Яндекс

PYCON RUSSIA

How I Learned to Stop Worrying and Love the BFG

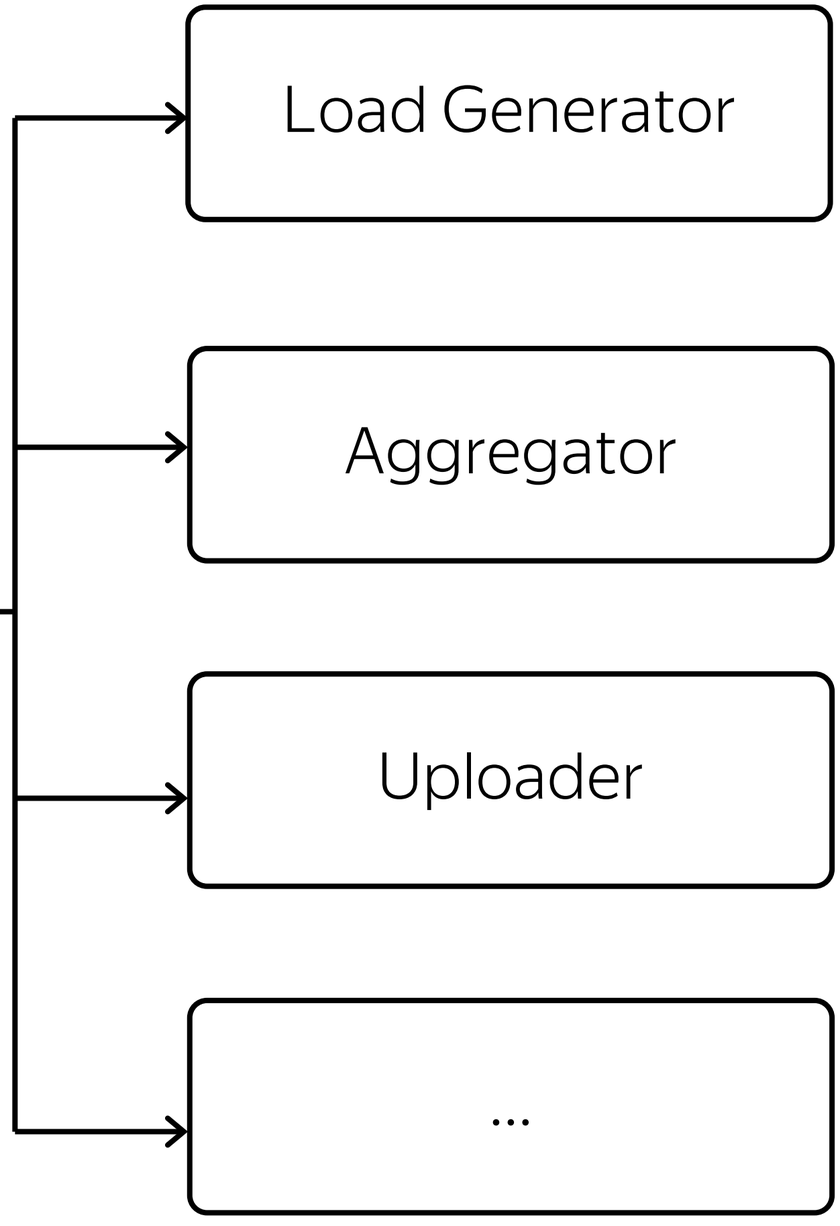
Надежда Миргородская, Яндекс

План

- Что такое BFG
- Где его взять
- И что с ним делать

Что такое BFG

Часть Яндекс Танка



Генераторы нагрузки

	Phantom	JMeter	BFG
Производительность	Высокая	Средняя	Средняя
Интерфейс	Консоль	GUI	IDE
Сценарии?	Нет	Да	Да
Язык сценариев	-	Java	Python

