

Яндекс



Учимся анализировать результаты нагрузочного тестирования

Алексей Лавренюк

Словарик нагрузчика

- › танк – генератор нагрузки
- › стрельба – нагрузочный тест
- › мишень – обстреливаемый сервис
- › ручка – один из URL сервиса (от Handler)
- › патроны – тестовые данные (запросы)
- › гильзы – метки в тестовых данных

Мишень

Тренировочный сервис на Tornado (ссылка на код в конце доклада)

Машинка в Openstack (Xenial, 32x Xeon 2GHz, 58GiB)

У сервиса пять пронумерованных ручек:

- › target.example.org/1 – target.example.org/5

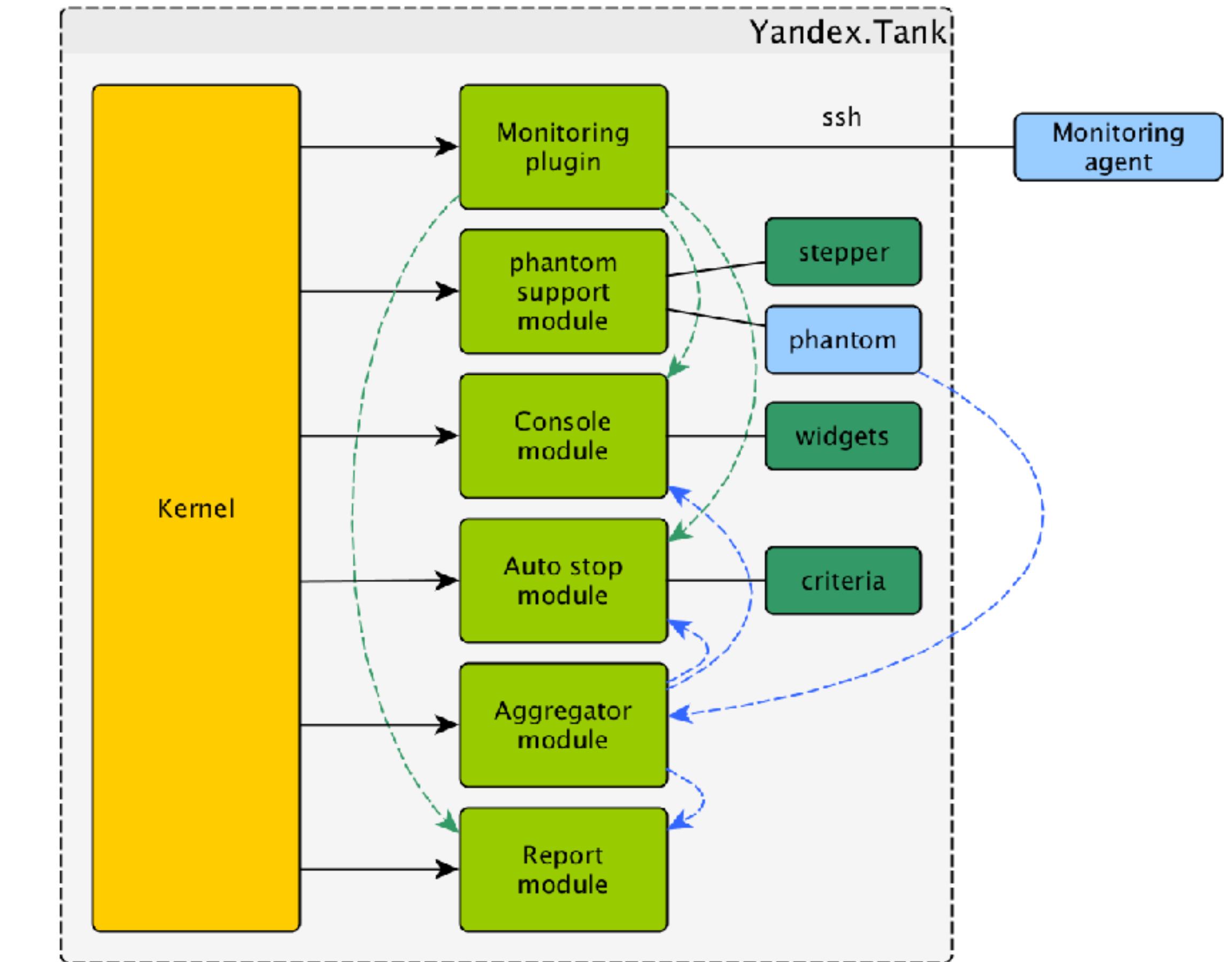
| каждая ручка делает что-то очень простое

Генератор

Яндекс.Танк – нагрузочный фреймворк, opensource

В Яндексе есть облако таких танков, с которых можно стрелять.

Вы можете развернуть свой танк
(github.com/yandex/yandex-tank)

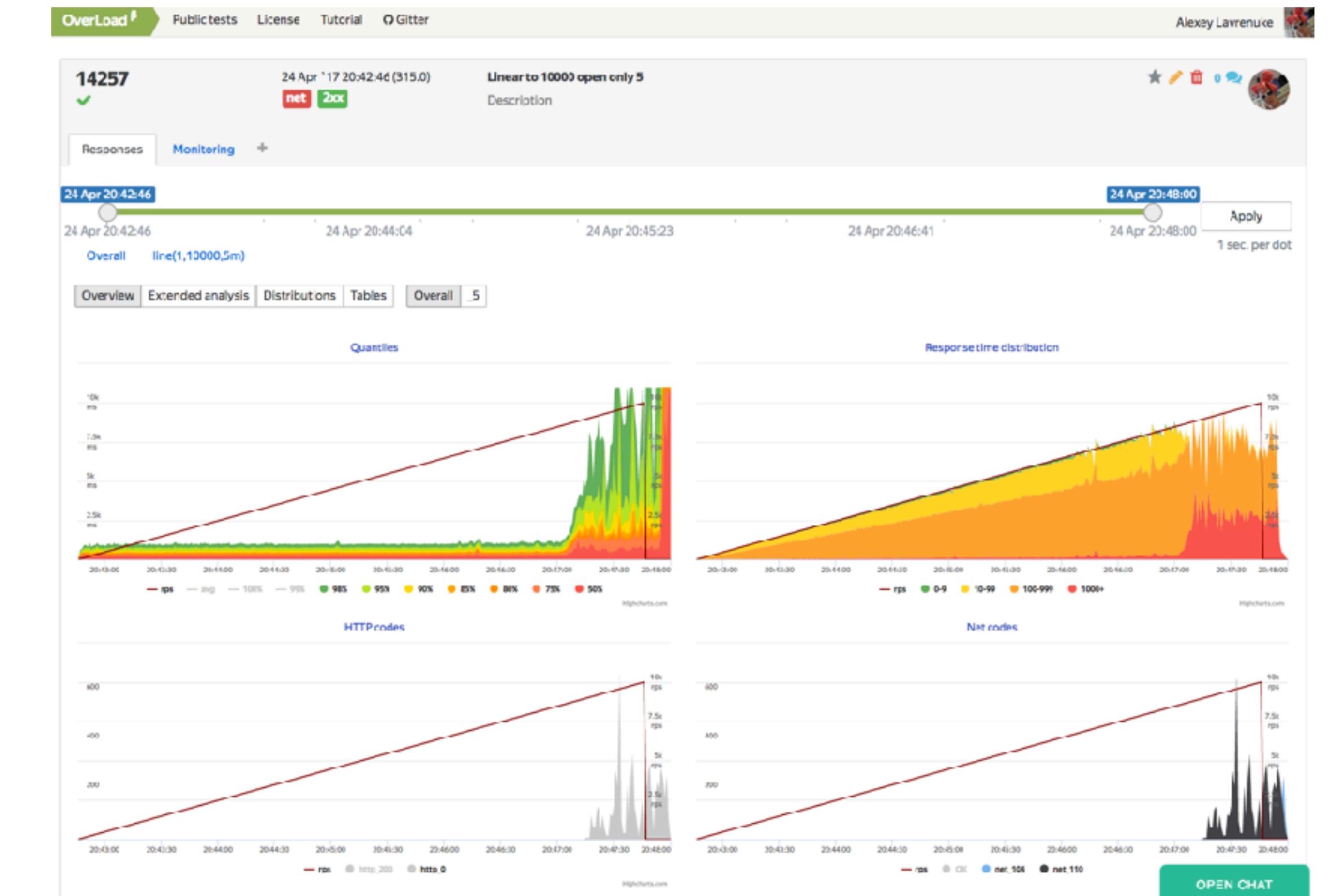


Анализ

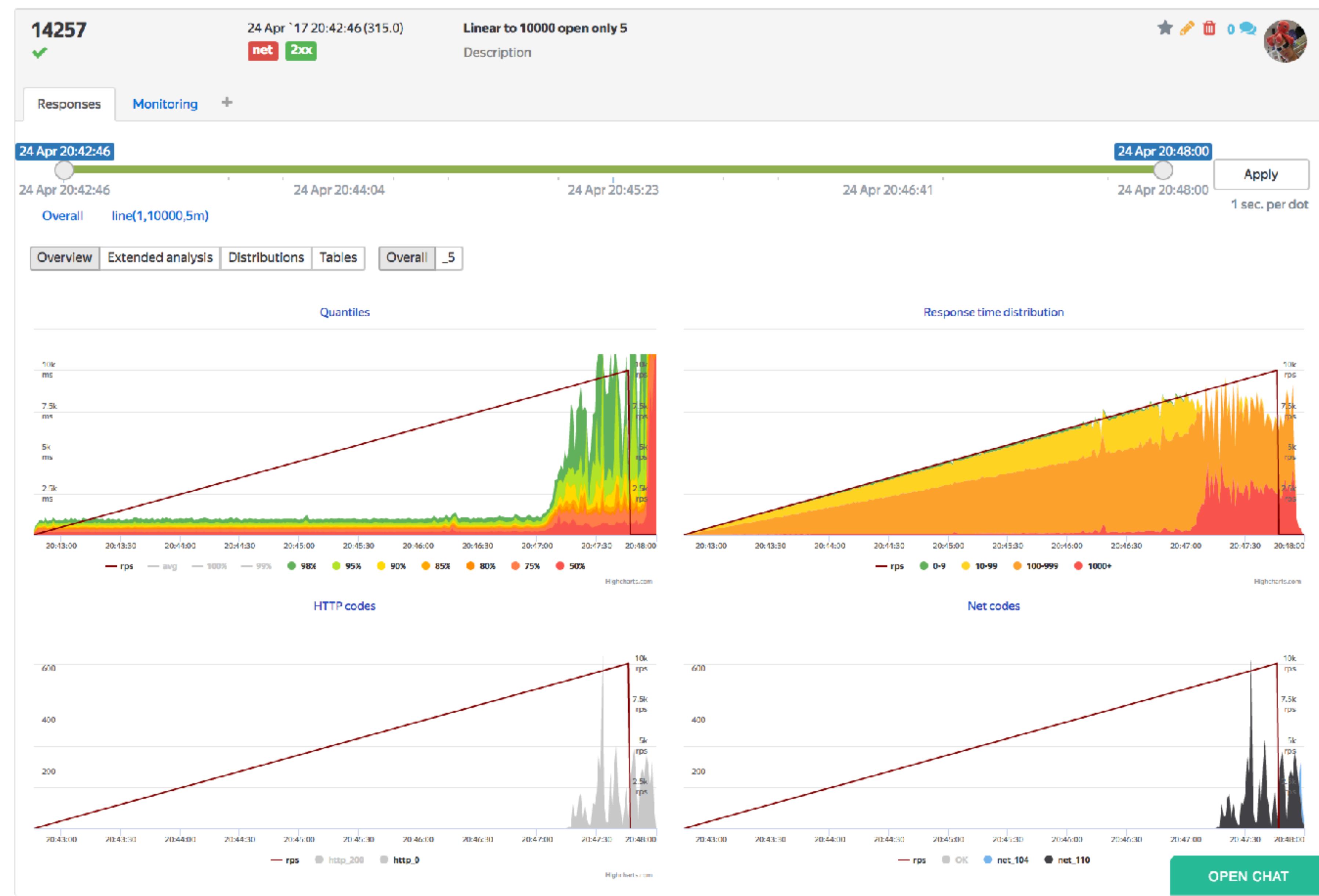
Overload – сервис для хранения и анализа результатов тестов.

Публичная бета доступна бесплатно.

Я залил туда результаты тестов и каждый из вас сможет их детально изучить (overload.yandex.net)



 overload.yandex.net/14257



**14257**

24 Apr '17 20:42:46 (315.0)

net 2xx

Linear to 10000 open only 5

Description



Responses

Monitoring +

24 Apr 20:42:46

24 Apr 20:42:46

24 Apr 20:44:04

24 Apr 20:45:23

24 Apr 20:46:41

24 Apr 20:48:00

Apply

1 sec. per dot

Overall line(1,10000,5m)

Overview

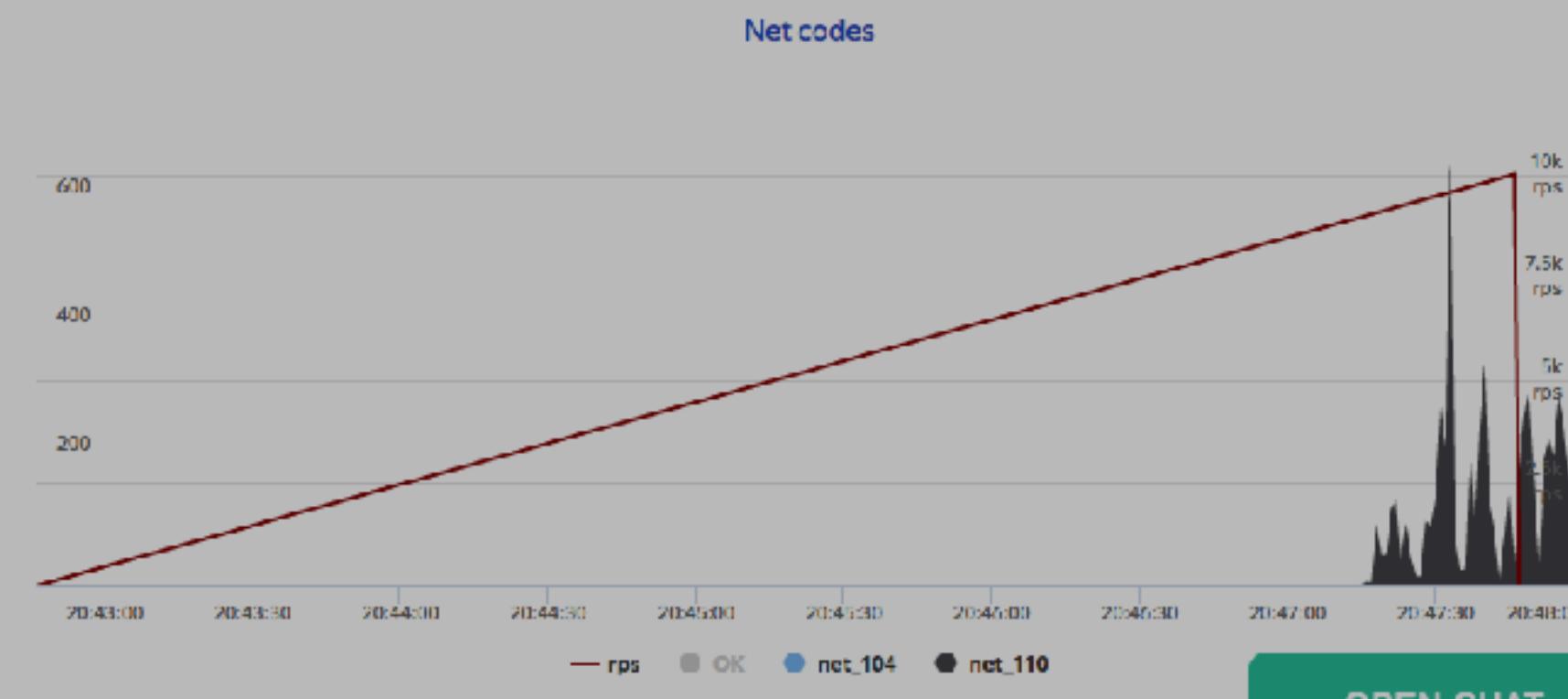
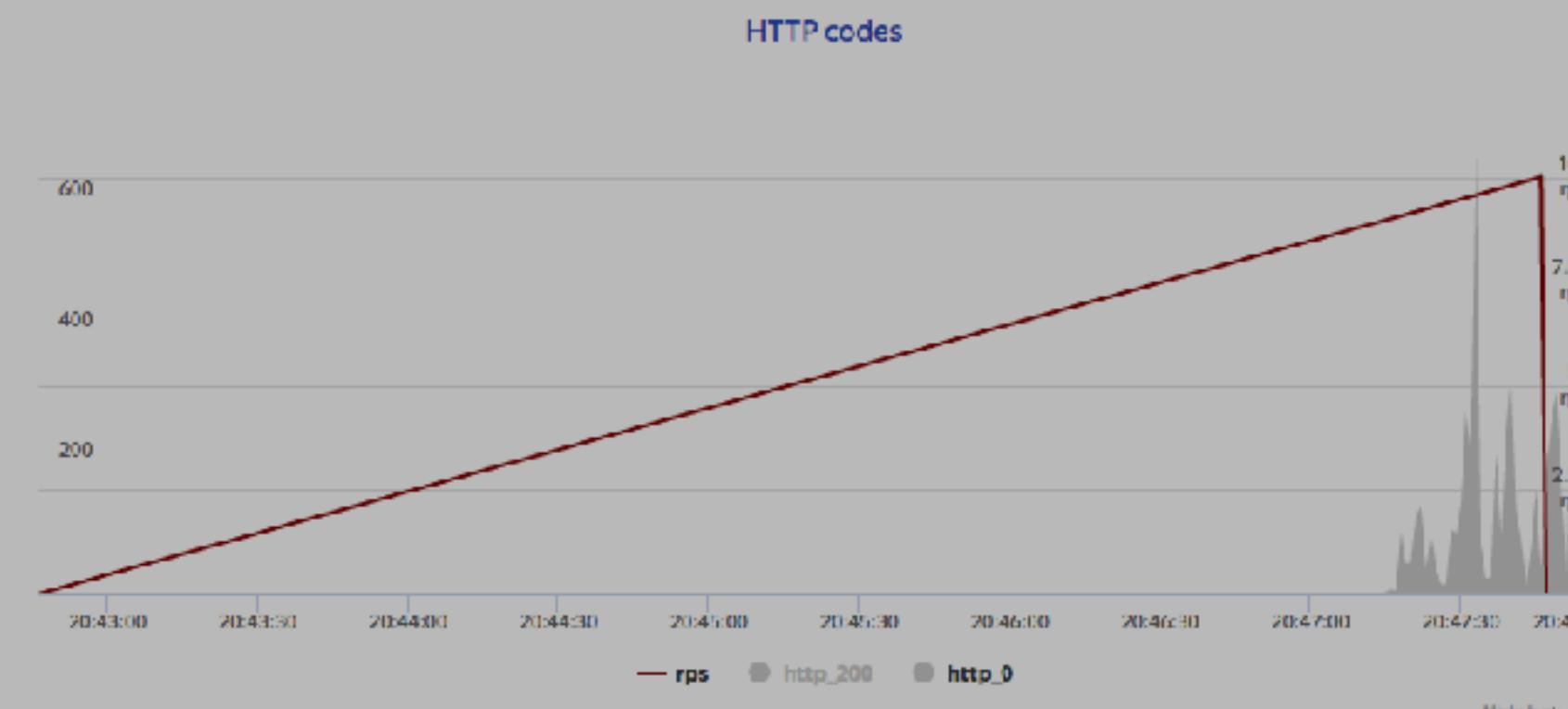
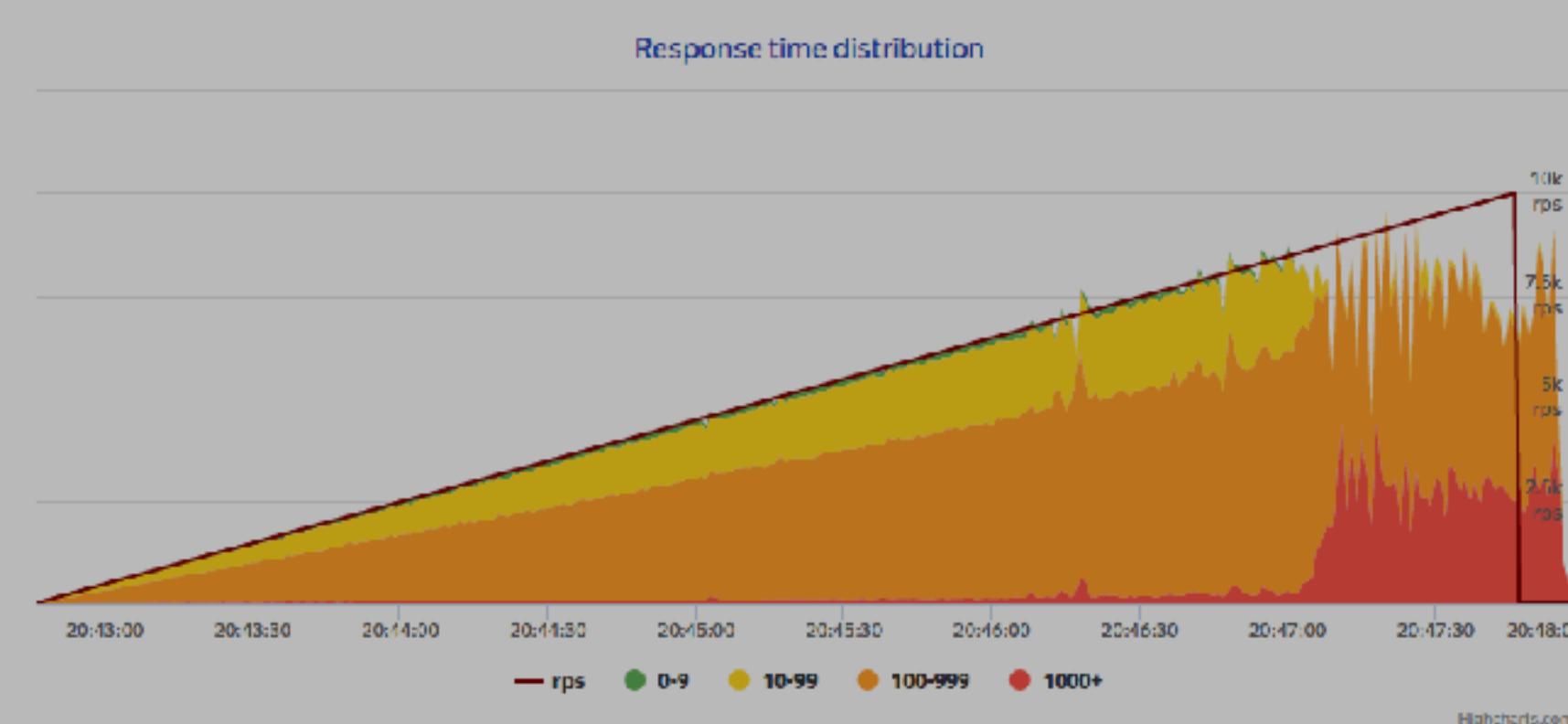
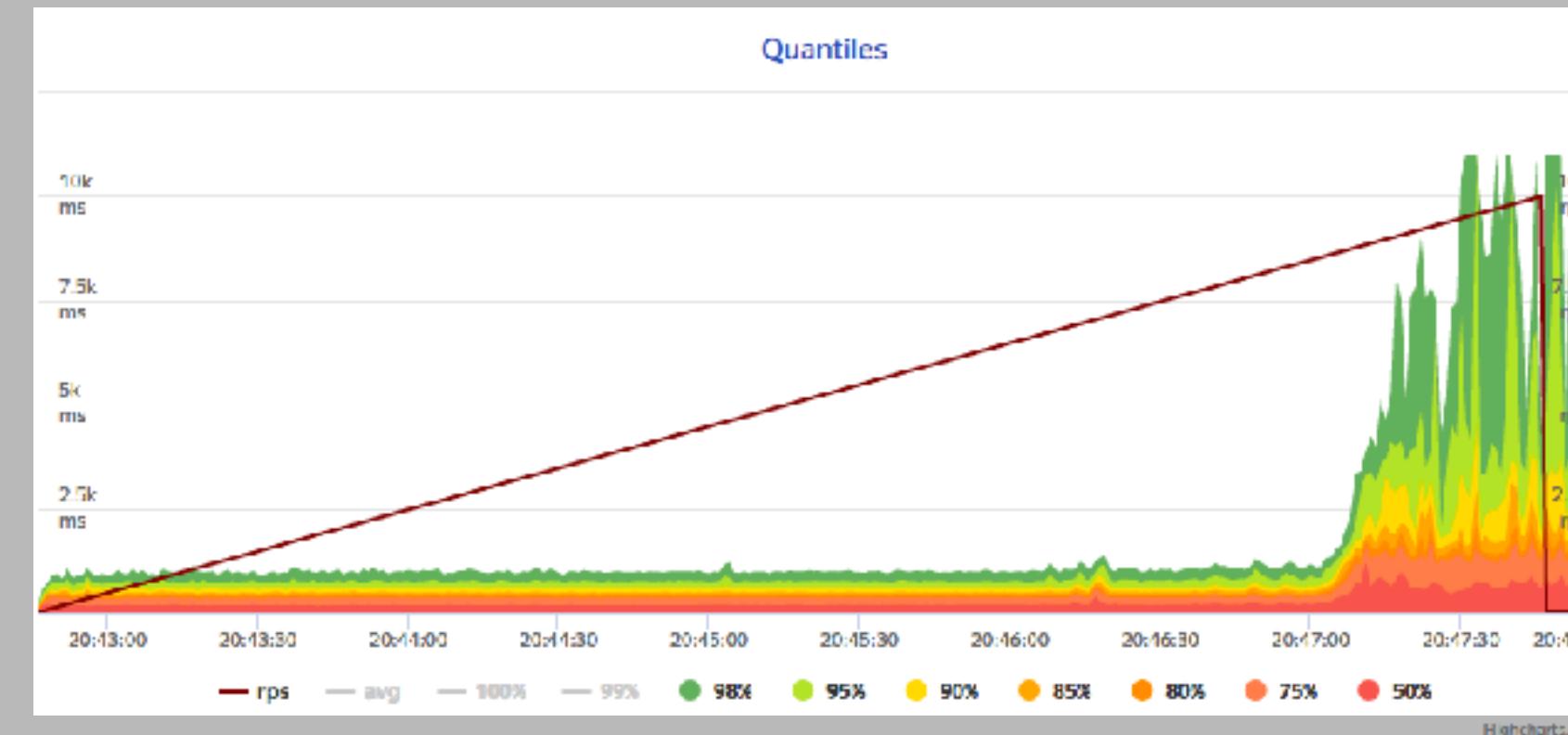
Extended analysis

