Master
Node
Node
Node
Scheduled and packed dynamically onto nodes

What is Kubernetes ?

- Kubernetes is a open source orchestration system for managing containerized applications across multiple hosts. Etymologically kubernetes mean “Helmsman of a ship”.
- It actively manages the containers to ensure that the state of the cluster continually matches the user's intentions.
- As of now, it supports only Docker containers.
- Using the concepts of "labels" and "pods", it groups the containers which make up an application into logical units for easy management and discovery.

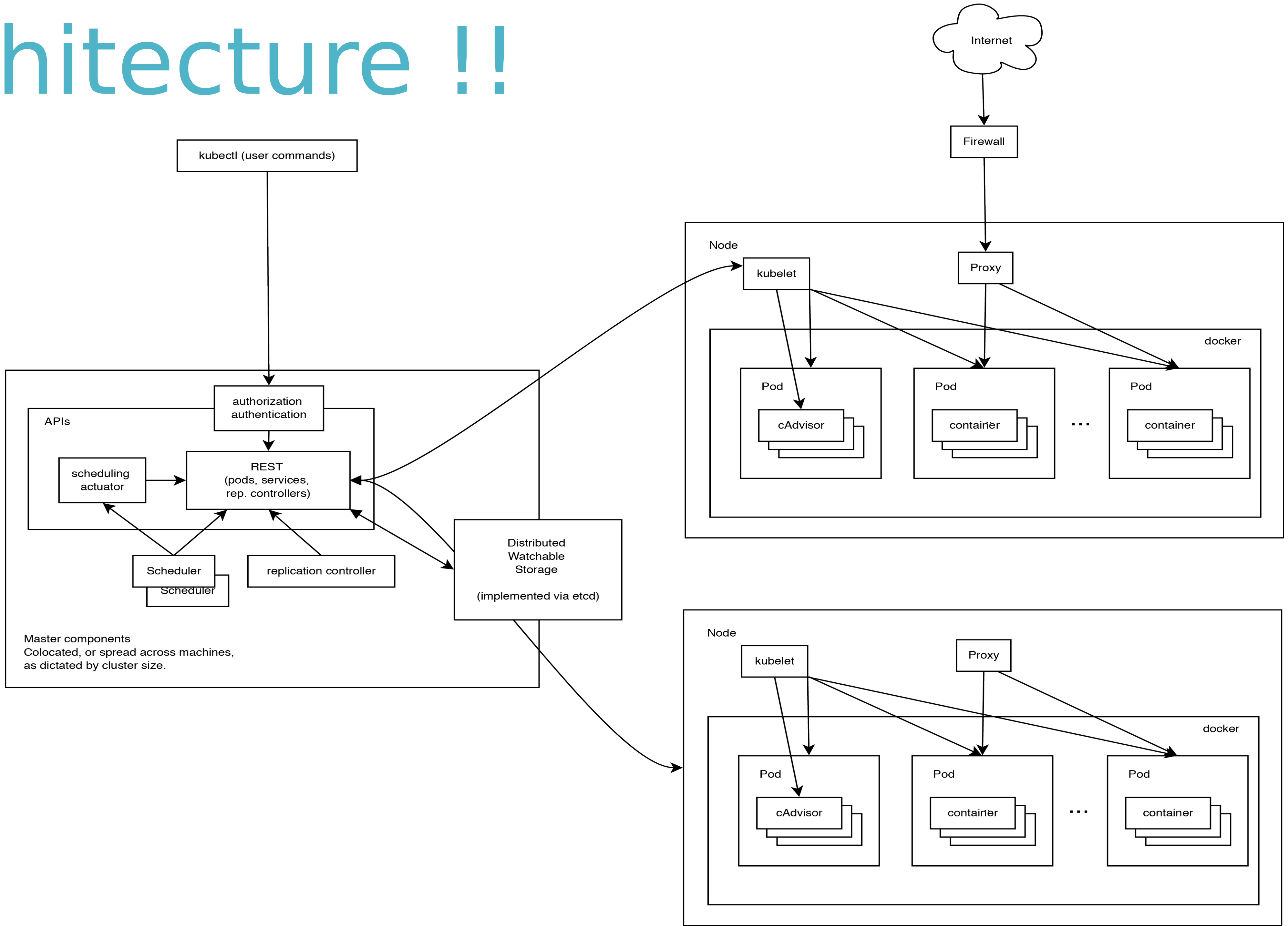
Why Kubernetes ?

- It’s lightweight, simple and accessible.
- It’s highly portable into multi-cloud world, public, private or hybrid.
- It’s highly modular, designed so that all of its components are easily swappable.
- Easy management of Linux containers as a single system to accelerate Dev and simplify Ops.
- It provides basic mechanisms for deployment, maintenance, and scaling of applications.

Terminologies

- **Pods:** All containers run inside pods. A pod can host a single container, or multiple cooperating containers. In latter case, the containers in the pod are guaranteed to be co-located on the same machine and can share resources.
- **Replication Controller:** A replication controller ensures that a specified number of pod "replicas" are running at any one time. If there are too many, it will kill some. If there are too few, it will start more. It runs under controller manager service discussed in further slides.
- **Node:** Node is a worker machine in Kubernetes, previously known as Minion.

Architecture !!



On Master..

- **Docker:** It is an open platform for building, shipping and running distributed applications.
- **Flanneld:** It is used as an overlay network which simplifies networking between master and nodes.
- **Etcd:** It is an open-source distributed key value store that provides shared configuration and service discovery. All persistent master state is stored in an instance of etcd. This provides a great way to store configuration data reliably. With watch support, coordinating components can be notified very quickly of changes.
- **API-Server:** The apiserver serves up the Kubernetes API. It mainly processes REST operations, validates them, and updates the corresponding objects in etcd.
- **Scheduler:** The scheduler binds unscheduled pods to nodes.
- **Controller Manager Server:** All other cluster-level functions are

On Minions

- **Kubelet:** The Kubelet manages pods and their containers, their images, their volumes, etc.
- **Kube-proxy and Services:** Think of Services and kube-proxy as a distributed multi-tenant load-balancer.
- Services are defined at the creation time of a pod. Proxy reflects services on each node and can do simple TCP and UDP stream forwarding (round robin) across a set of endpoints.
- Minion also needs **Docker and Flanneld** services to work.

Networking

- Flannel is used as an overlay network which simplifies networking between master and nodes. It is backed by etcd server running on master.
- Flannel creates an overlay mesh network that provisions a subnet to each node. This gives a unique IP address to each pod from /24 network subnet of host.

REFERENCES

- http://funkload.nuxeo.org/funkload.pdf
- http://blog.flux7.com/blogs/benchmarks/using-sysbench-to-benchmark-mysql
- https://dev.mysql.com/downloads/benchmarks.html
- man of sysbench

THANK YOU.

Akash Agarwal (201406593)

Akhil Singh (201302181)

Soujanya Ponnapalli (201301192)

RanVijay Singh (201250899)