BFG: pro et contra

Плюсы

- это код
- экосистема Питона

Минусы

- это код
- производительность не так высока

Где взять BFG

Это опенсорс

https://github.com/yandex/yandex-tank

https://overload.yandex.net

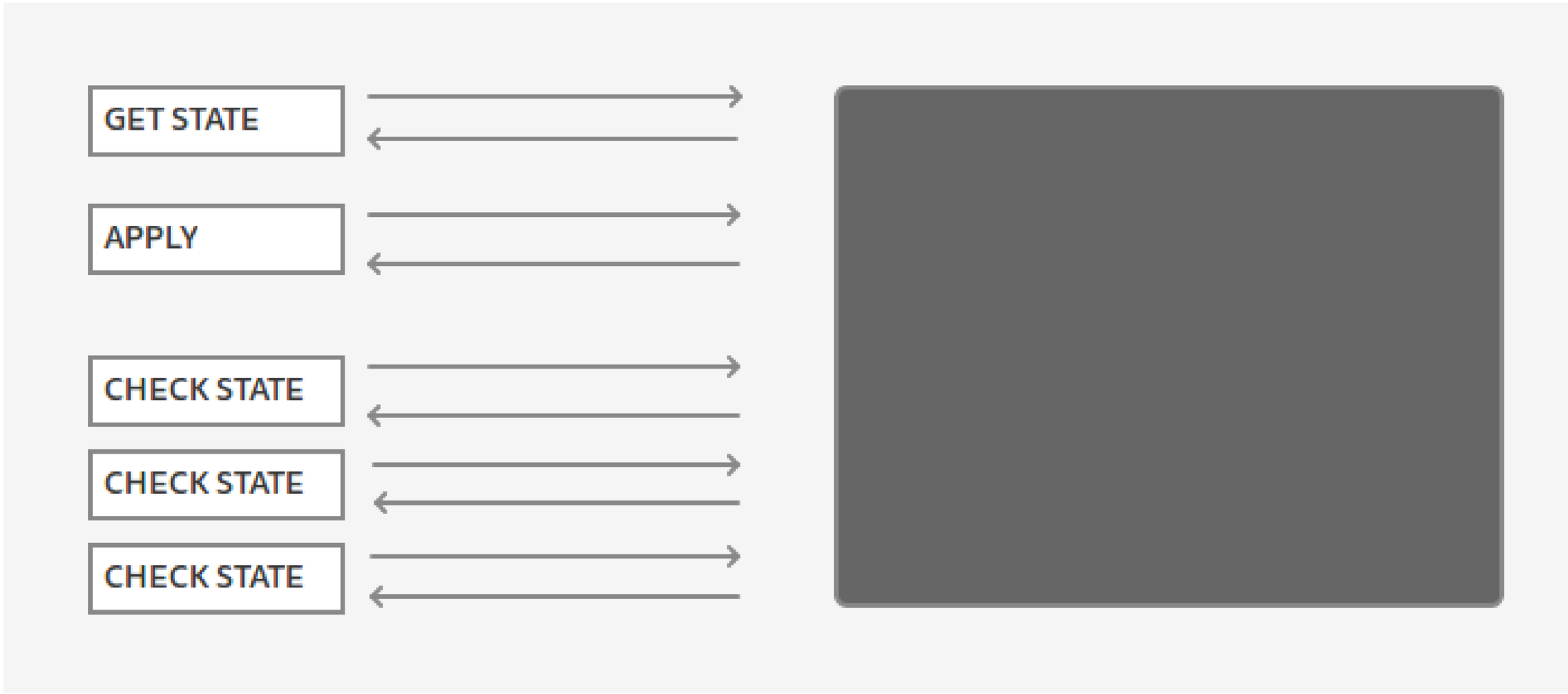
https://gitter.im/yandex/yandex-tank

И что с ним делать

Вопросы производительности

- Время ответа
- Доля ошибок
- Стабильность
- Предельная нагрузка
- Отказоустойчивость

Нам бы побыстрее: сценарий



BFG testware

- Сценарий
- Тестовые данные
- Конфигурационный файл

Сценарий BFG: класс LoadTest

```
class LoadTest(object):
    def __init__(self, gun):
        self.gun = gun
    def get_state(self):
        response = requests.get(get_state_url)
        config = parse_state(response)
    def apply(self):
        requests.post(apply_url, data=config)
```