Distributions

Tables

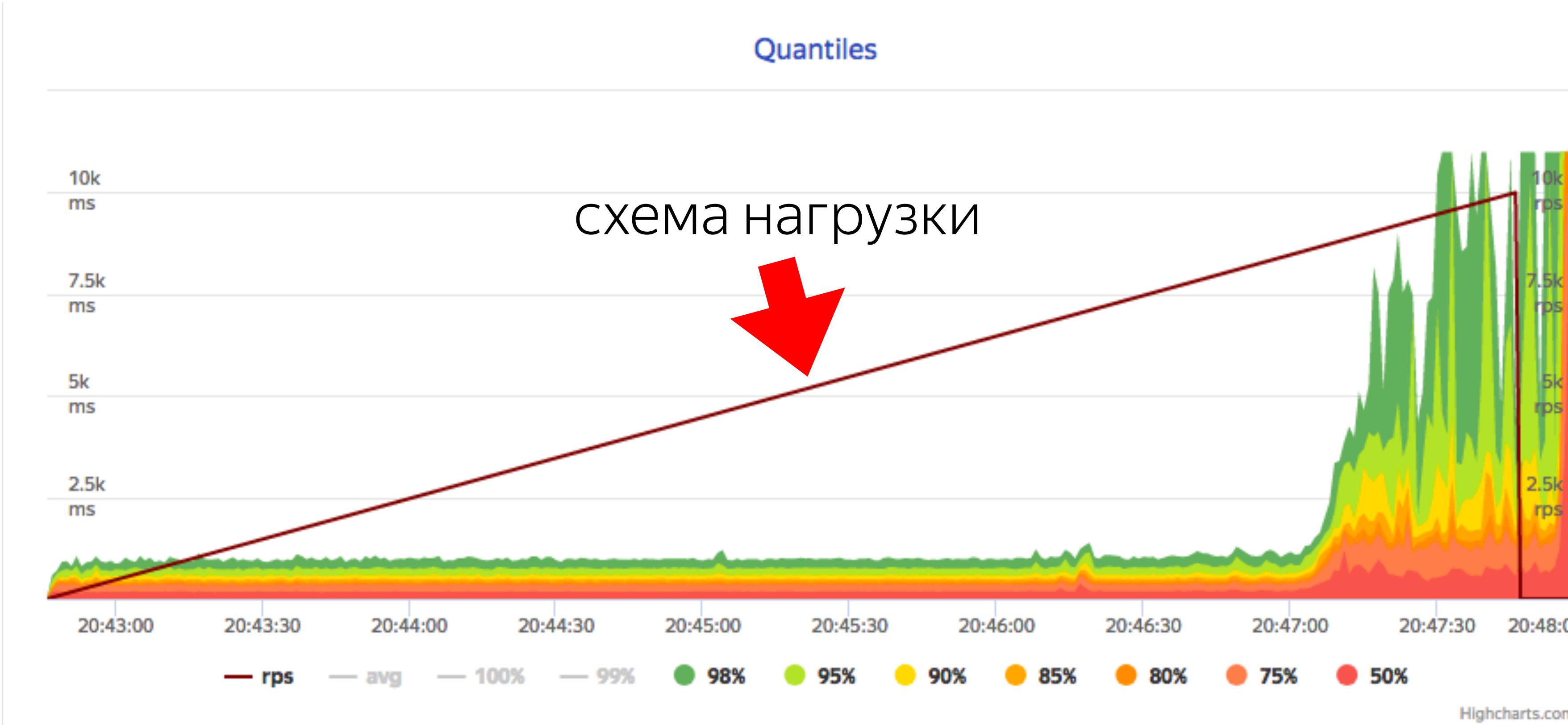
Overall

_5



OPEN CHAT

Квантили времен ответов



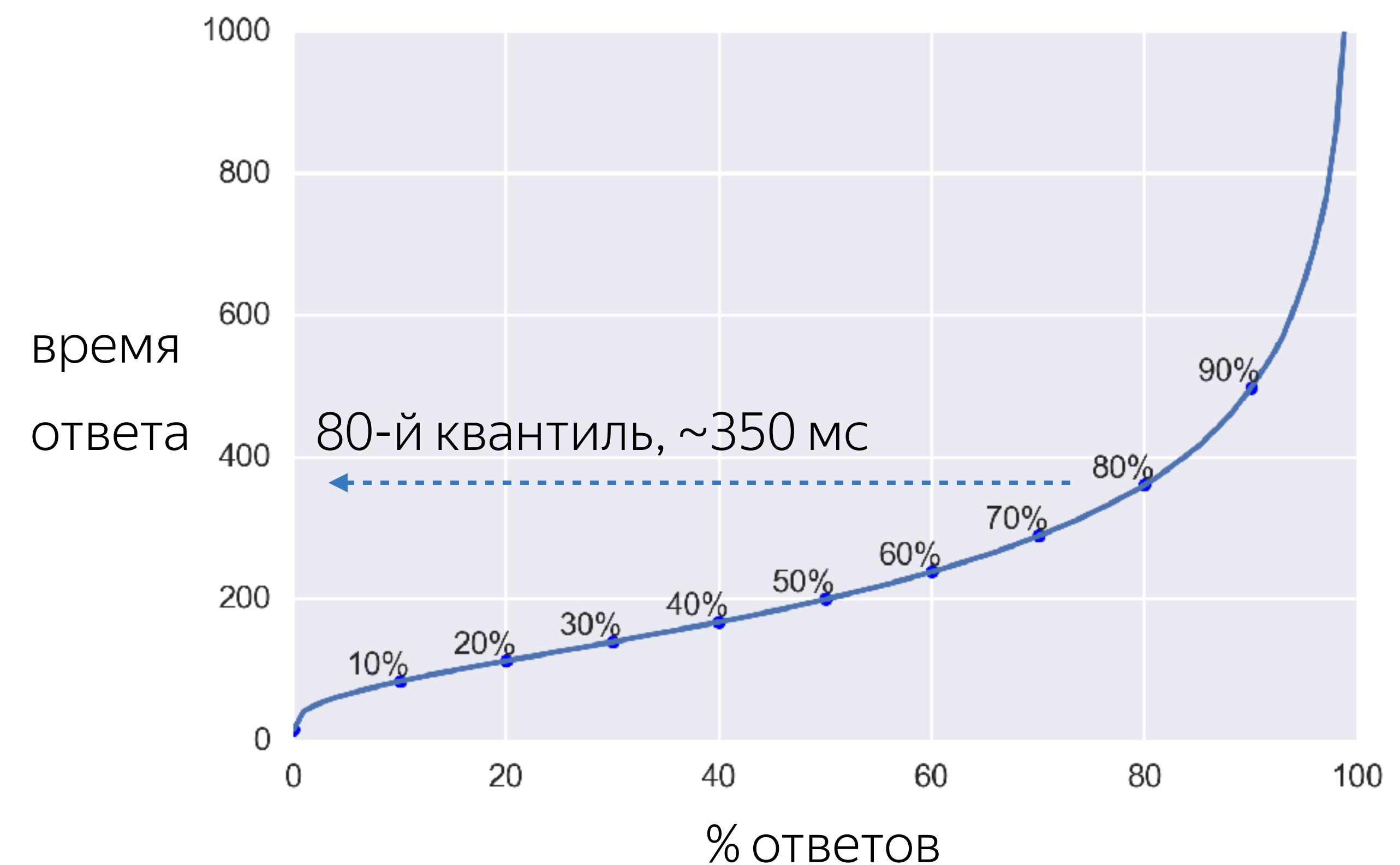
Квантили

Квантиль* уровня n – время,
в которое укладывается
 $n\%$ ответов

Медиана – 50-й квантиль

Максимум – 100-й квантиль,
минимум – 0-й квантиль

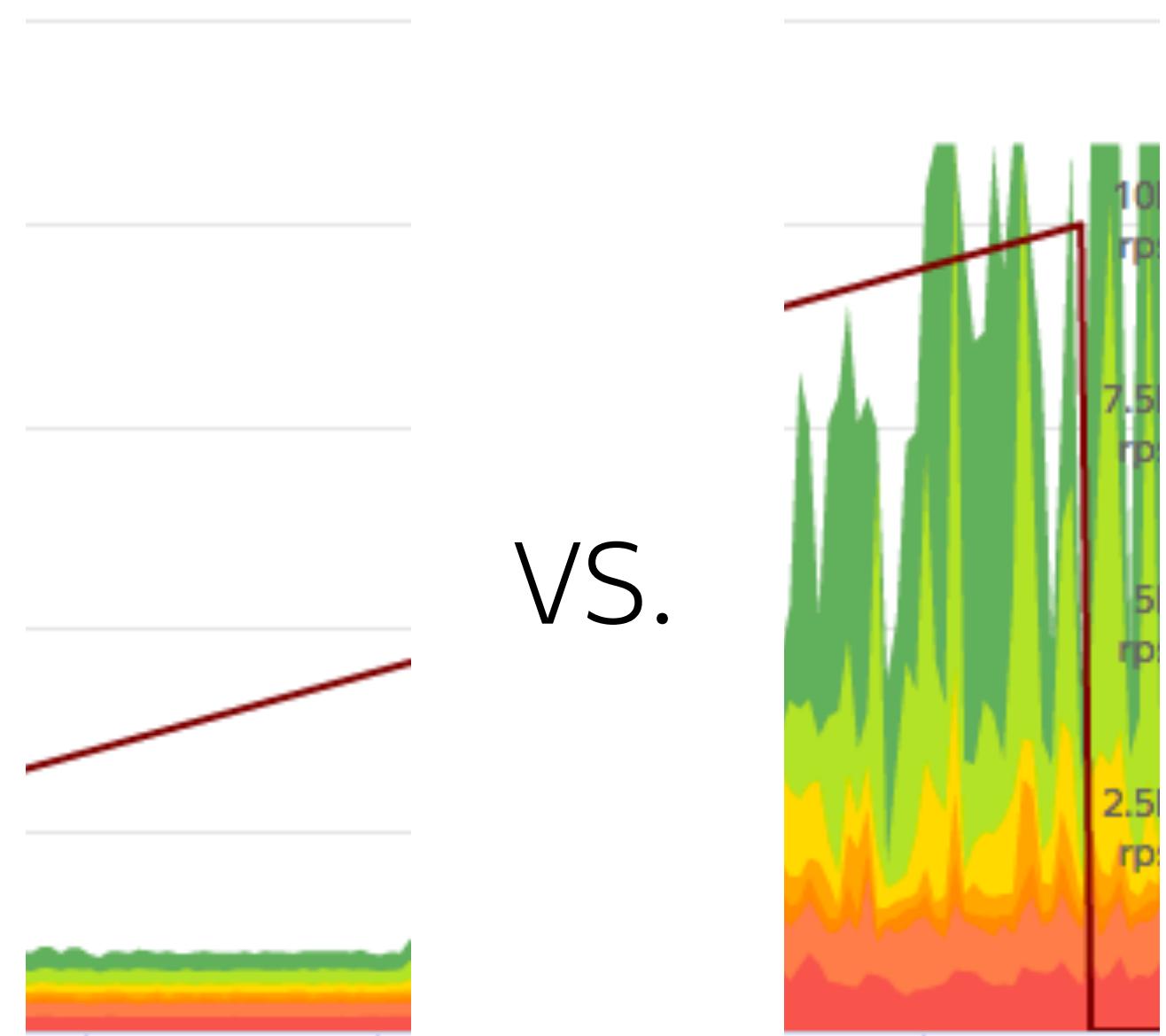
На графике в отчете – квантили за
каждую секунду теста



*если быть точным, то это процентиль, но по сути – разницы между ними нет

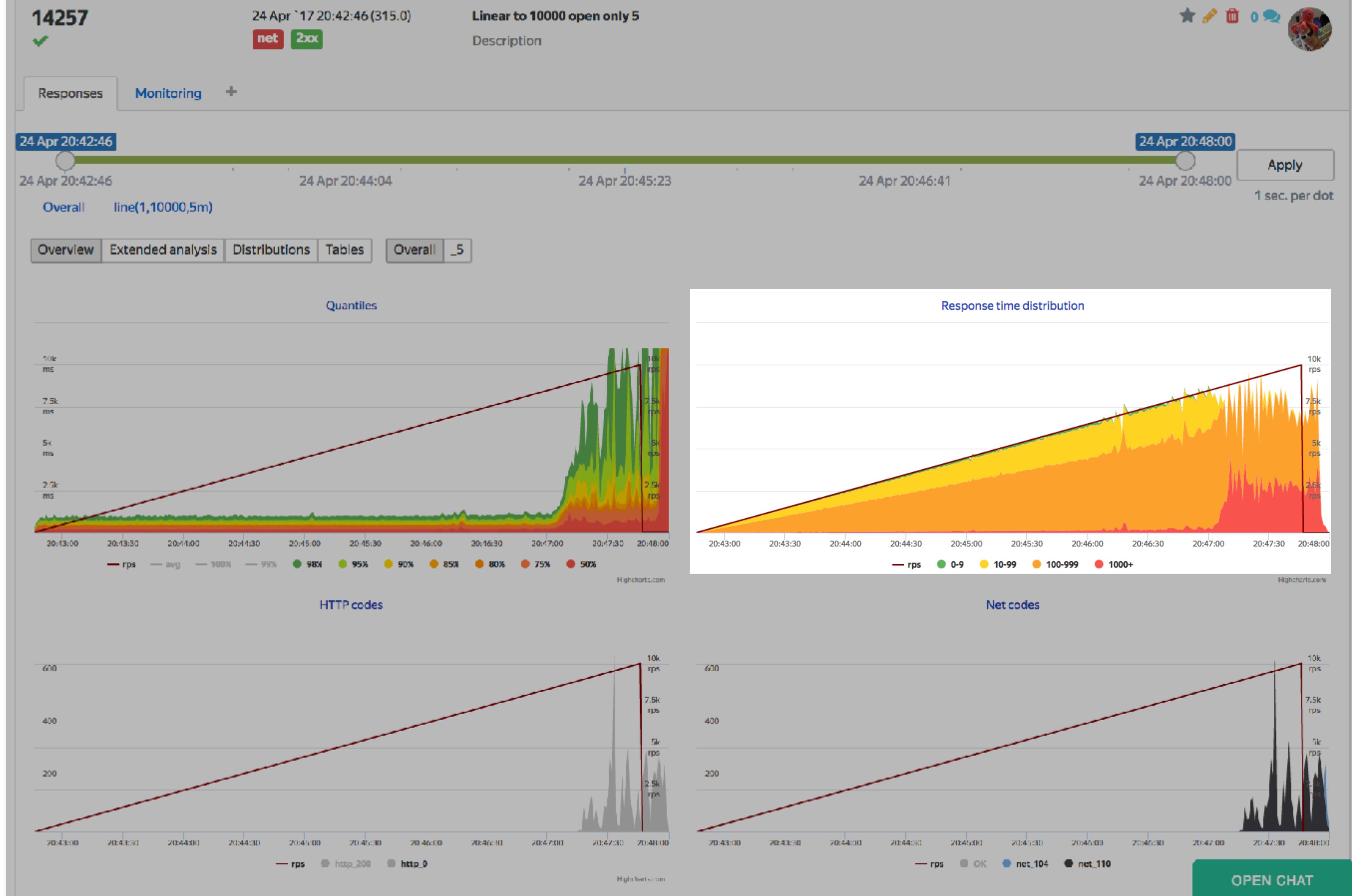
Чем отличаются фрагменты?

- › слева меньше RPS, чем справа
- › слева времена ответов меньше
- › слева времена ответов стабильнее

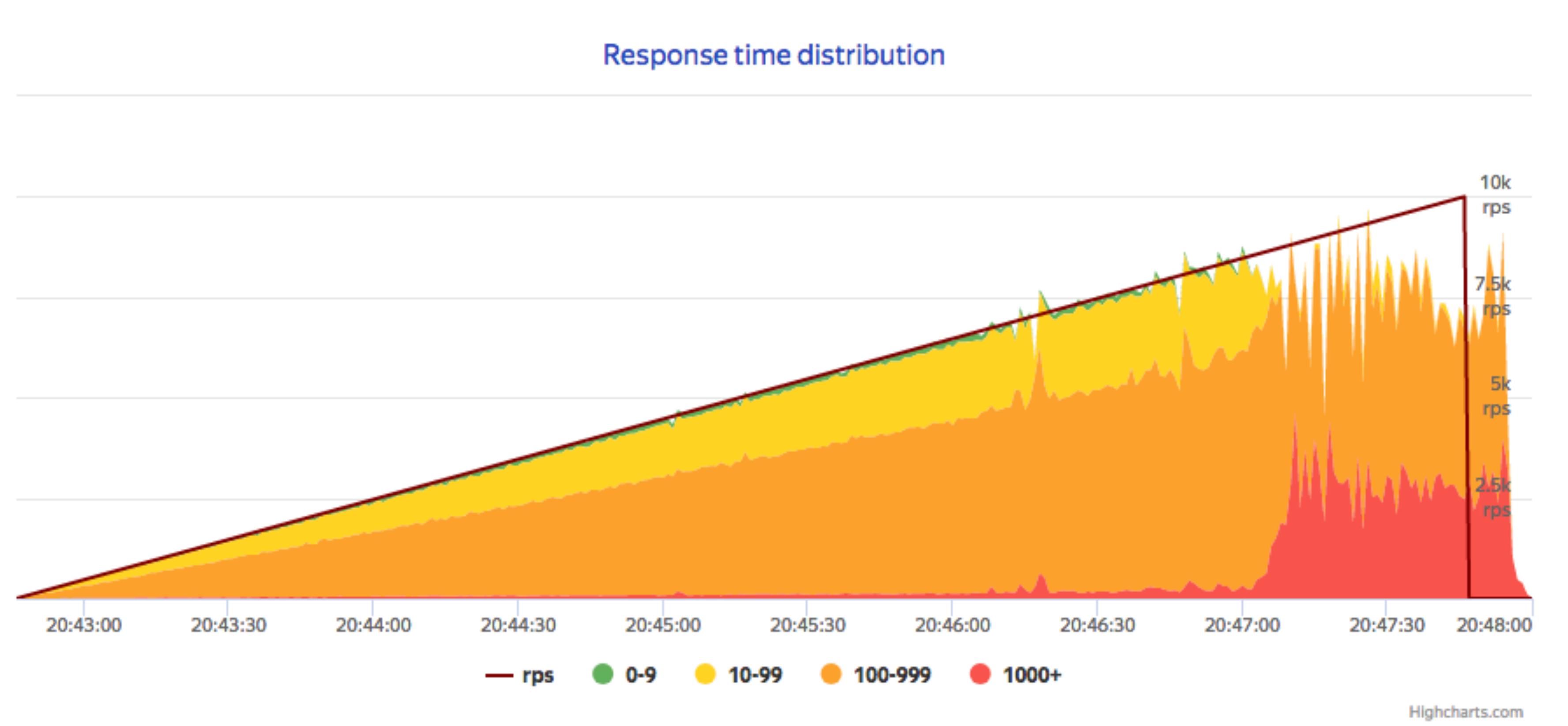




меньше – лучше!

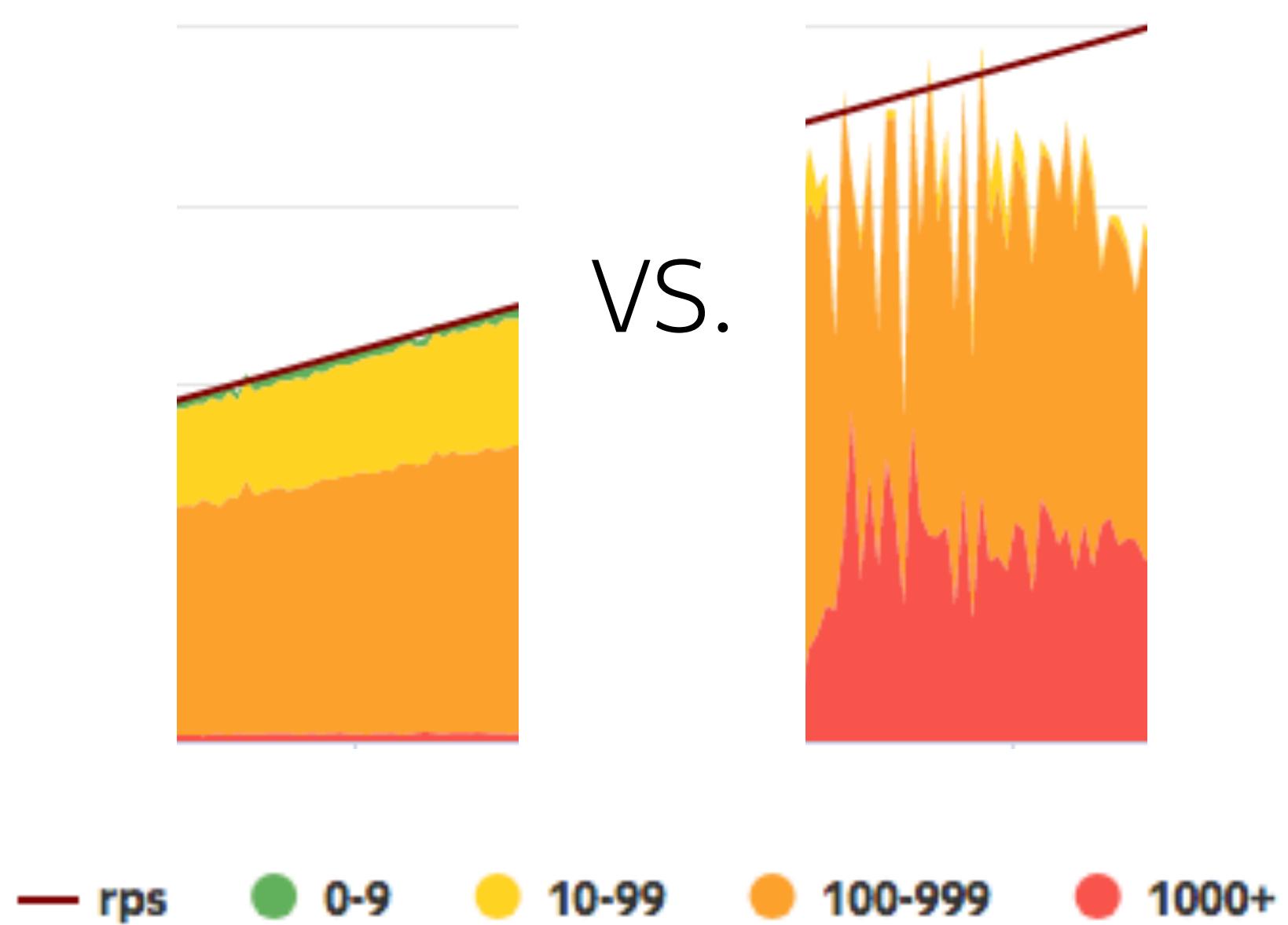


Распределение времен ответов



Чем отличаются фрагменты?

- › слева меньше нагрузка
- › слева стабильнее число ответов
- › справа число ответов **не дотягивает** до поданной нагрузки
- › справа больше медленных ответов и меньше быстрых





больше – лучше!

**14257**

24 Apr '17 20:42:46 (315.0)

net 2xx

Linear to 10000 open only 5

Description

[Responses](#)[Monitoring](#) +

24 Apr 20:42:46

24 Apr 20:42:46

24 Apr 20:44:04

24 Apr 20:45:23

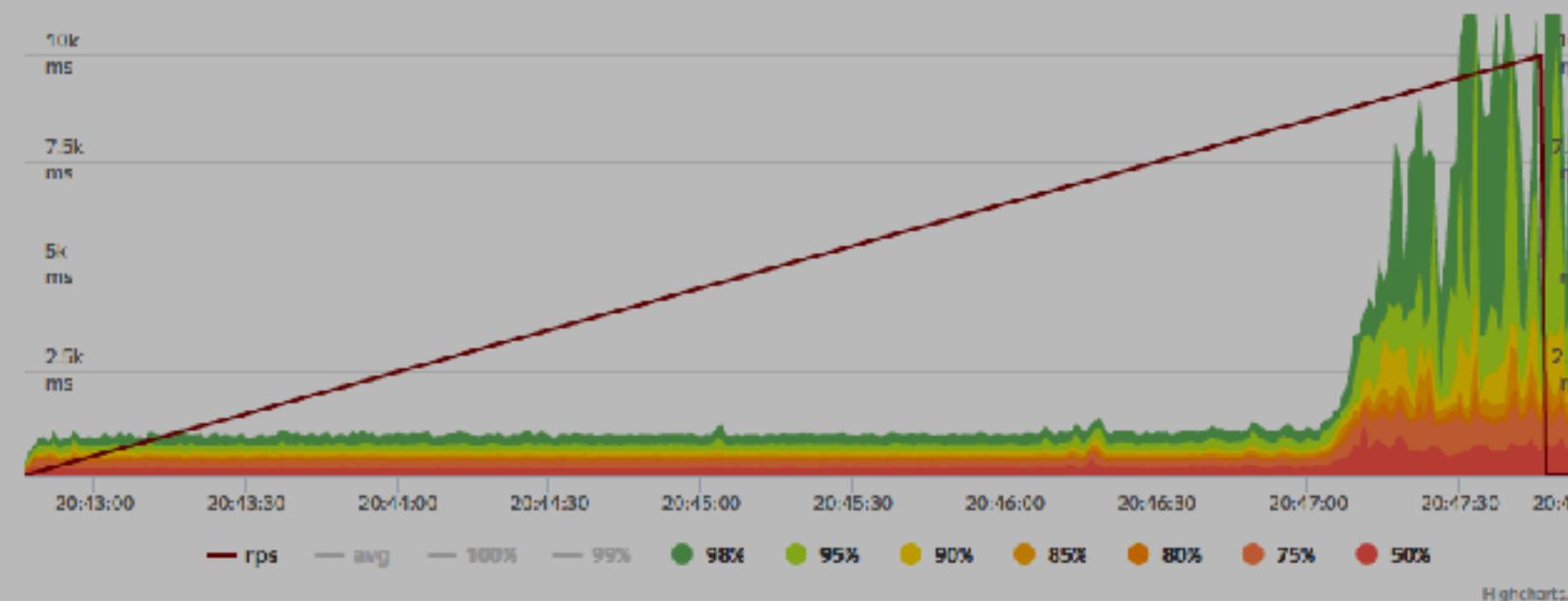
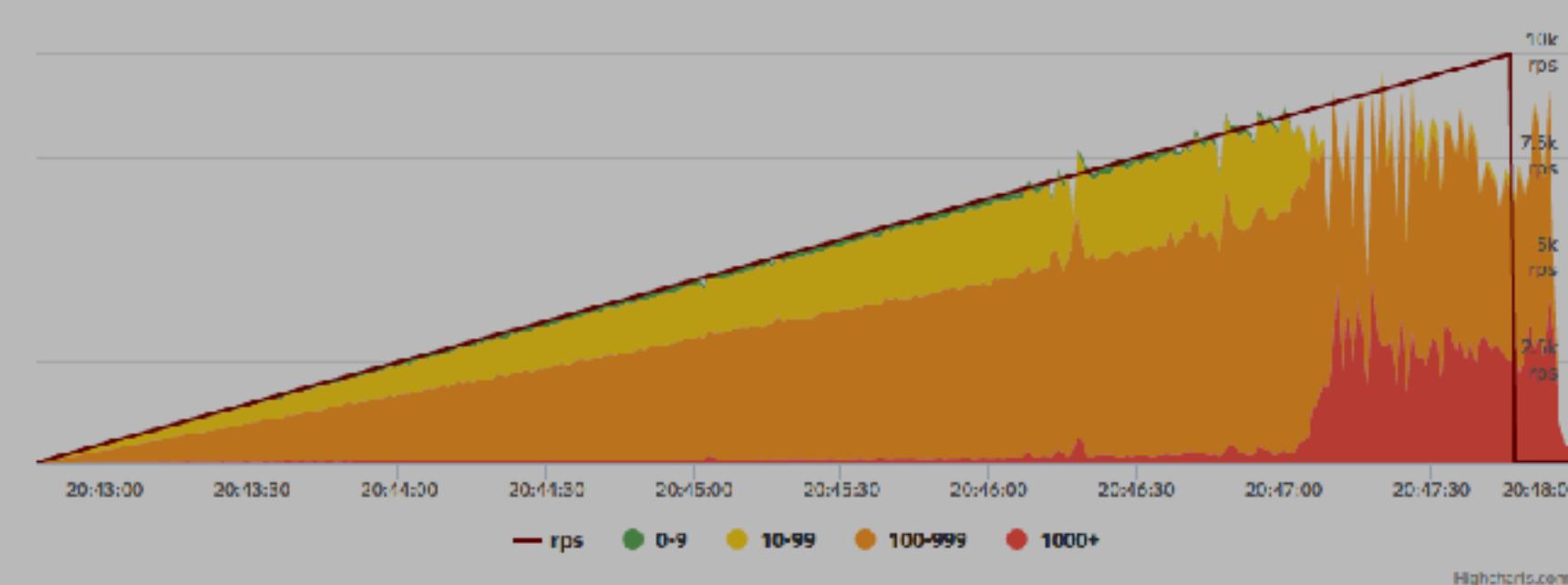
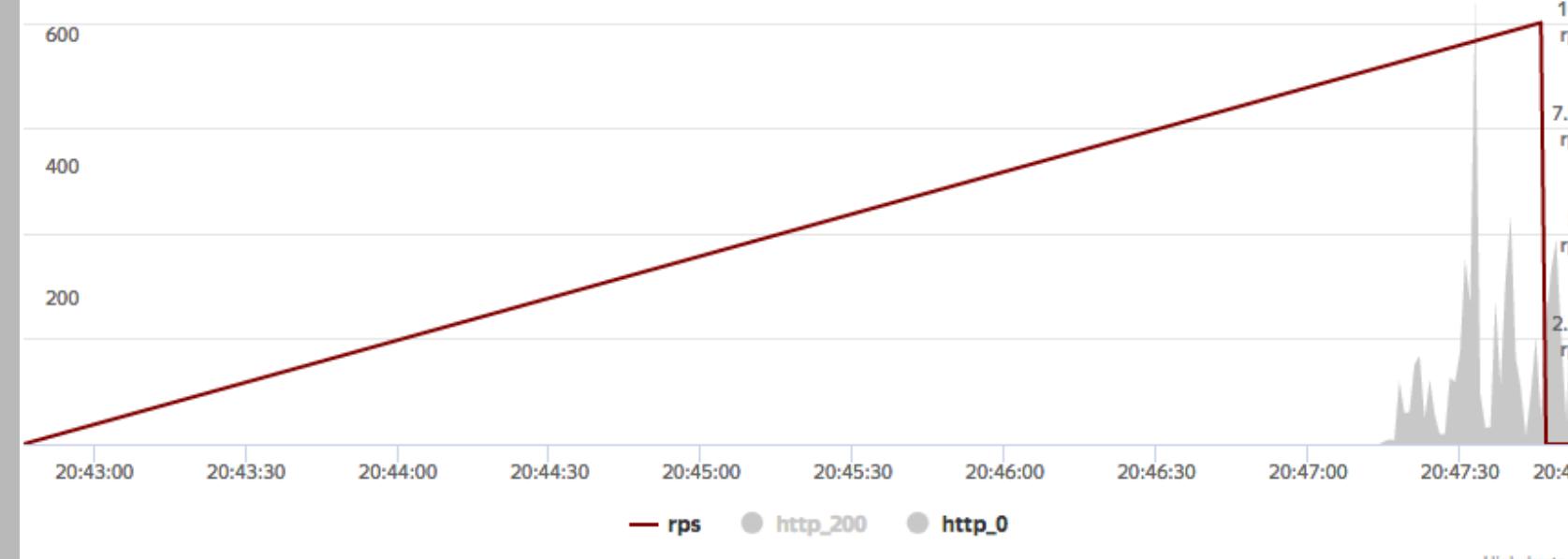
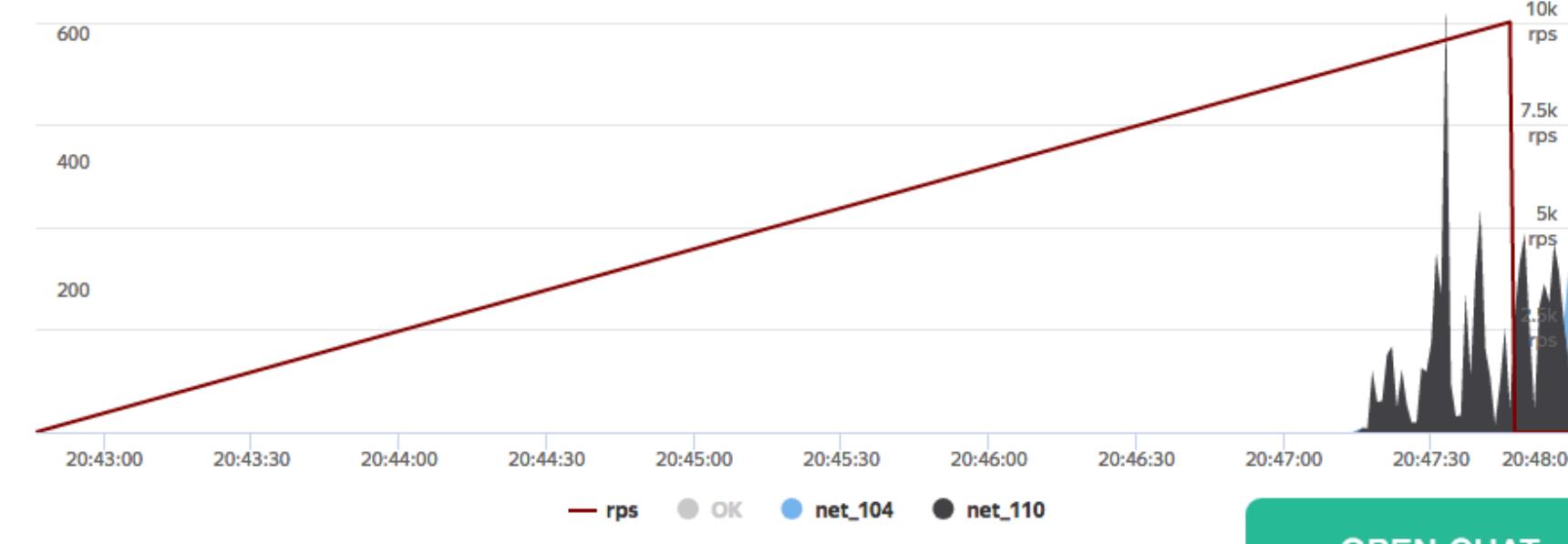
24 Apr 20:46:41

24 Apr 20:48:00

[Apply](#)

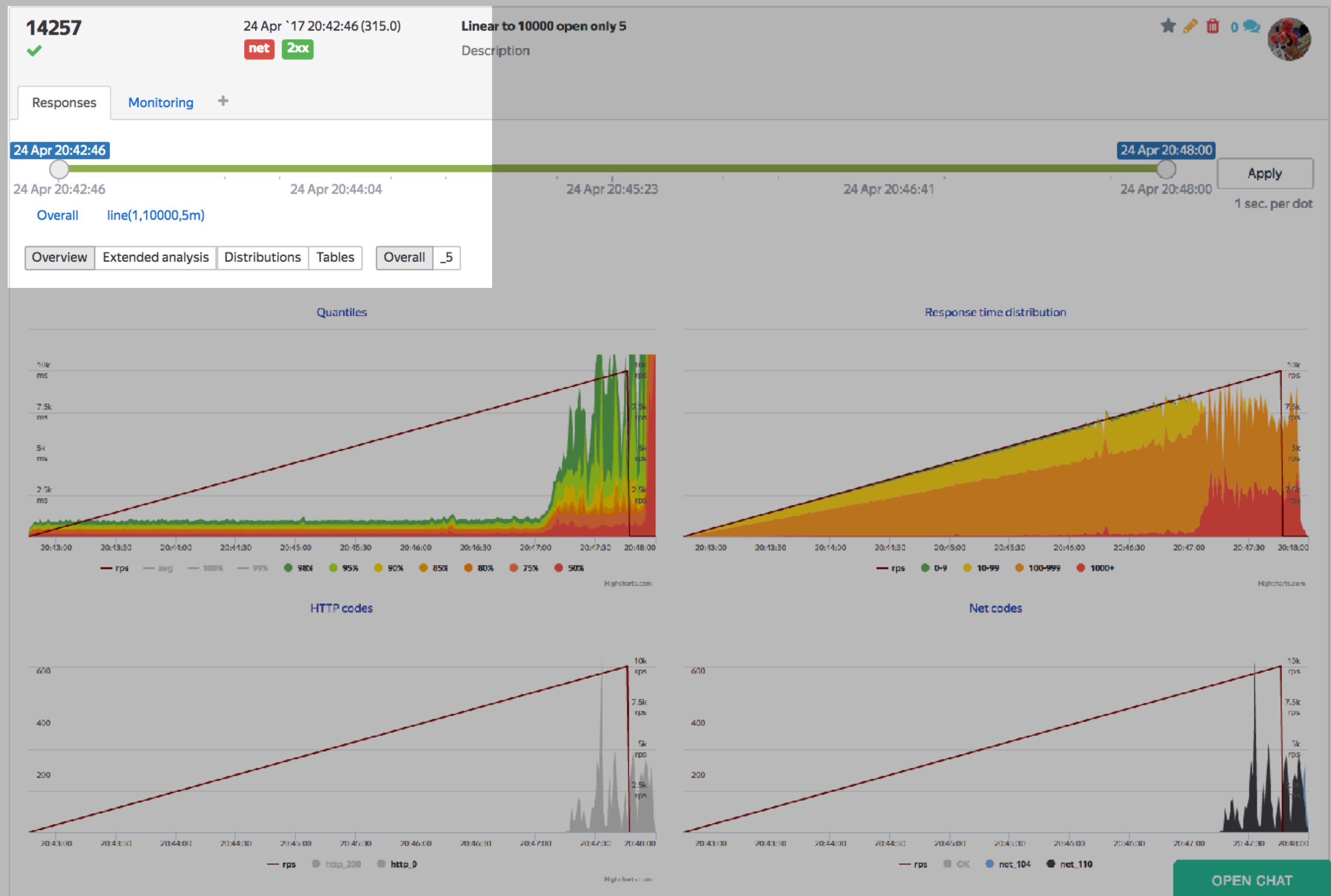
1 sec. per dot

Overall line(1,10000,5m)

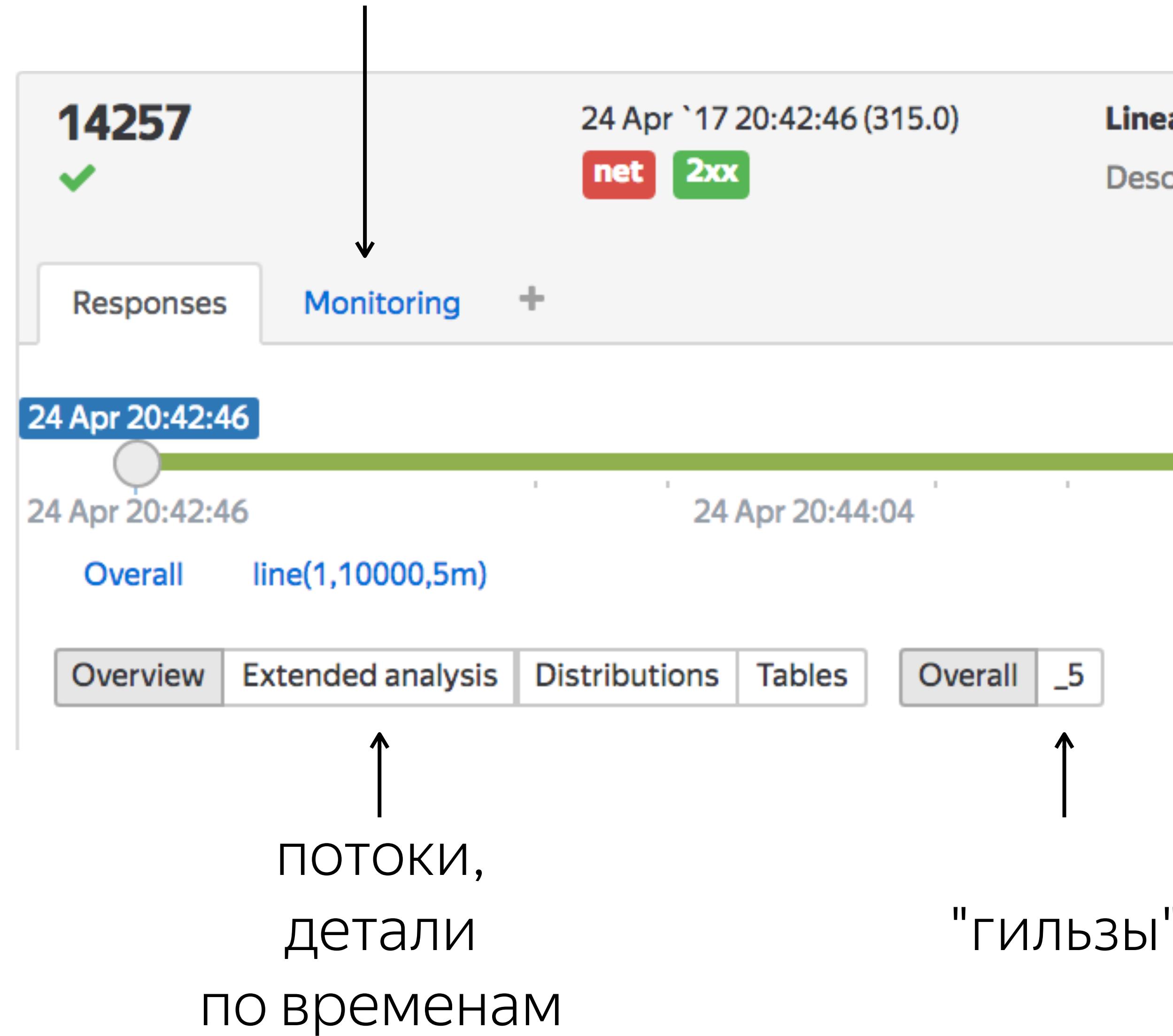
[Overview](#) [Extended analysis](#) [Distributions](#) [Tables](#) [Overall](#) [_5](#)[Quantiles](#)[Response time distribution](#)[HTTP codes](#)[Net codes](#)[OPEN CHAT](#)



ошибок быть не должно



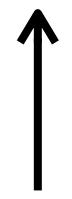
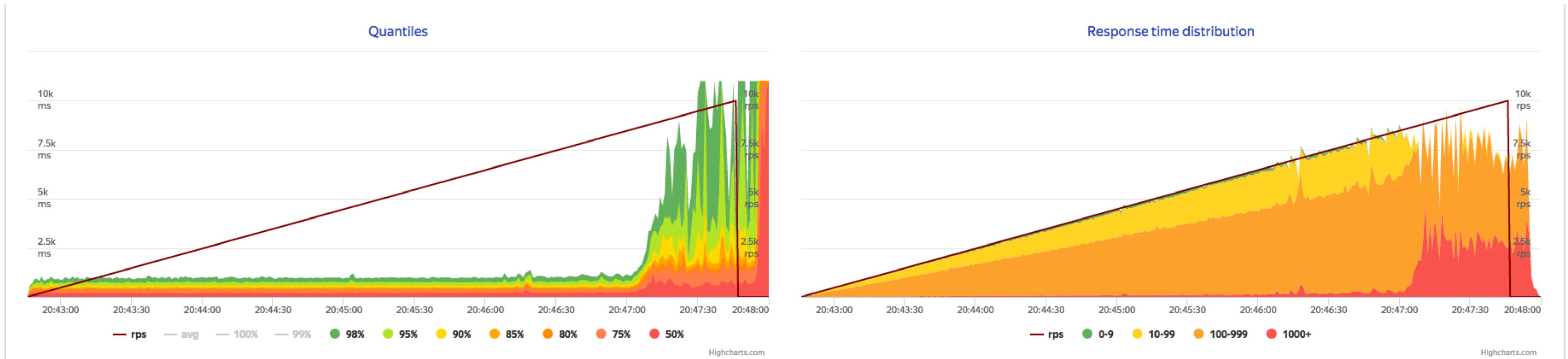
тут прячутся
графики мониторинга





приходите тыкать в них
после доклада

Итак:



времена ответов
меньше - лучше!