Добавляем измерения

```
class LoadTest(object):
  def get_state(self):
    with sef.gun.measure('get_state') as measure:
      response = self.client.get_state()
      measure['proto_code'] = response.status_code

result = parse_state(response)
```

Порядок вызовов: метод default

```
def default(self):
    self.get_state(get_state_url)
    self.apply(config)
    until state == 'DONE’:
        self.check_state()
```

Базовый конфиг

```
[bfg]

gun_type = ultimate
loop = 100
instances = 10
instances_schedule = const(10,10m)
ammofile = ./ammo.txt


[ultimate_gun]

module_path = ./
module_name = performance_test
```

Время ответа по запросам

Кумулятивные квантили по тэгам

	99%	98%	95%
Весь тест	6500.000 ms	1500.000 ms	1500.000 ms
apply	12000.000 ms	1500.000 ms	1500.000 ms
delta	5500.000 ms	1500.000 ms	750.000 ms
get_state	60000.000 ms	60000.000 ms	60000.000 ms

Доля ошибок по запросам

HTTP коды по тэгам

	200	503
Весь тест	8415	36
apply	4176	18
delta	4178	18
get_state	61	0

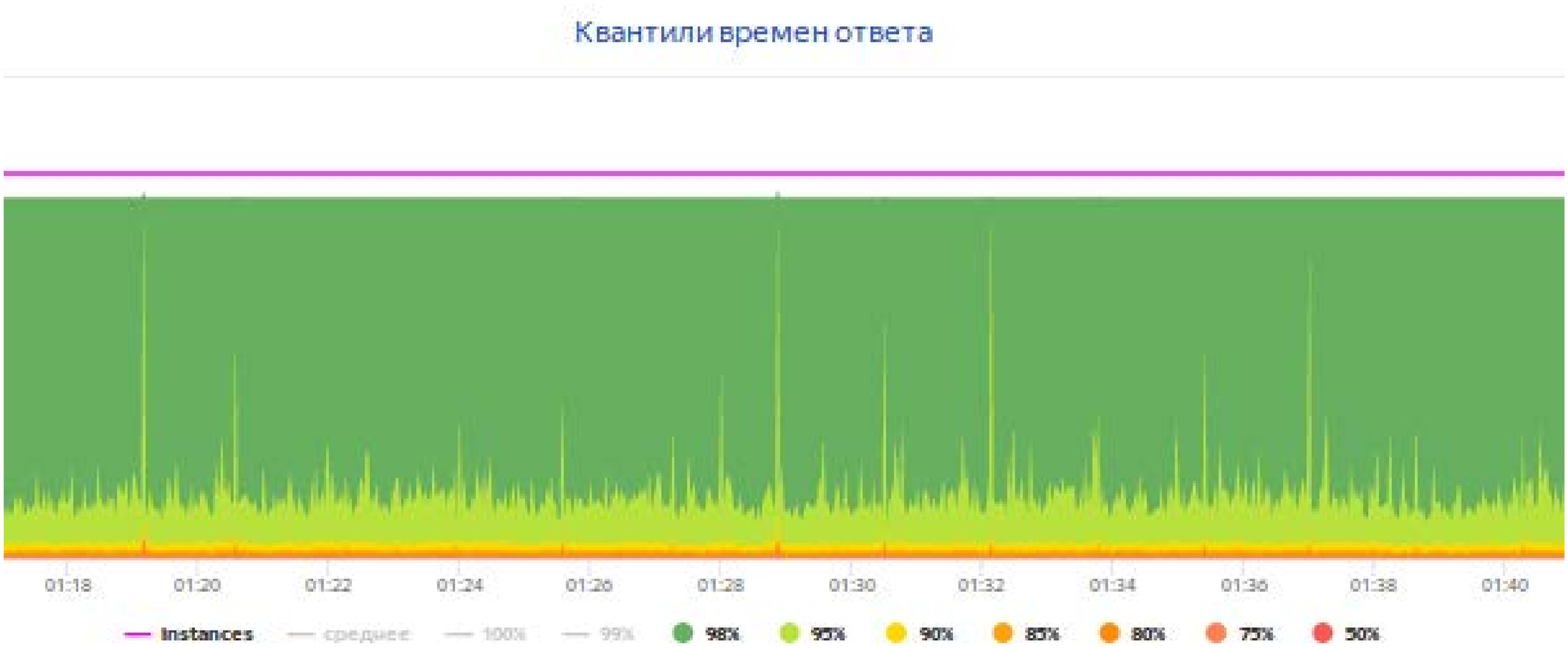
Теги внутри тегов:
общее время транзакции

```
def default(self):
    self.get_state()
    with self.gun.measure("activated"):
        with self.gun.measure("apply"):
            self.apply()
        with self.gun.measure("check_state"):
            self.check_state()
```