число ответов
больше - лучше!

С чего начать?



Готовим тестовые данные

У нас – просто набор URI. Бывает все намного сложнее.

| конфиг танка – это .ini файл с набором пар ключ-значение

```
[phantom]
uris=/1
/2
/3
/4
/5
```

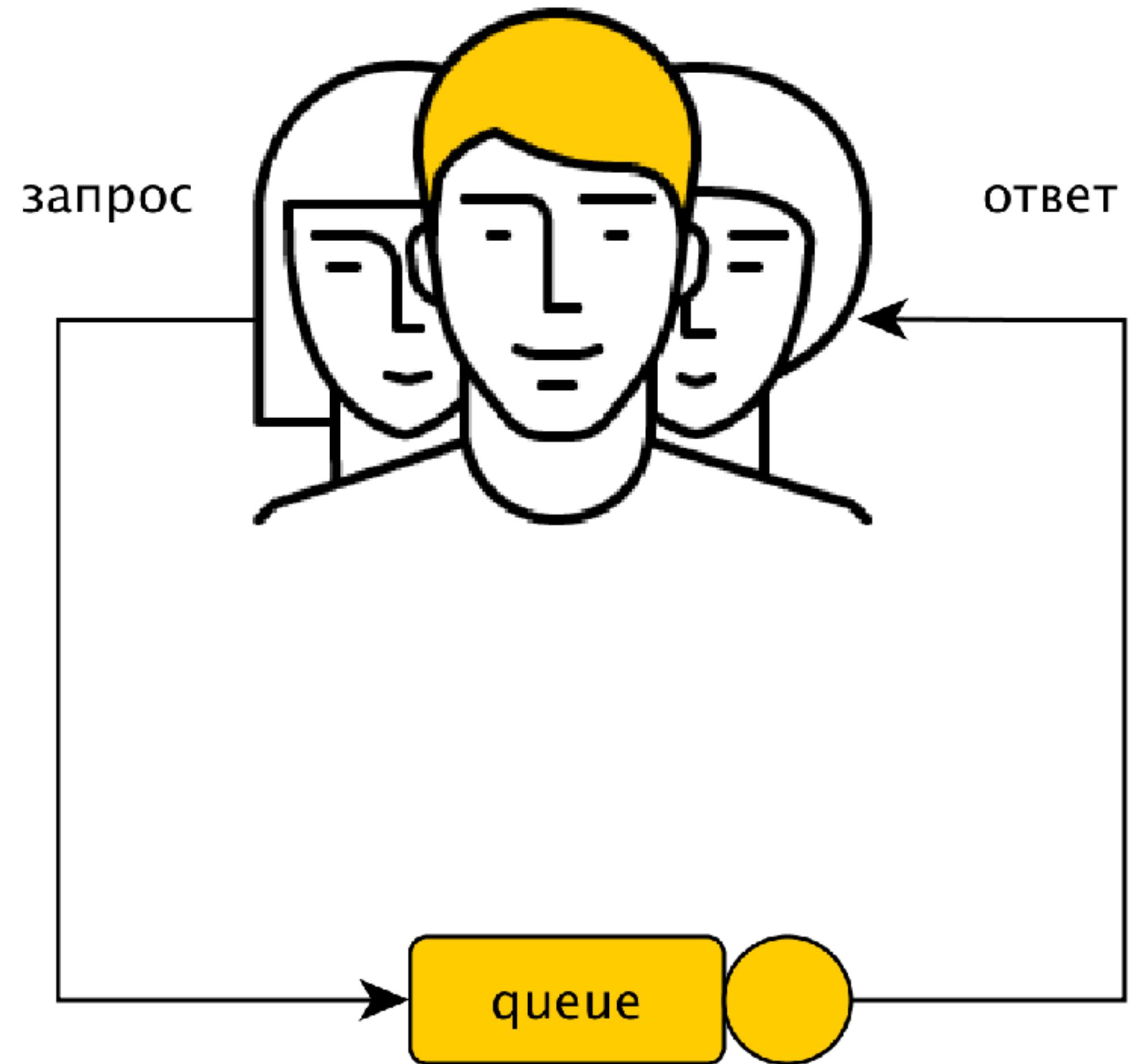
Как подавать нагрузку

- › управлять числом виртуальных пользователей. Стрельба "свободными инстансами"
- › управлять потоком запросов при неограниченном числе пользователей. В реальности – нужен достаточно большой пул для данных условий. Стрельба по "жесткому расписанию"

"Свободные инстансы"

В закрытых системах есть обратная связь, которая не дает генератору "добрить" сервис. Пользователи ждут ответа, если сервис перегружен.

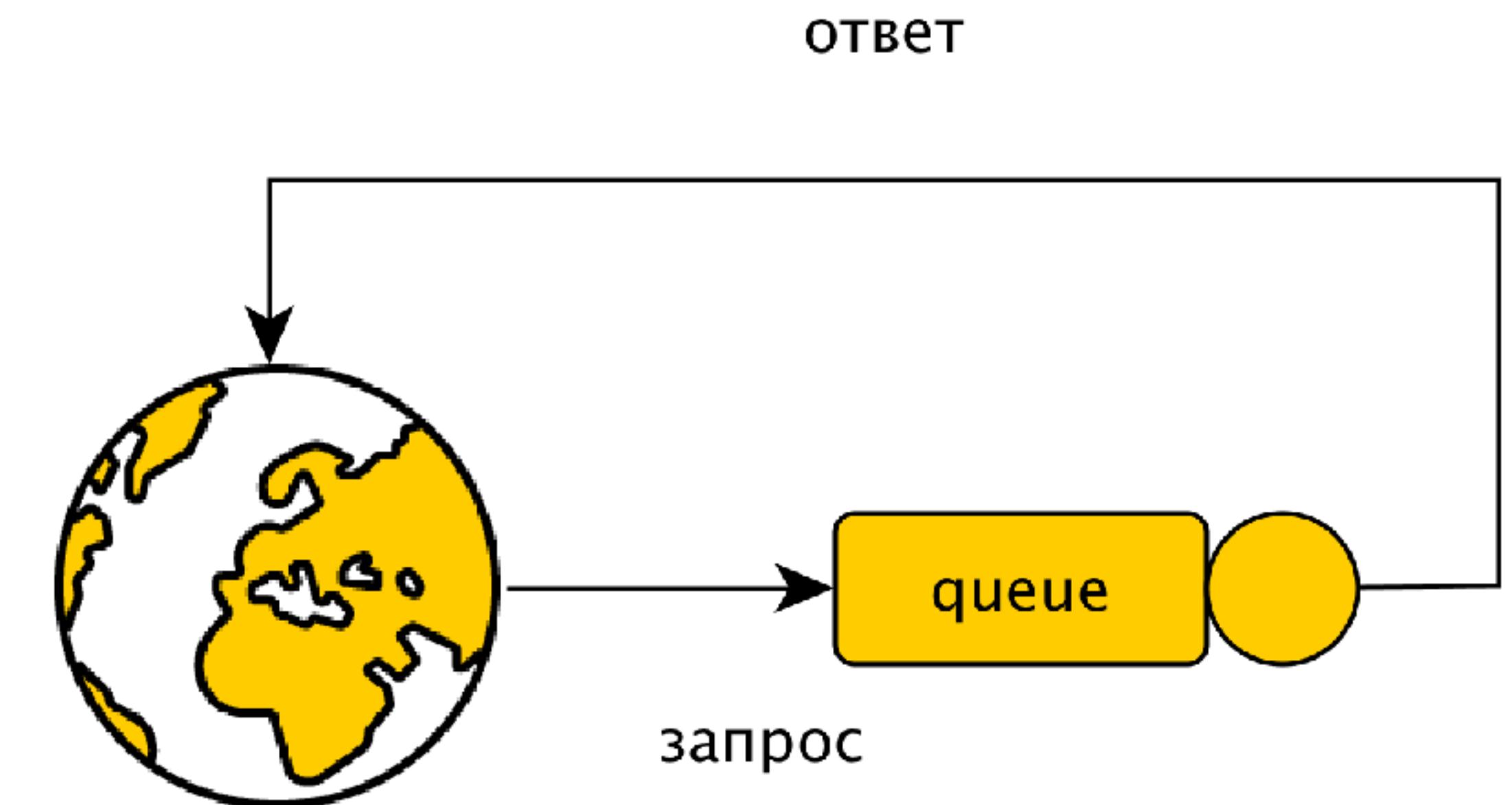
| 5 кассиров, 5 покупателей



Жесткое расписание

В открытых системах нет обратной связи, потому что пользователи не дожидаются ответов. Интернет – открытая система

| 5 кассиров, 500 покупателей





СКОЛЬКО ПОЛЬЗОВАТЕЛЕЙ
НУЖНО ДЛЯ ЭМУЛЯЦИИ
ОТКРЫТОЙ СИСТЕМЫ?

Производительность генератора

Формула Литтла

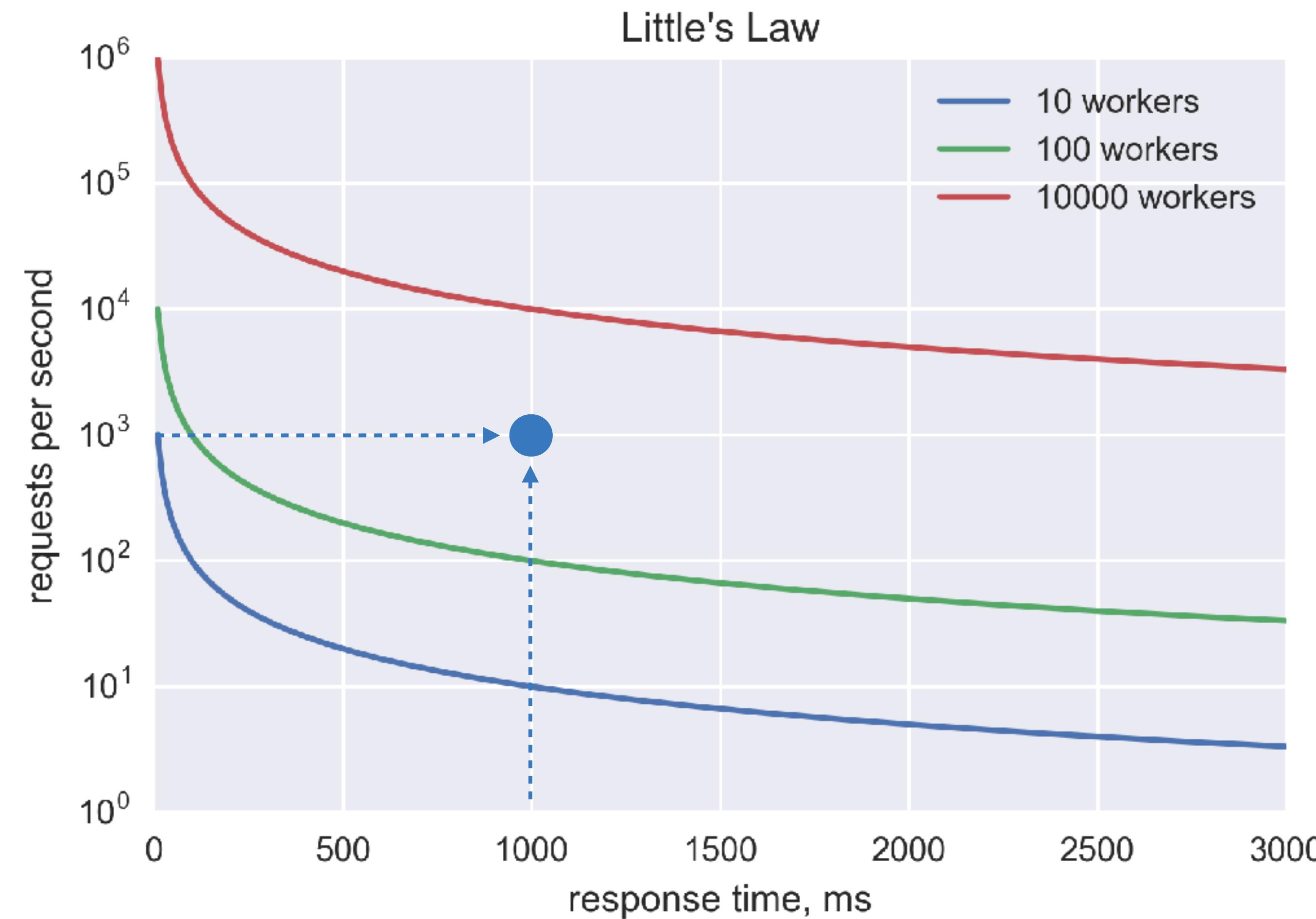
Среднее время обработки запроса –
Т миллисекунд.

Однопоточный генератор выдаст
1000 / Т запросов в секунду.

$$| \quad RPS = 1000 / T * workers$$

Пока не уперлись – можно считать,
что нагрузка не влияет на генератор.
Это открытая система.

Выбираем число инстансов





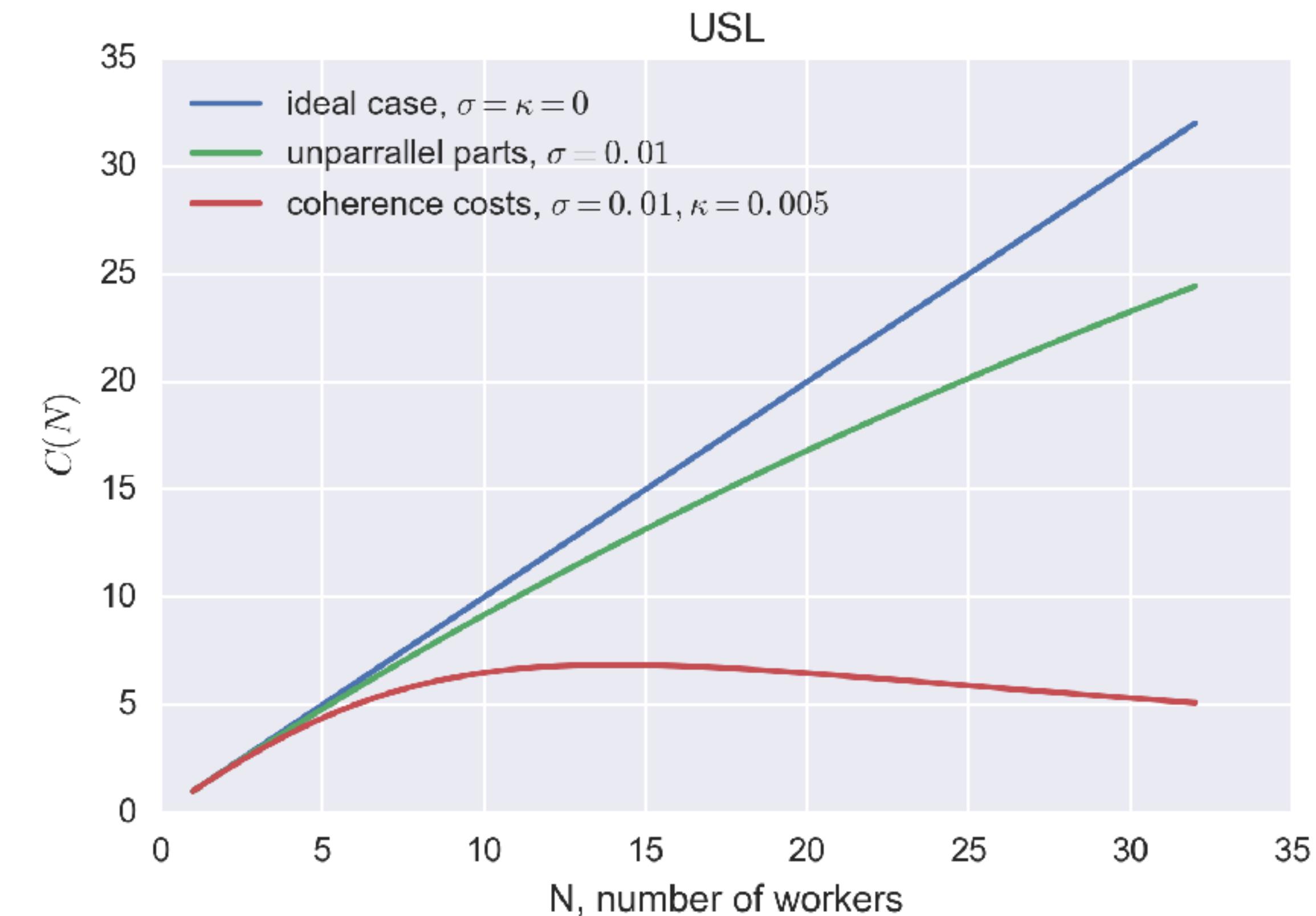
время ответа зависит
от числа параллельных
запросов

Кривая масштабируемости

больше непараллельных частей
– хуже масштабируемость (узкая
дверь в магазин)

больше затрат на обмен данными
– быстрее деградация (кассиры
болтают друг с другом)

[www.perfdynamics.com/
Manifesto/USLscalability.html](http://www.perfdynamics.com/Manifesto/USLscalability.html)