Вопросы производительности

- Время ответа ✓
- Доля ошибок ✓
- Стабильность
- Предельная нагрузка
- Отказоустойчивость

Стабильность



Вопросы производительности

- Время ответа ✓
- Доля ошибок ✓
- Стабильность ✓
- Предельная нагрузка
- Отказоустойчивость

Профиль нагрузки: интенсивность

instances_schedule

- instances_schedule = line(1,3,3m)

rps_schedule

- instances = 10
- rps_schedule = const(50,5m)

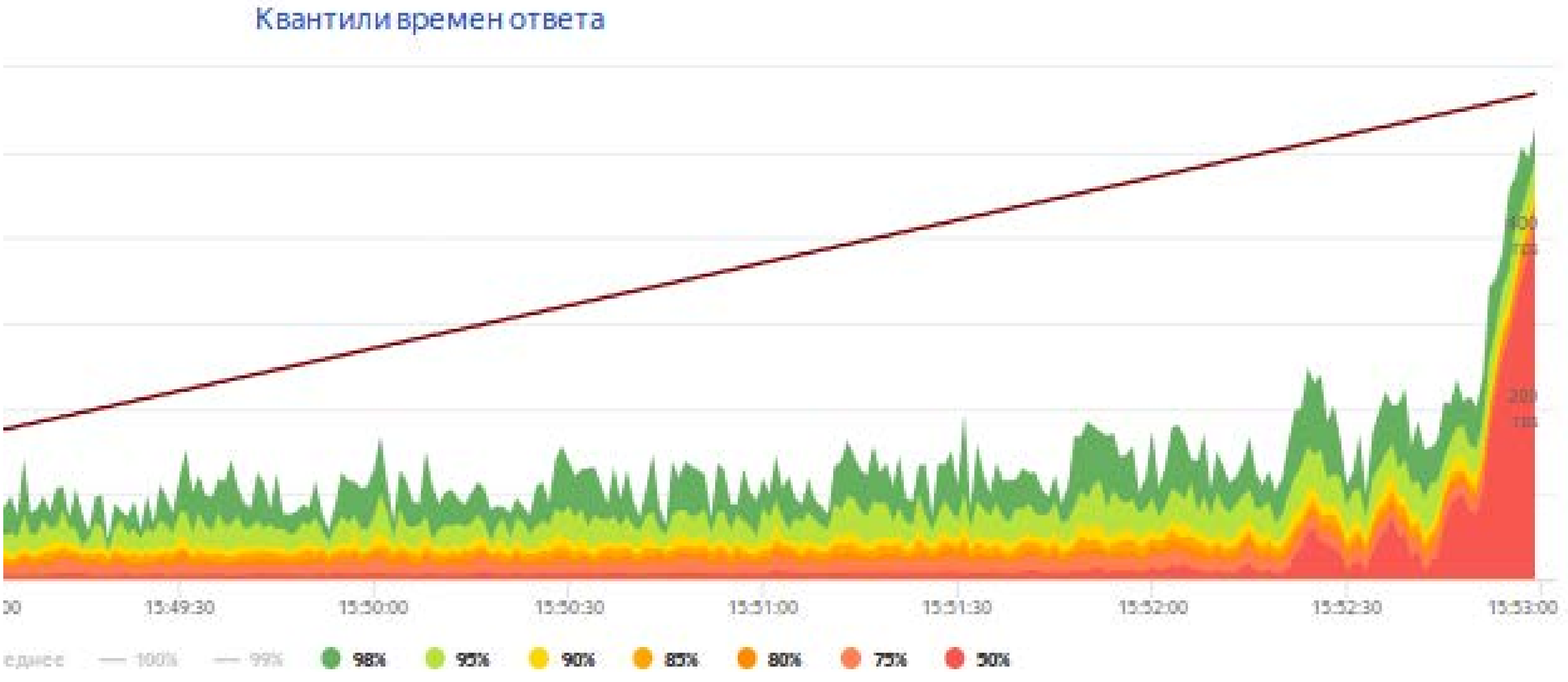
free_instances

- instances = 10

Подкладываем соломки

```
autostop = http(5xx, 30%, 60s)
autostop = time(1500s, 10m)
```