Какую нагрузку подавать

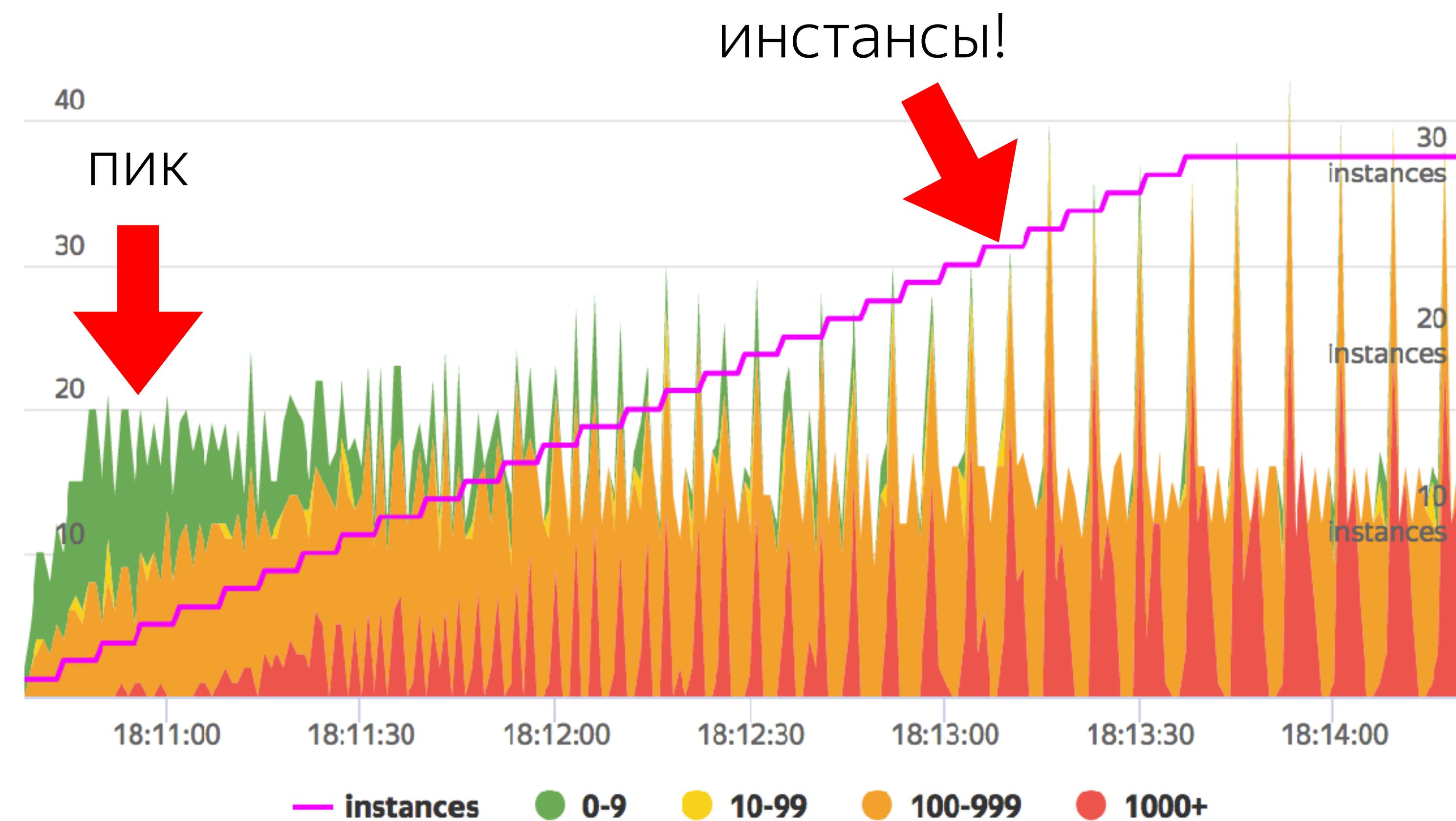
Начнем с эмуляции закрытой системы

| будем постепенно увеличивать число воркеров

```
[phantom]  
instances_schedule=line(1,30,3m)
```

overload.yandex.net/14229

Response time distribution



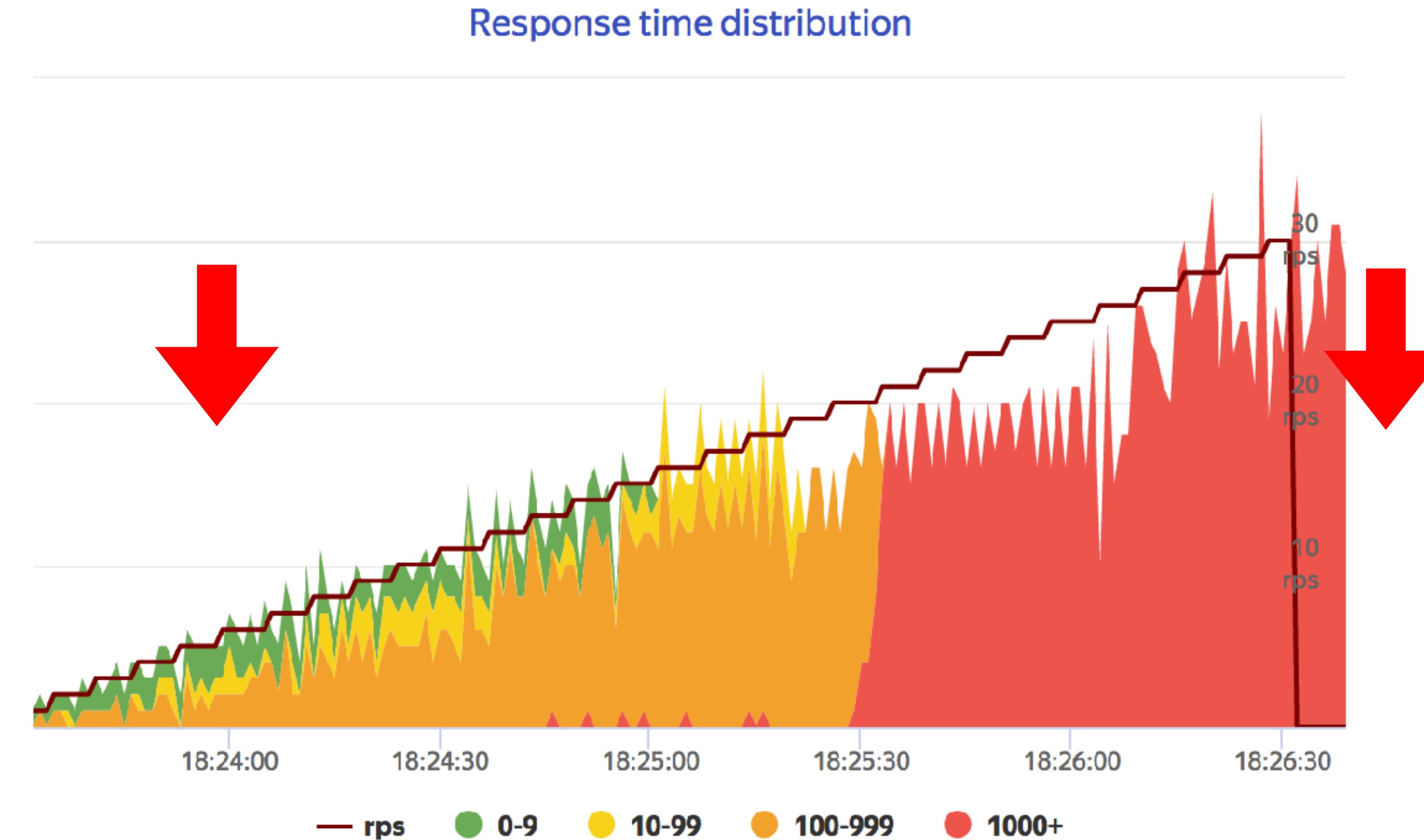
Проверяем в открытой модели

Открытая модель "жестче" по отношению к сервису. Поэтому
свалиться он может раньше

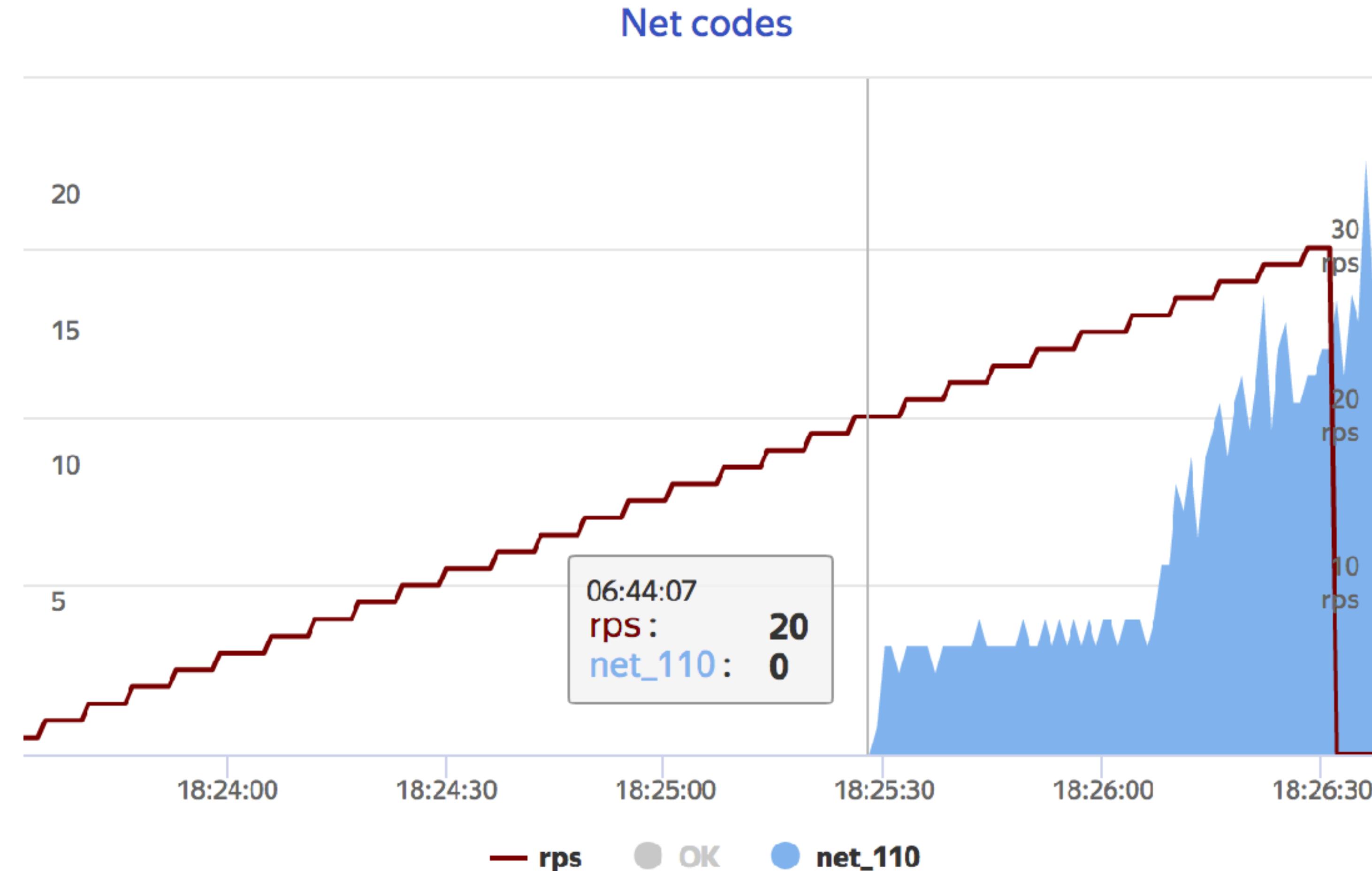
| предел мы выяснили в прошлом teste

```
[phantom]  
rps_schedule=line(1,30,3m)
```

overload.yandex.net/14230



overload.yandex.net/14230





ярко выраженная
разладка, легко
настроить автостоп

Сравнение моделей нагрузки

Закрытая модель

› instances_schedule

Открытая модель

› rps_schedule

Сравнение моделей нагрузки

Закрытая модель

- › instances_schedule
- › розовая линия – число инстансов

Открытая модель

- › rps_schedule
- › красная линия – число запросов в секунду

Сравнение моделей нагрузки

Закрытая модель

- › `instances_schedule`
- › **розовая линия – число инстансов**
- › **расписание задает число инстансов**

Открытая модель

- › `rps_schedule`
- › **красная линия – число запросов в секунду**
- › **расписание задает число запросов в секунду**

Сравнение моделей нагрузки

Закрытая модель

- › instances_schedule
- › розовая линия – число инстансов
- › расписание задает число инстансов
- › от сервиса зависит число запросов в секунду

Открытая модель

- › rps_schedule
- › красная линия – число запросов в секунду
- › расписание задает число запросов в секунду
- › от сервиса зависит используемое число инстансов

Сравнение моделей нагрузки

Закрытая модель

пристреляться,
оценить предел
производительности

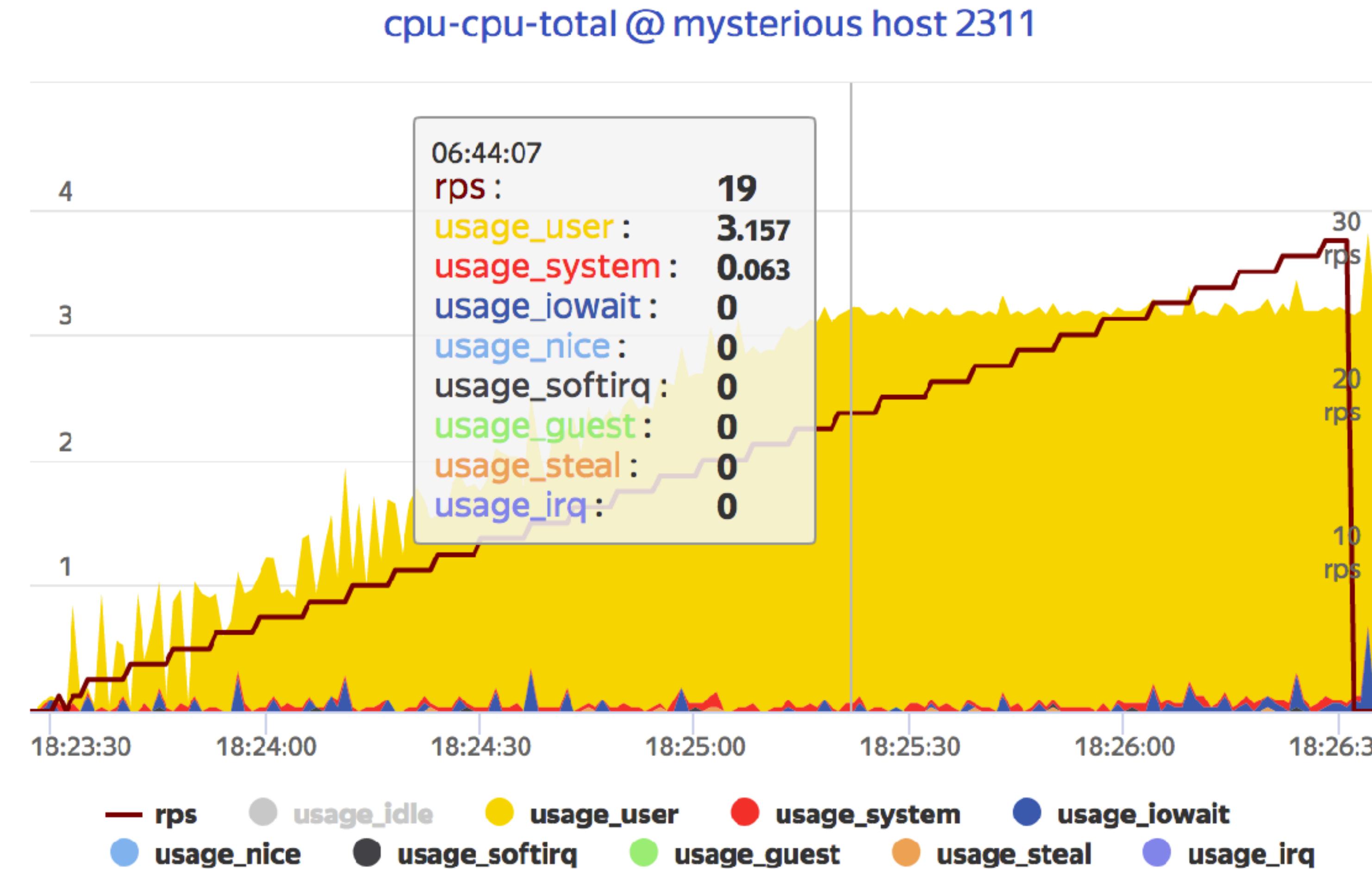
Открытая модель

автоматически определить
предел производительности,
обнаружить узкие места

Анализ системных ресурсов



overload.yandex.net/14230





используется только
одно ядро процессора

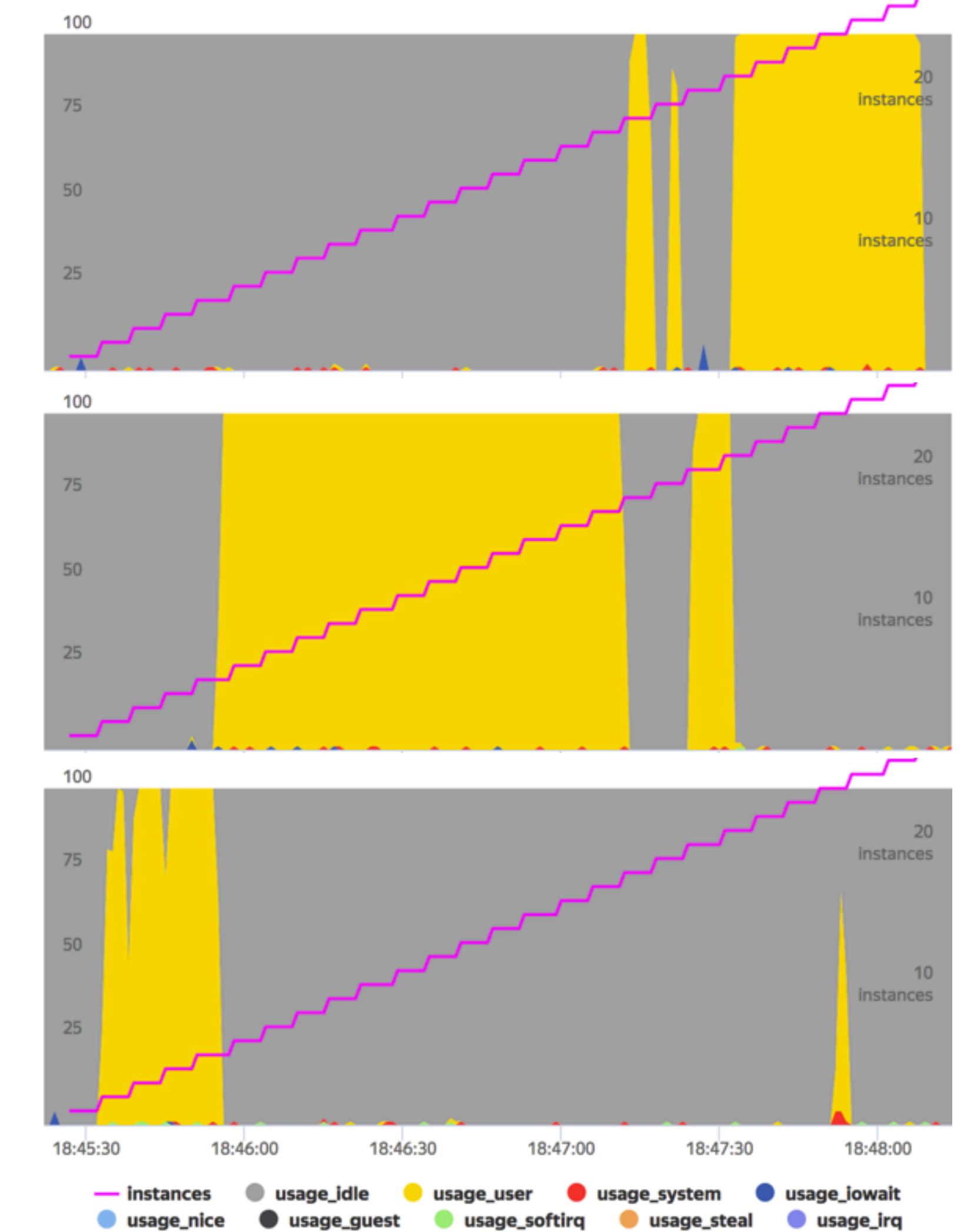
Мониторим процессор поядерно

Удостоверимся, что дело действительно в этом

```
<CPU regcru="true"/>
```

overload.yandex.net/14235

Воркер пересекивает с ядра на ядро, но в каждый момент времени загружено только одно



Чиним воркеры

Увеличиваем число воркеров. В Tornado это тривиально. И ставим воркеров по числу ядер

#было:

```
server.start()
```

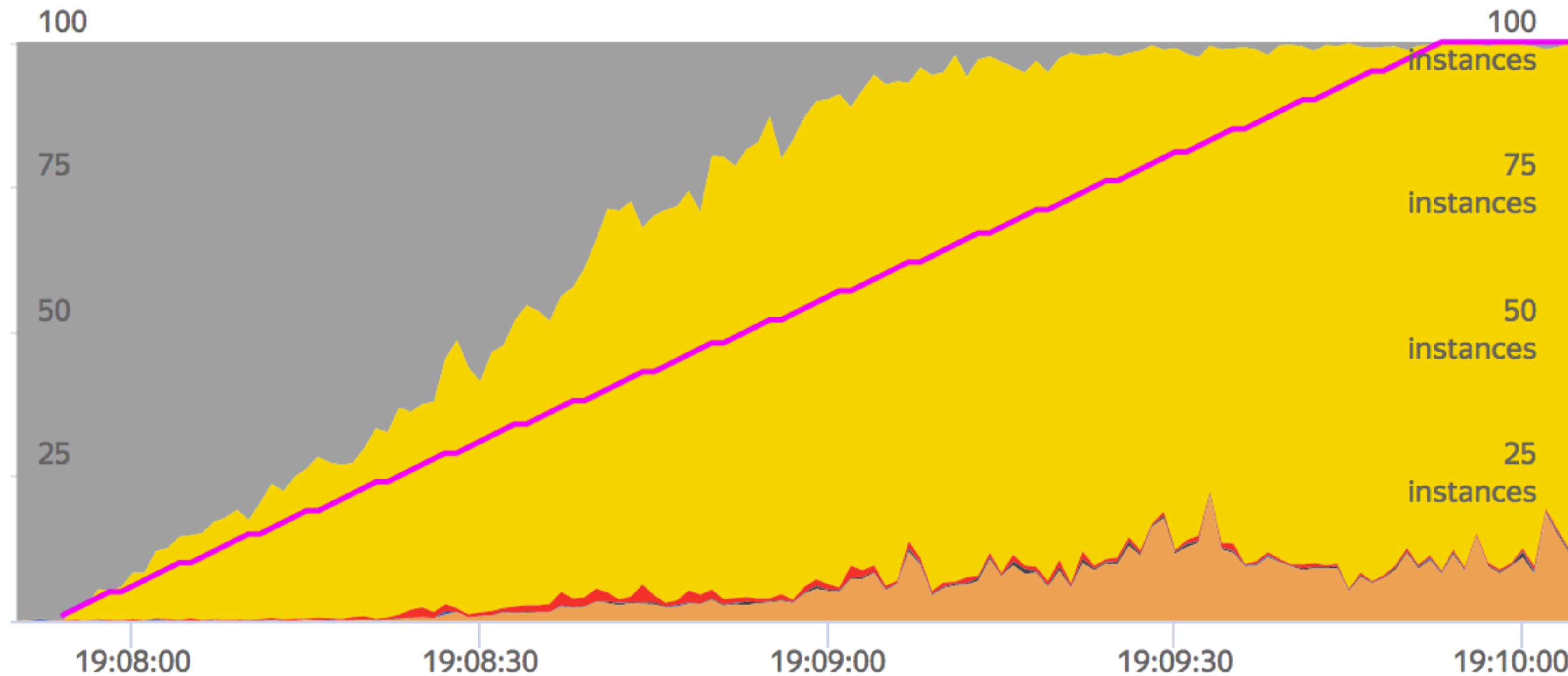
#стало:

```
server.start(args.workers)
```

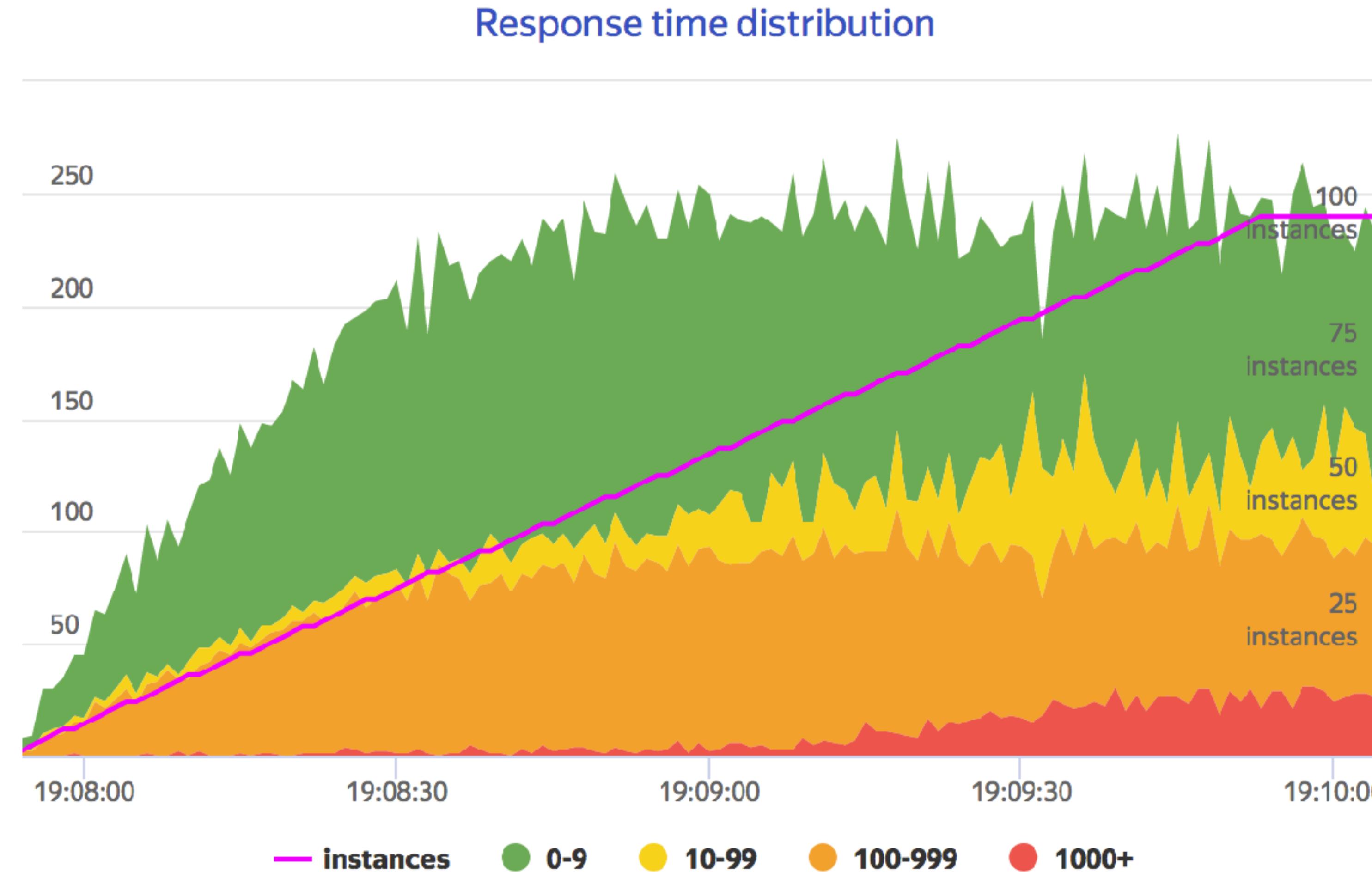
noob	1011616	0.1	0.0	1182820	32264	pts/2	S+	May02	21:25	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011617	0.1	0.0	1182864	32304	pts/2	S+	May02	21:23	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011618	0.1	0.0	1182876	32320	pts/2	S+	May02	21:26	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011619	0.1	0.0	1182940	32404	pts/2	S+	May02	21:19	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011620	0.1	0.0	1182788	32356	pts/2	S+	May02	21:30	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011621	0.1	0.0	1182876	32328	pts/2	S+	May02	21:25	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011622	0.1	0.0	1182596	32032	pts/2	S+	May02	21:32	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011623	0.1	0.0	1182768	32208	pts/2	S+	May02	21:24	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011624	0.1	0.0	1182792	32244	pts/2	S+	May02	21:23	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011625	0.1	0.0	1182788	32228	pts/2	S+	May02	21:26	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011626	0.1	0.0	1182828	32284	pts/2	S+	May02	21:28	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011627	0.1	0.0	1182860	32300	pts/2	S+	May02	21:27	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011628	0.1	0.0	1182796	32236	pts/2	S+	May02	21:31	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011629	0.1	0.0	1182852	32292	pts/2	S+	May02	21:34	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011630	0.1	0.0	1182836	32276	pts/2	S+	May02	21:25	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011631	0.1	0.0	1182804	32372	pts/2	S+	May02	21:37	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011632	0.1	0.0	1182920	32404	pts/2	S+	May02	21:28	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011633	0.1	0.0	1183000	32472	pts/2	S+	May02	21:36	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011634	0.1	0.0	1182840	32304	pts/2	S+	May02	21:35	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011635	0.1	0.0	1182916	32356	pts/2	S+	May02	21:39	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011636	0.1	0.0	1182900	32340	pts/2	S+	May02	21:45	/usr/bin/python	/usr/local/bin/target	--workers	32
noob	1011637	0.1	0.0	1182952	32408	pts/2	S+	May02	21:47	/usr/bin/python	/usr/local/bin/target	--workers	32

overload.yandex.net/14242

cpu-cpu-total @ mysterious host 2311



overload.yandex.net/14242



Анализ тяжести запросов



Разбиваем результаты по ручкам

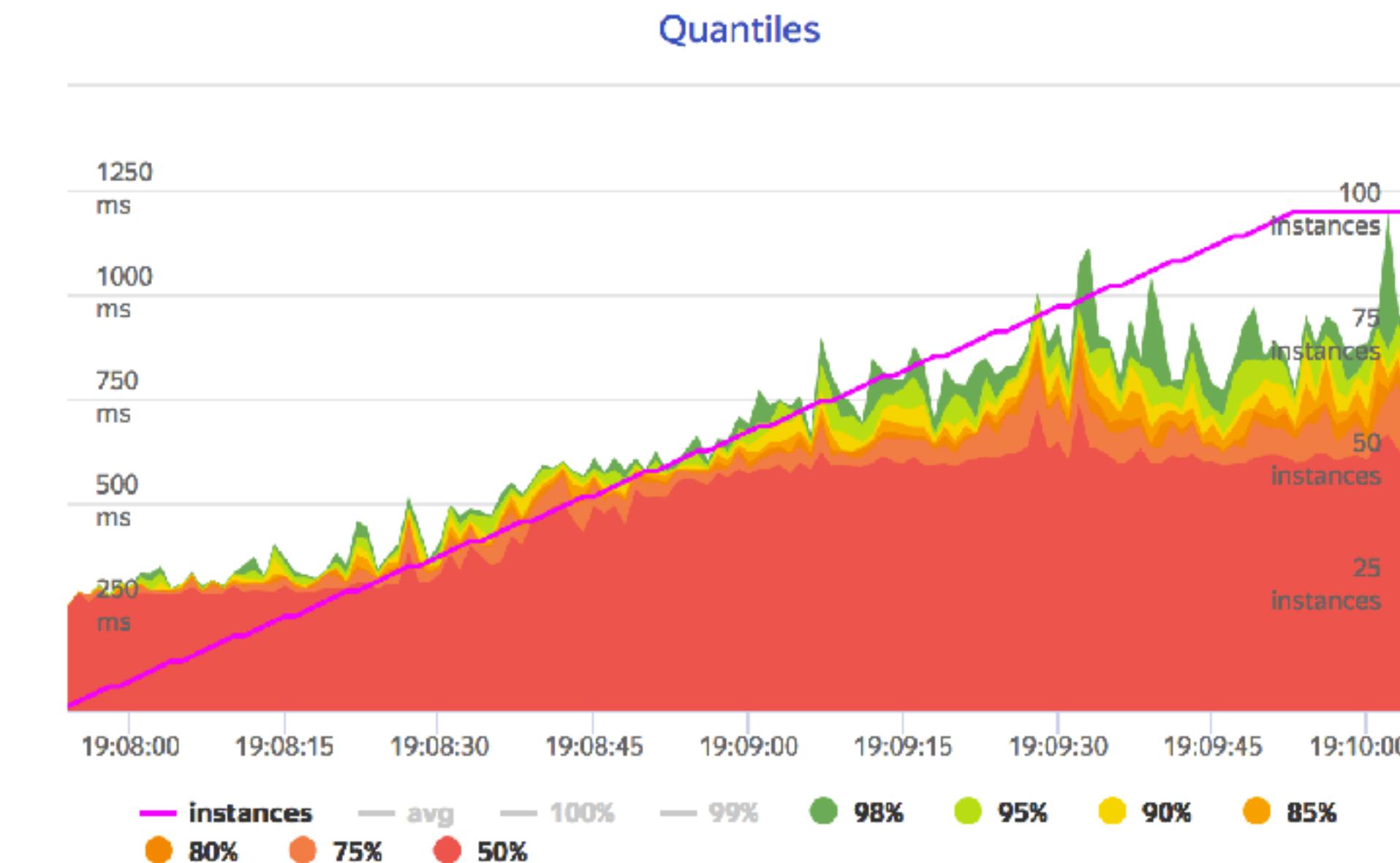
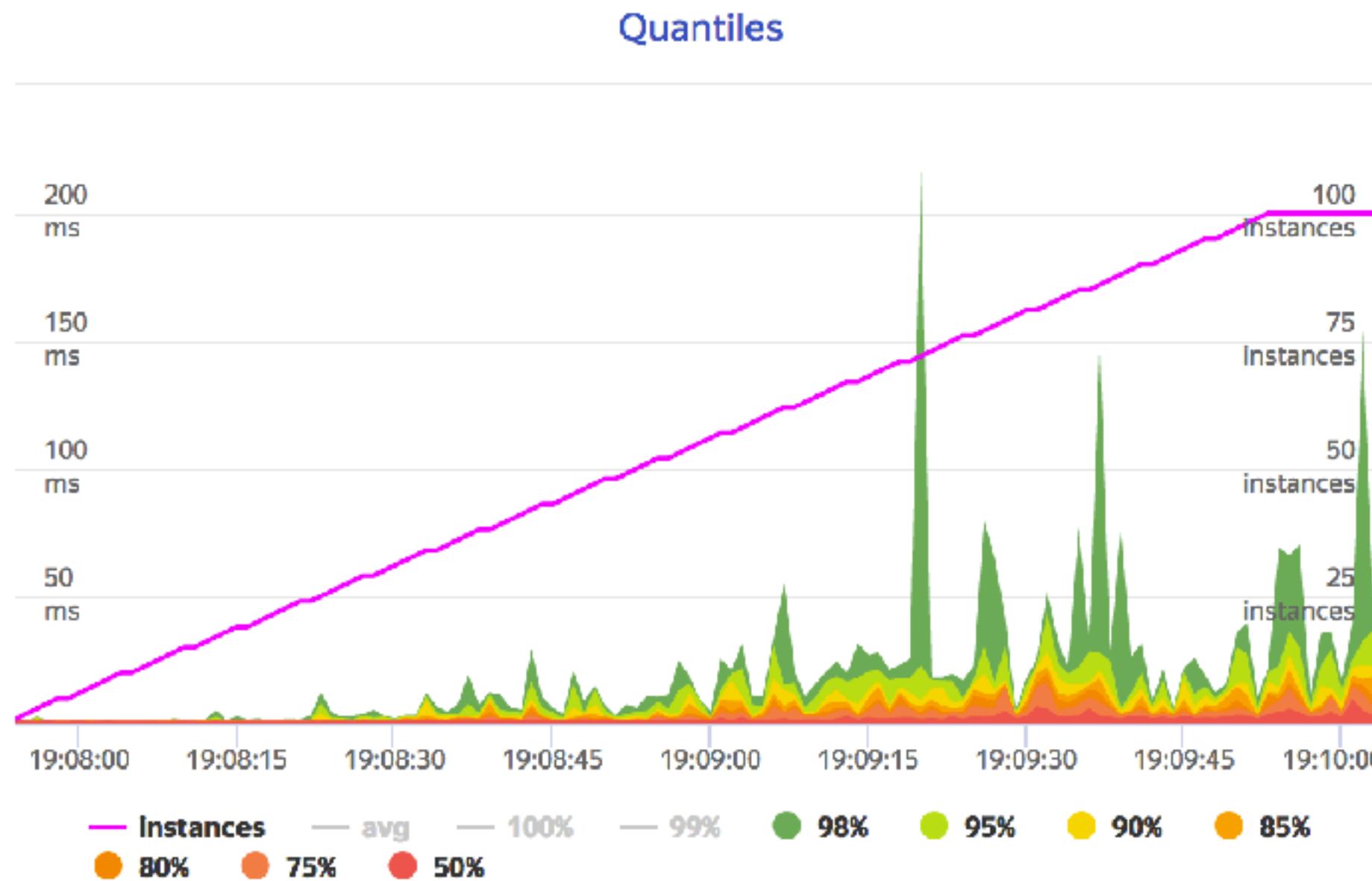
Танк автоматически маркирует запросы в разные ручки

гильзы генерятся из URL

```
[phantom]
```

```
autocases=1
```

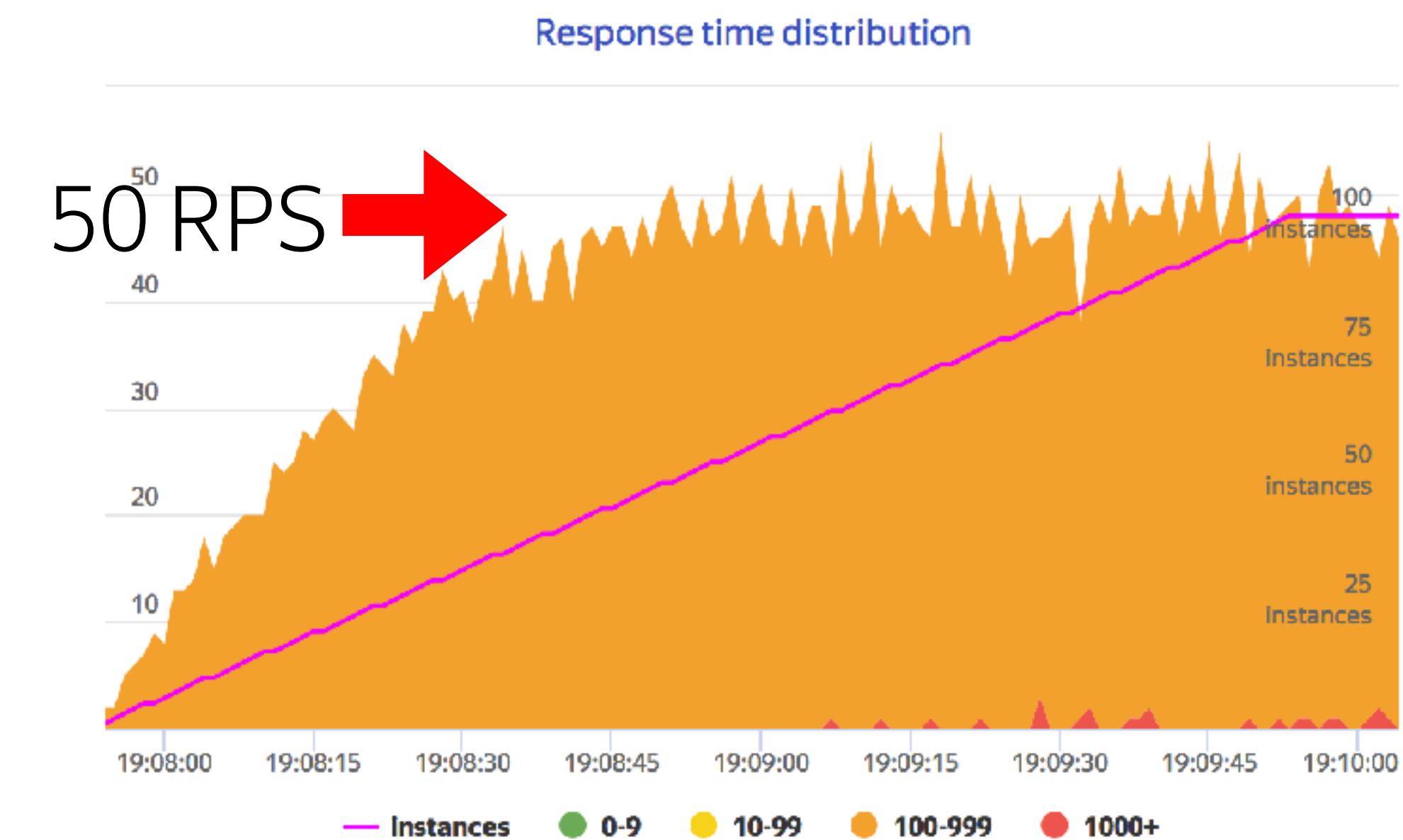
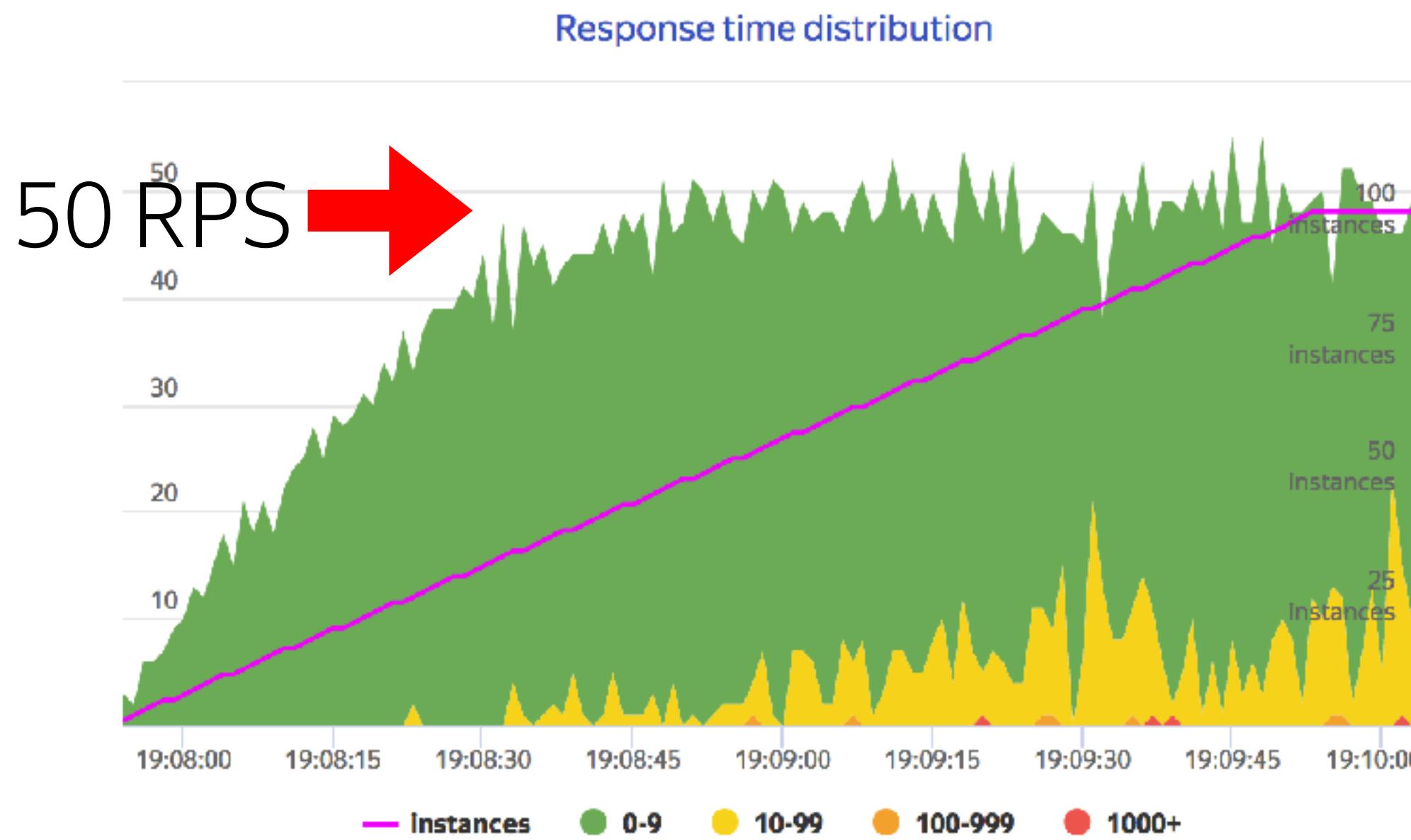
overload.yandex.net/14242



/1

/2

overload.yandex.net/14242



/1

/2

Нагрузочный тестировщик заходит в бар...