Стресс-тест


Квантили времен ответа
15:49:30
15:50:00
15:50:30
15:51:00
15:51:30
15:52:00
15:52:30
15:53:00
98%
95%
90%
85%
80%
75%
50%

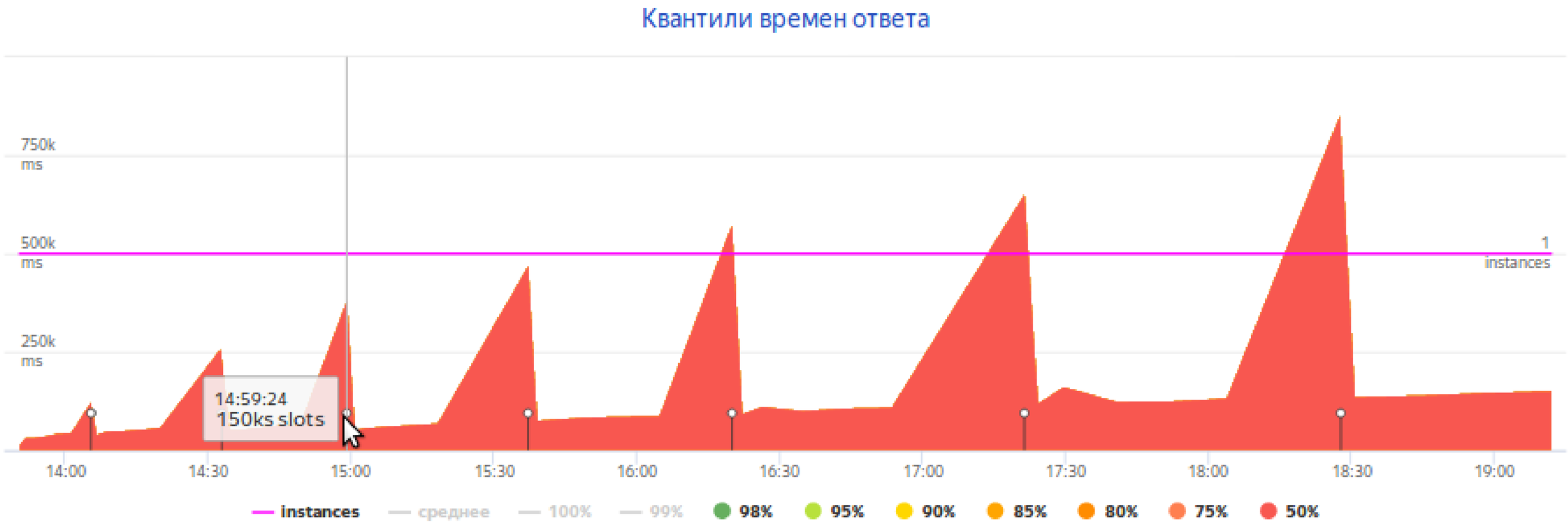
Вопросы производительности

- Время ответа ✓
- Доля ошибок ✓
- Стабильность ✓
- Предельная нагрузка ✓
- Отказоустойчивость

Отказ и восстановление

- Гасим реплику
- Проверяем состояния
- Фиксируем объем базы

Recovery test



HTTP коды

Много тестов

- Load test
- Stress test
- Stability test
- Recovery test
- Volume test

Profit!

- Повторяемость
- Входит в набор регрессионных тестов
- Однородность с функциональными тестами
- Легко вносить изменения

Спасибо за внимание!

Надежда Миргородская


szypulka@yandex-team.ru


szypulka


szypulka.github.io