У нас есть "тяжелые" и "легкие" запросы, но RPS у всех одинаковый. Потому что мы их берем из одной очереди



Тяжелые и легкие запросы



Коктейль, 2 минуты



Виски, 10 секунд



тяжелые и легкие
запросы в разных
очередях

Независимые очереди запросов

Заводим по одной секции на каждый тип запросов

```
[phantom]
```

```
uris=/1
```

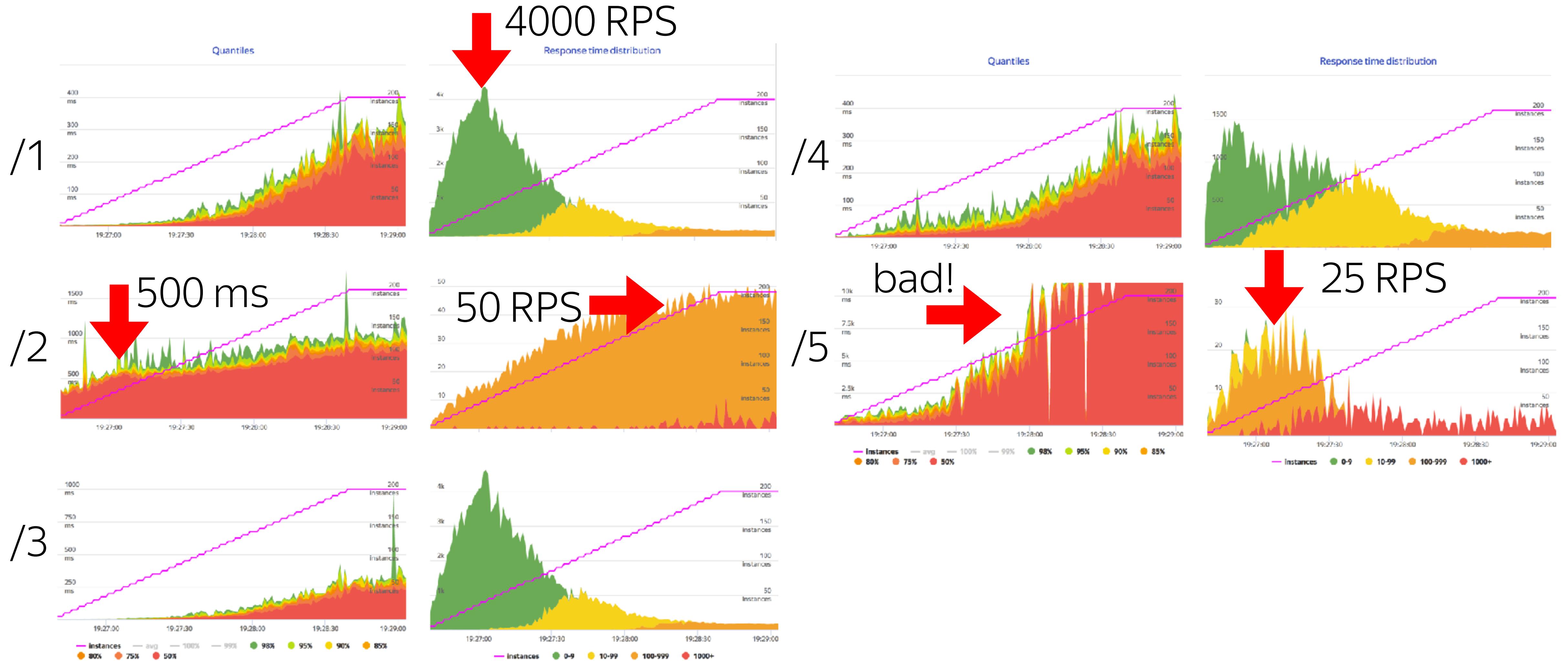
```
...
```

```
[phantom-1]
```

```
uris=/2
```

```
...
```

overload.yandex.net/14245

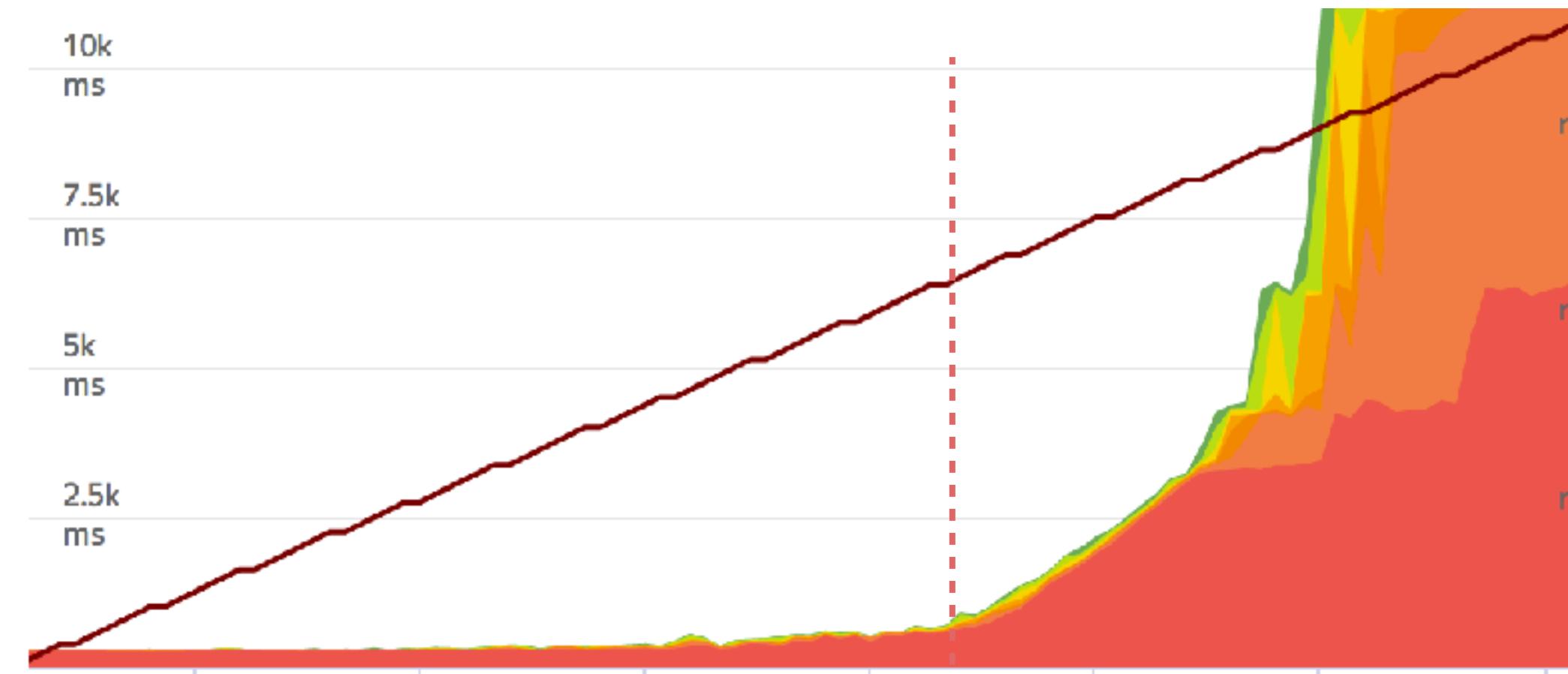


Ручка #2

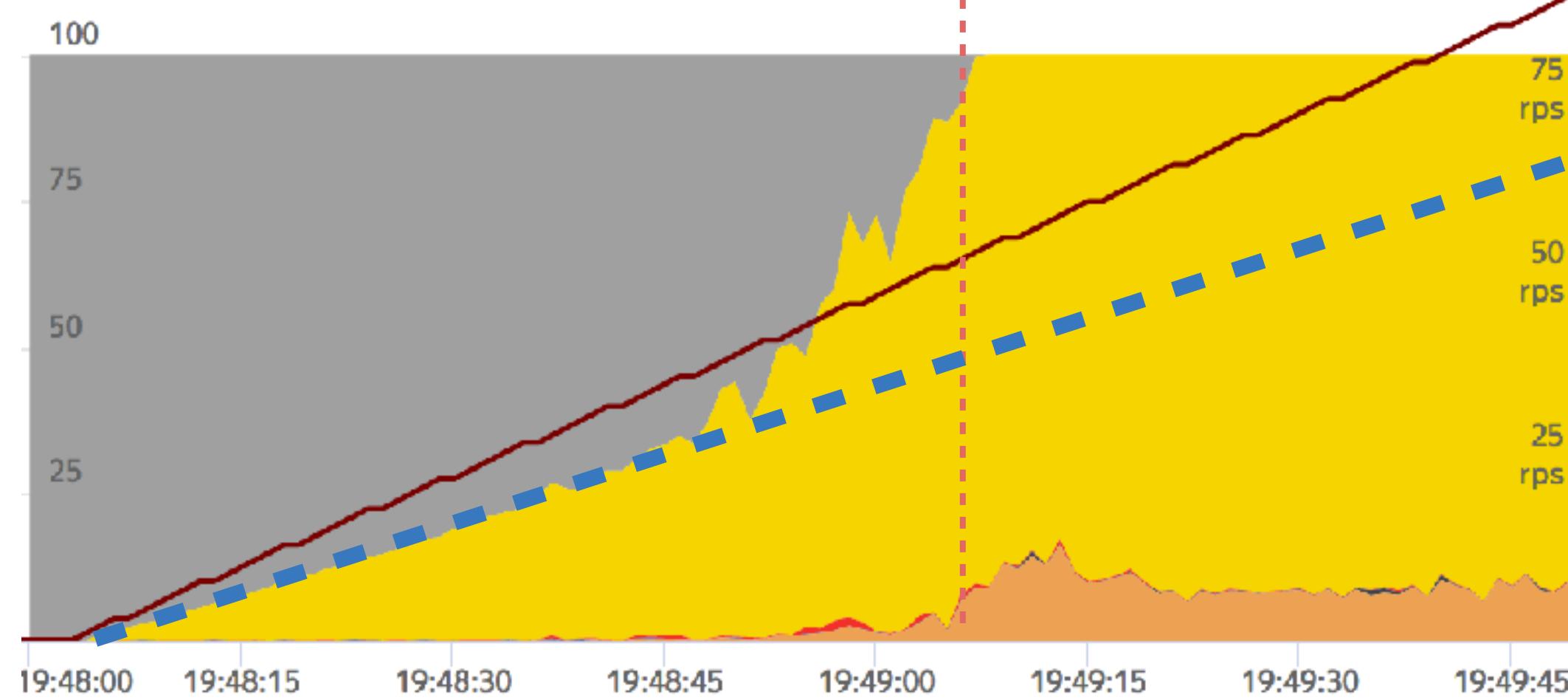


overload.yandex.net/14250

RT quantiles



CPU





потенциальное место
для оптимизации (ДЗ)

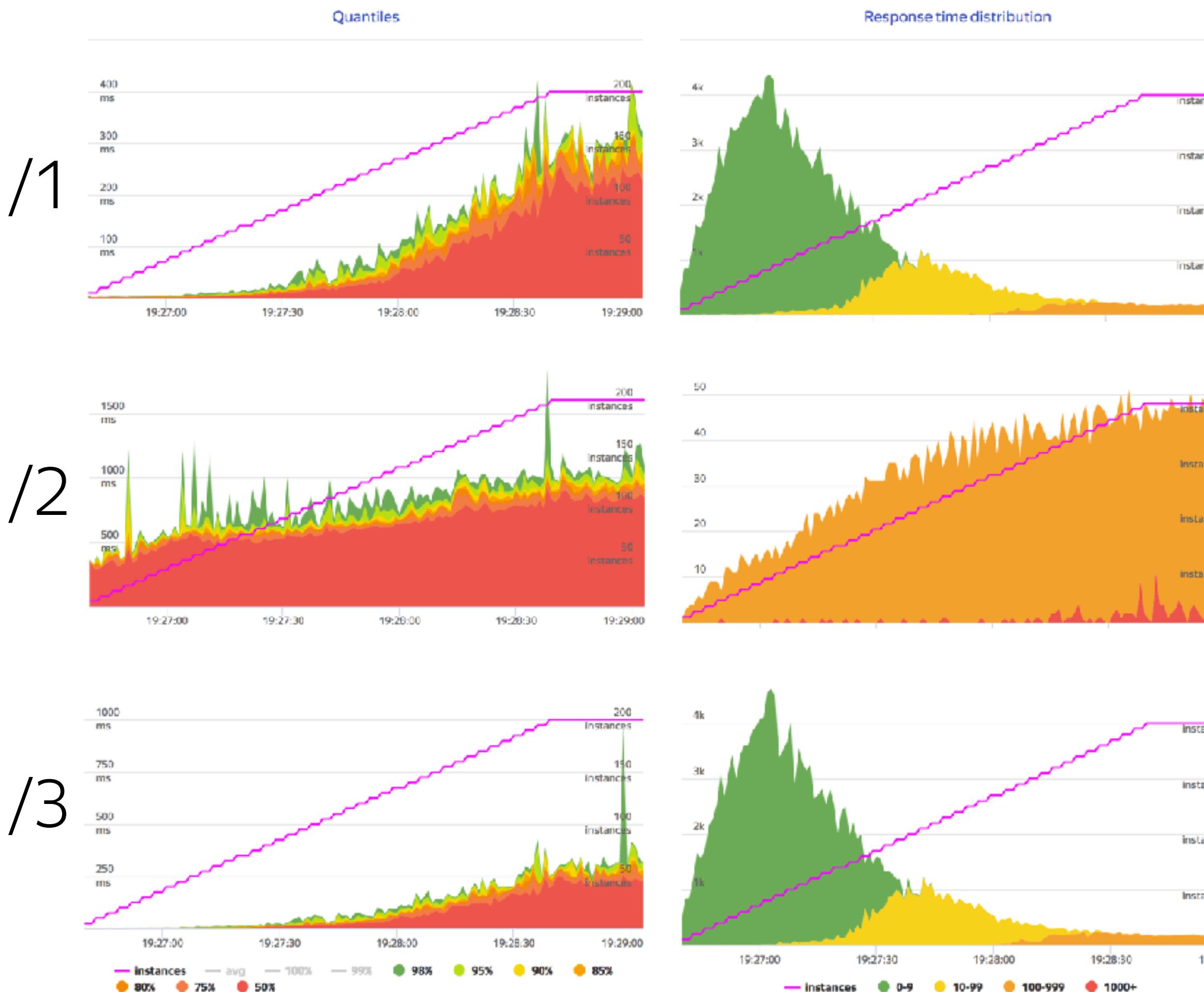
Узкое место: CPU

Код считает фракталы

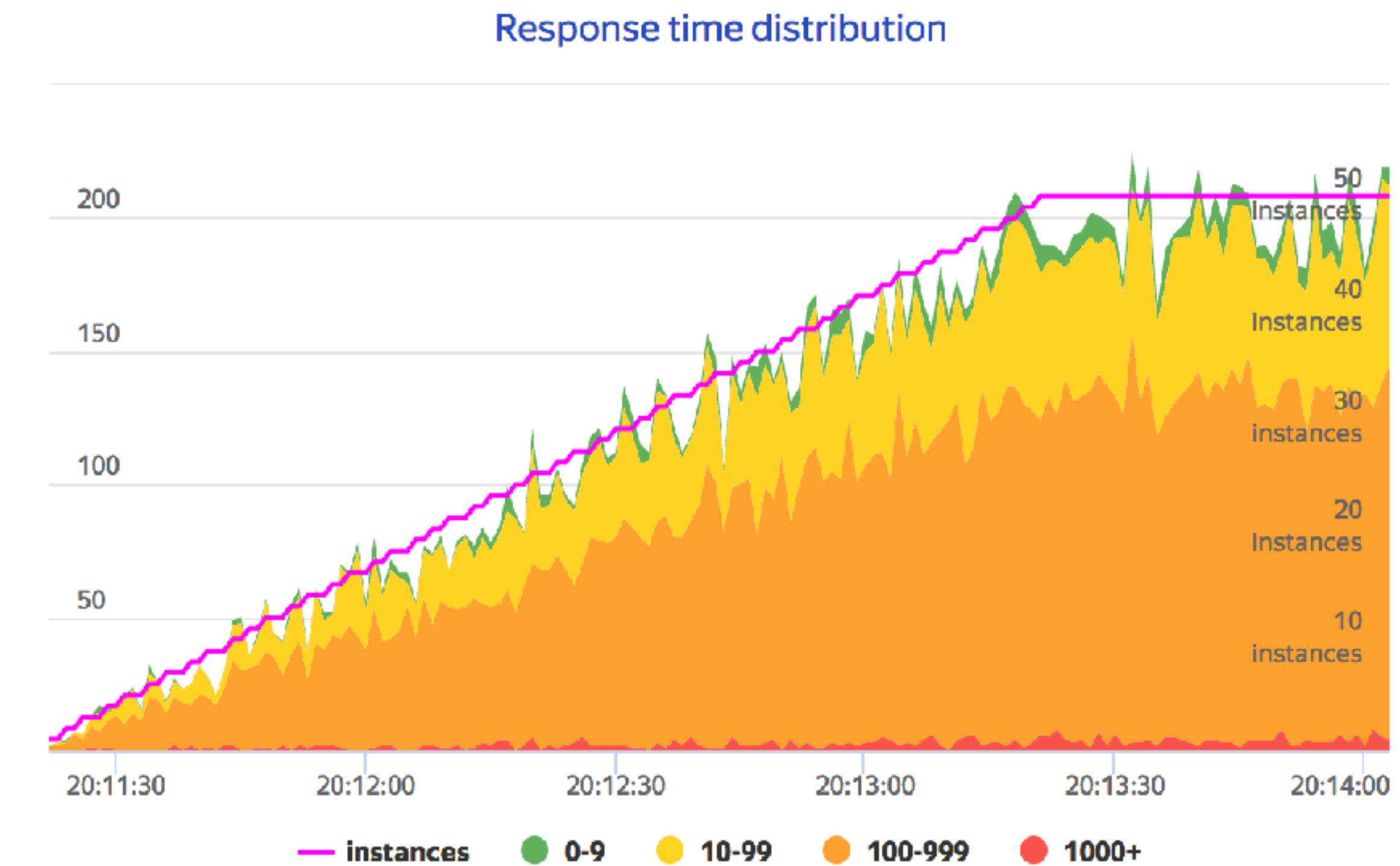
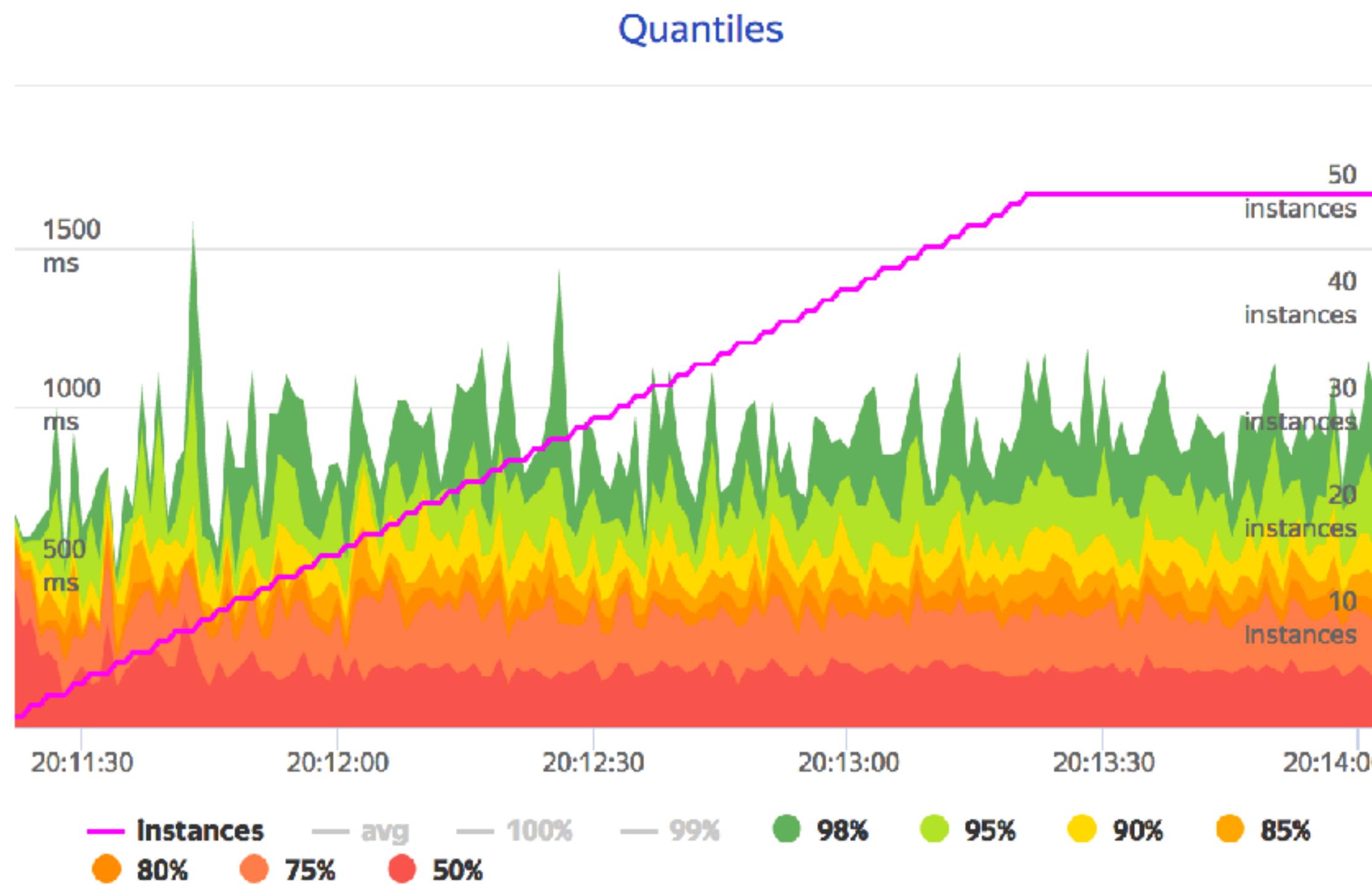
```
class CpuHandler(tornado.web.RequestHandler):  
    def get(self):  
        x, y = np.ogrid[-2:1:100j, -1.5:1.5:100j]  
        c = x + 1j*y  
        z = reduce(lambda x, y: x**2 + c, [1] * 3, c)  
        self.write(",".join(str(num) for num in np.angle(z)))
```

Ручка #5





overload.yandex.net/14254





50 инстансов, а ядер
всего 32, но все ок.

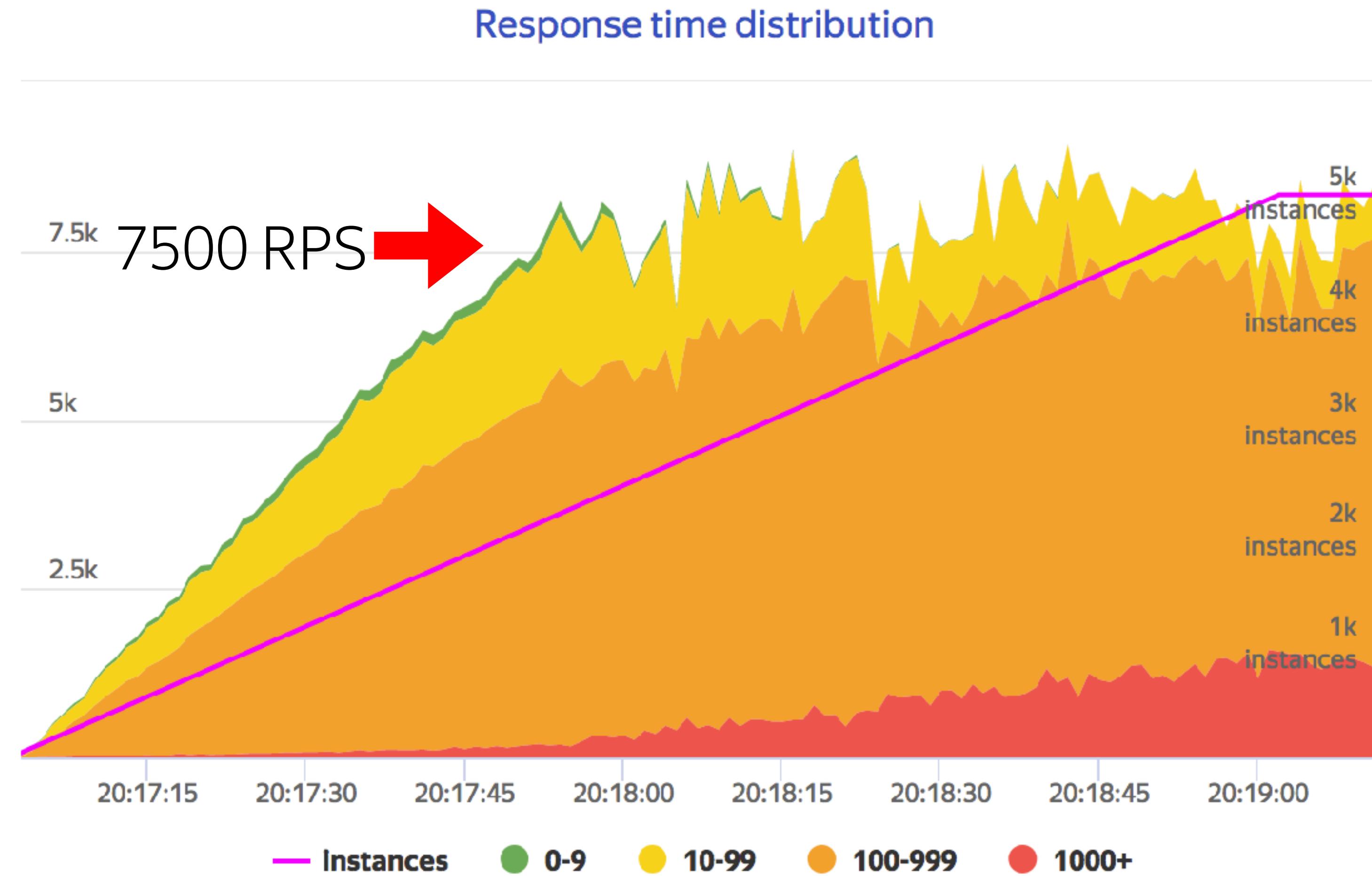
Почему?

Узкое место: внешний сервис

На самом деле в коде просто sleep =) Но поведение с внешним сервисом аналогично

```
class SleepHandler(tornado.web.RequestHandler):  
    @tornado.gen.coroutine  
    def get(self):  
        yield tornado.gen.sleep(np.random.lognormal(mean=-1.63, sigma=0.7))
```

overload.yandex.net/14255



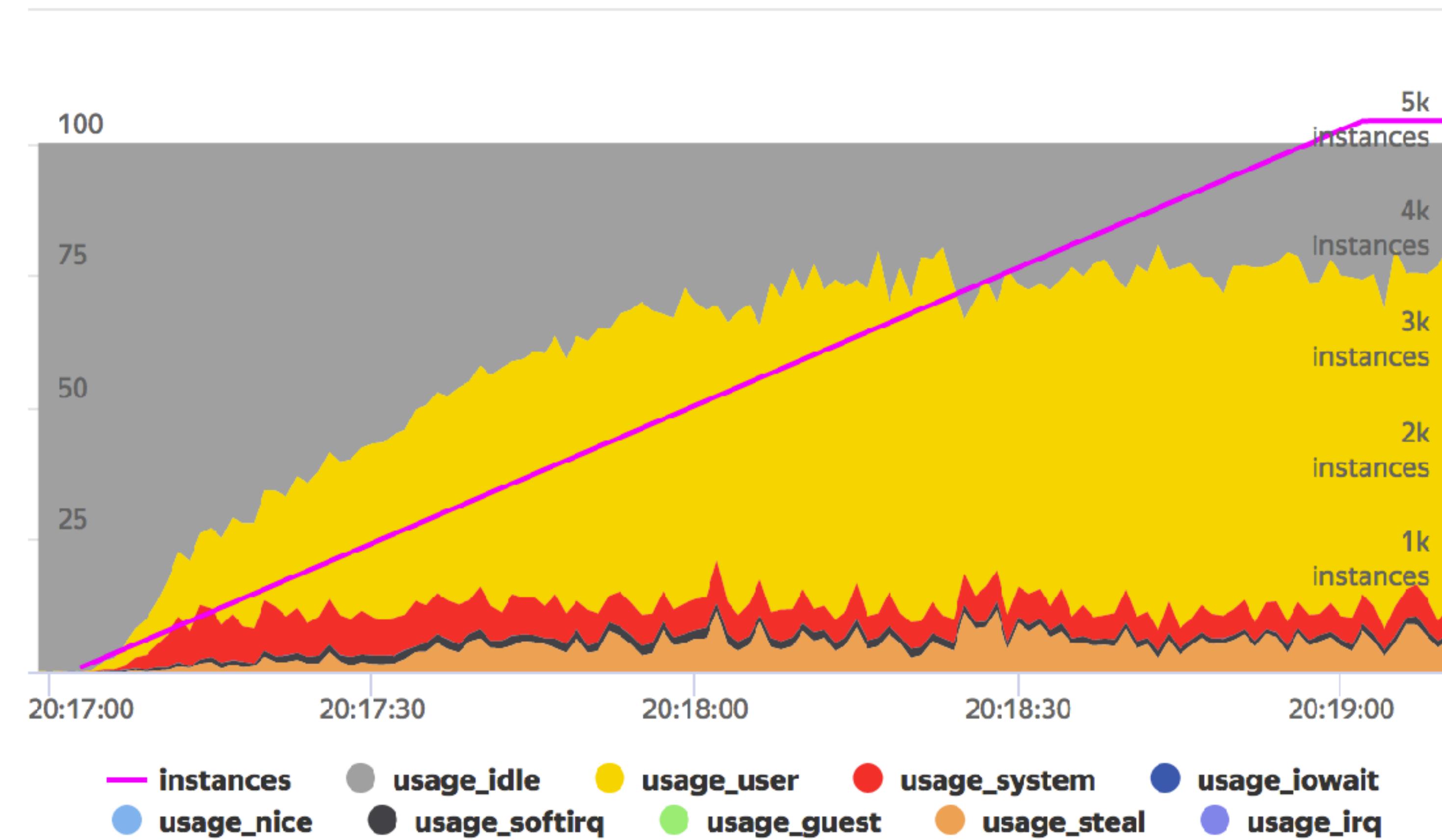


7500 RPS!

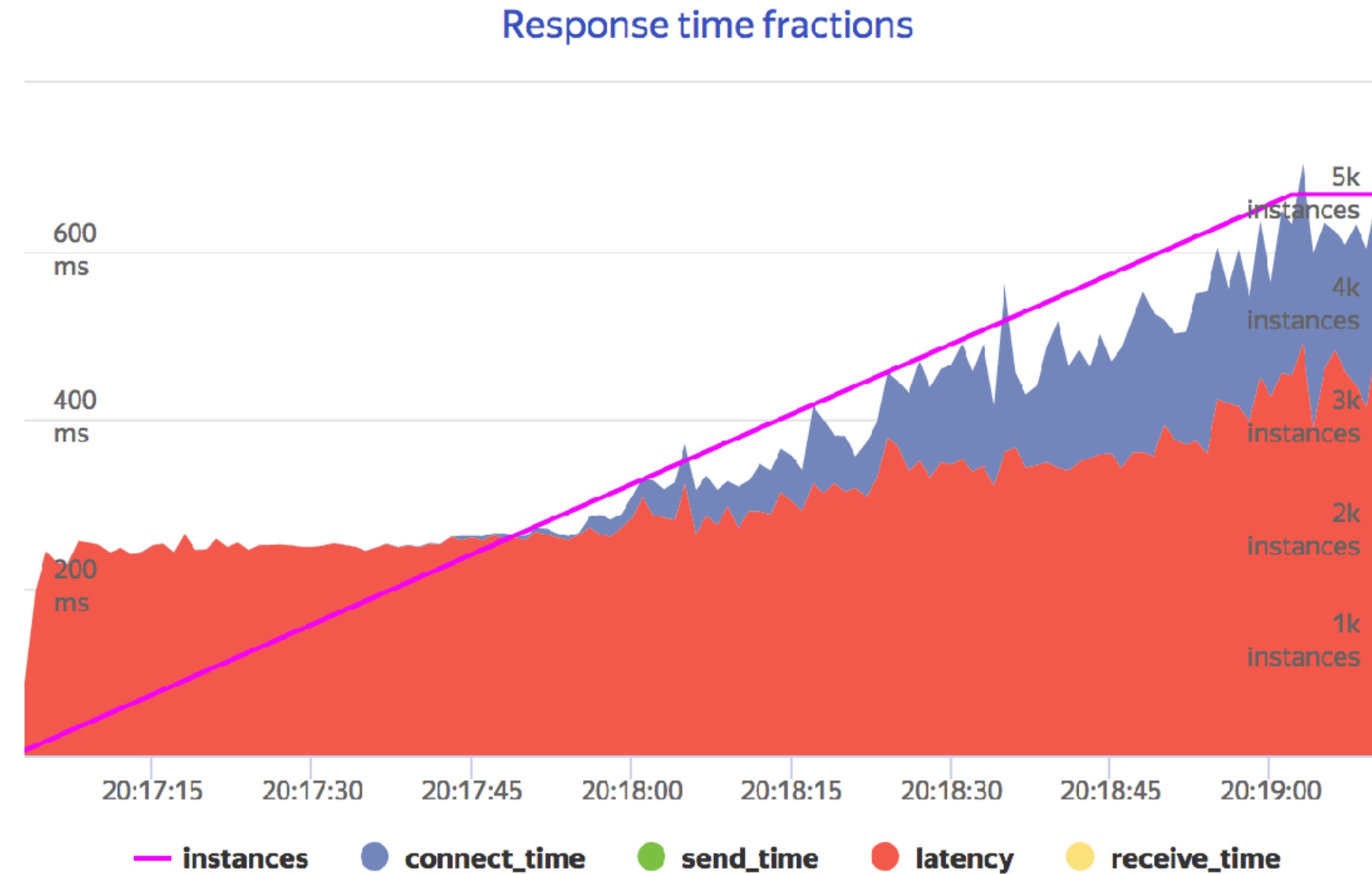
в общем teste было
всего 20! (дз)

overload.yandex.net/14255

cpu-cpu-total @ mysterious host 2311



overload.yandex.net/14255





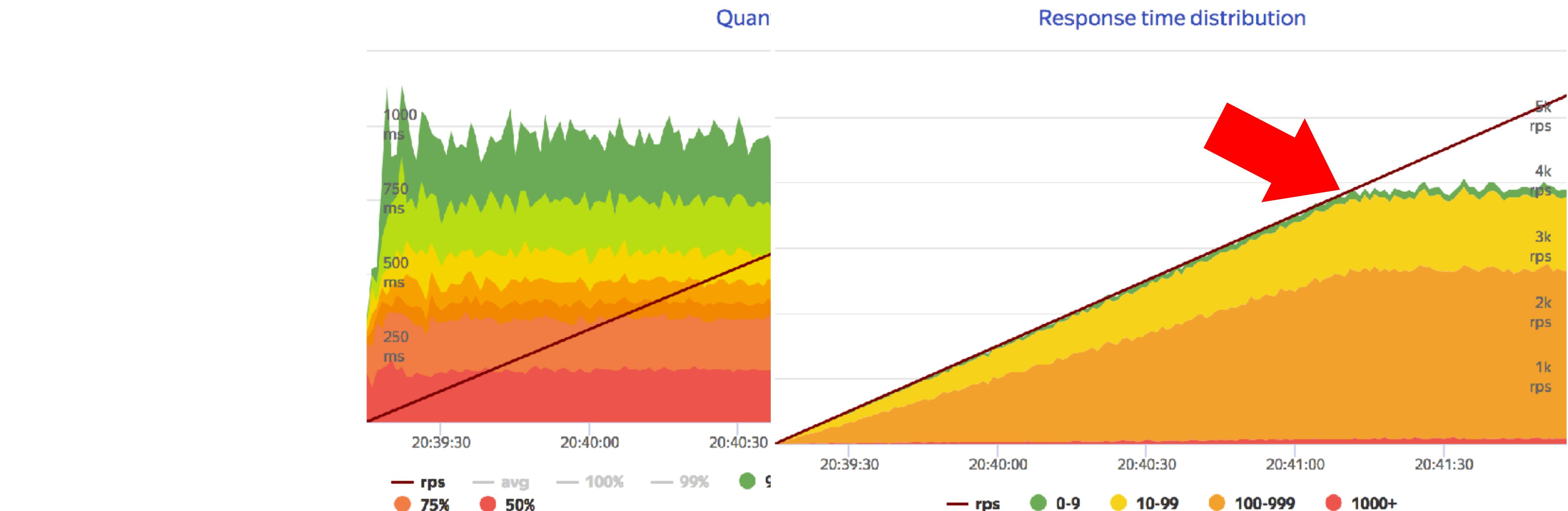
упираемся не в
процессор, а в число
соединений

подробности нужно исследовать дополнительно

Ручка #5
открытая модель



overload.yandex.net/14256

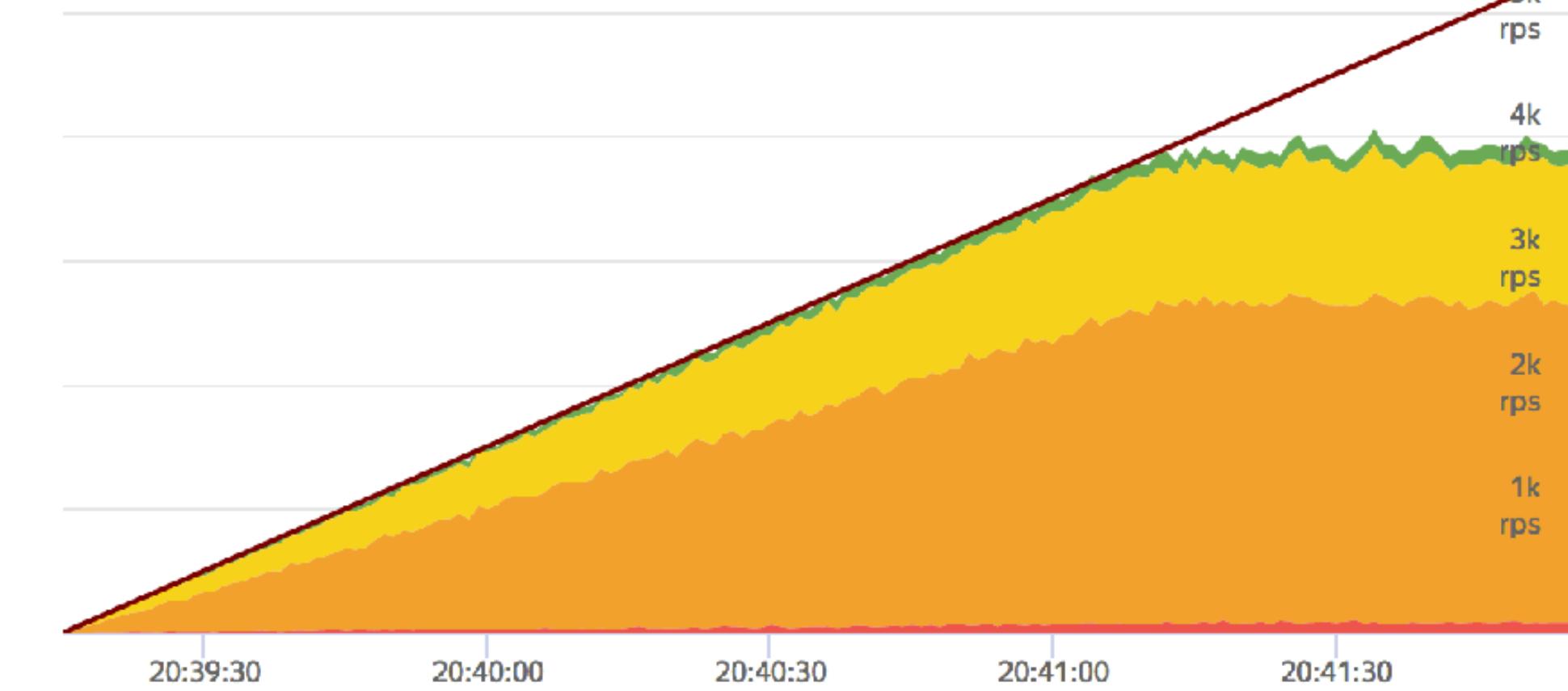
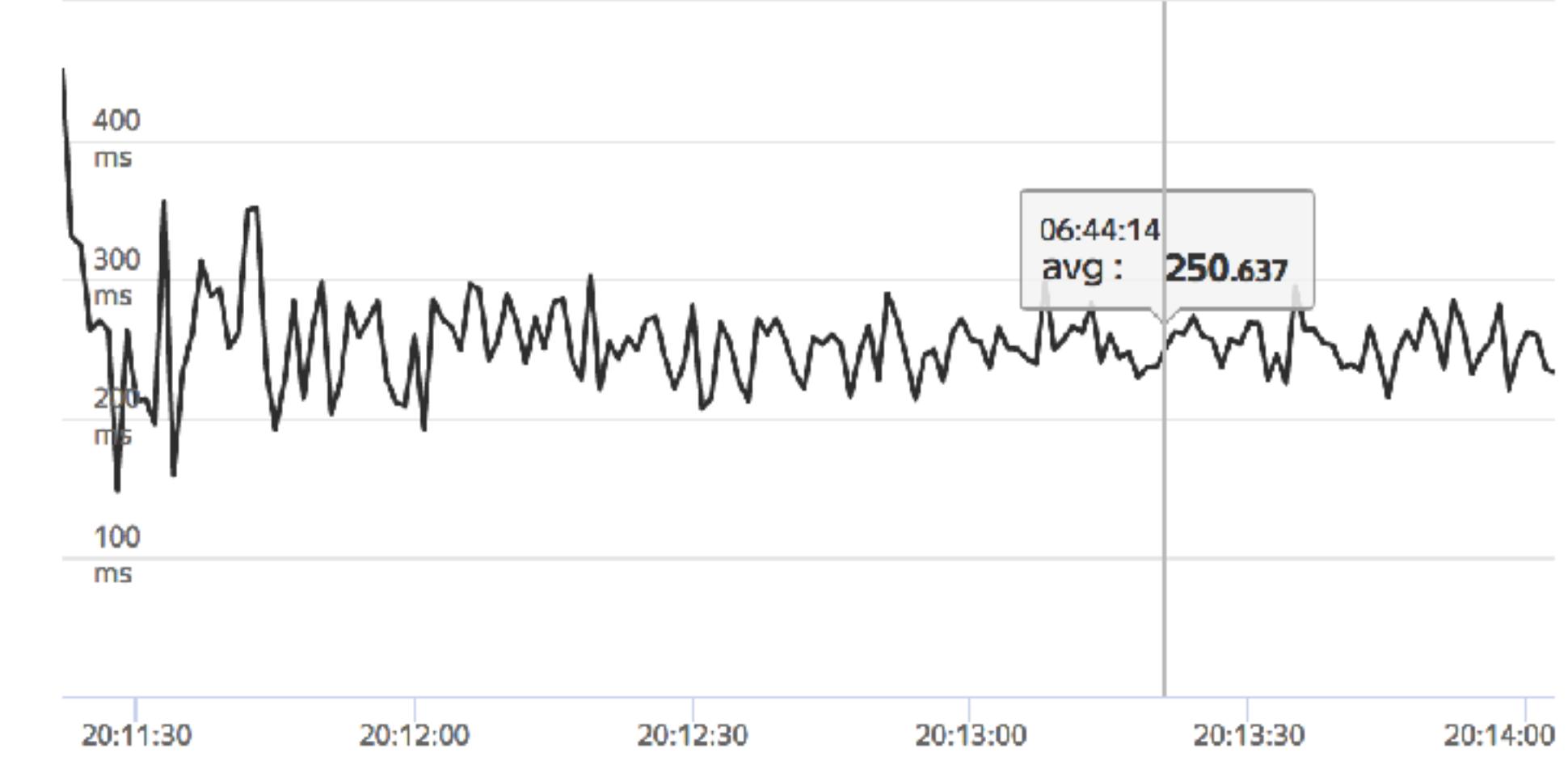


overload.yandex.net/14256

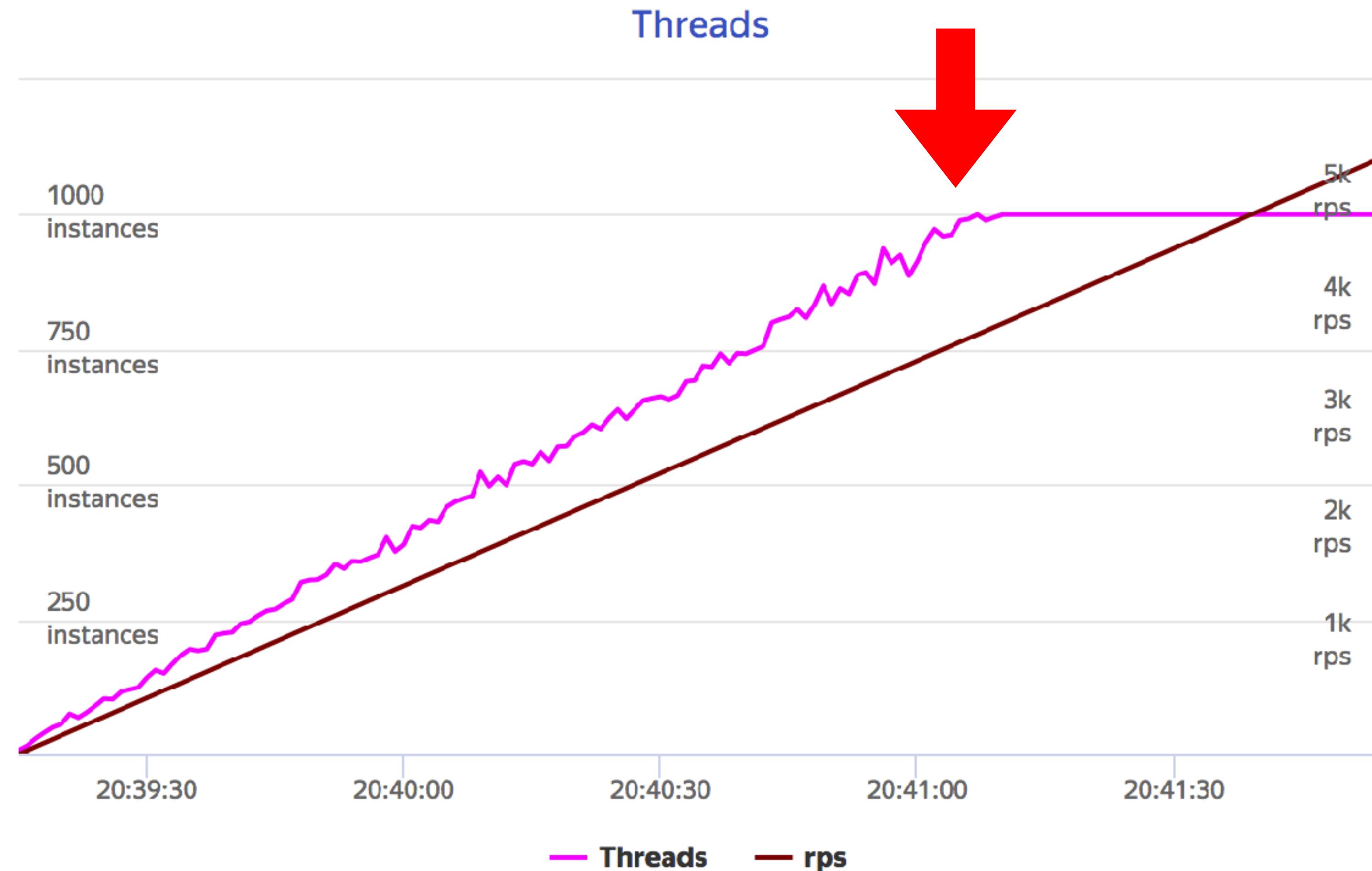
Вспоминаем формулу Литтла

| RPS = $1000 / T * \text{workers}$

$$1000 / 250 \text{ ms} * 1000 = 4000 \text{ RPS}$$



overload.yandex.net/14256





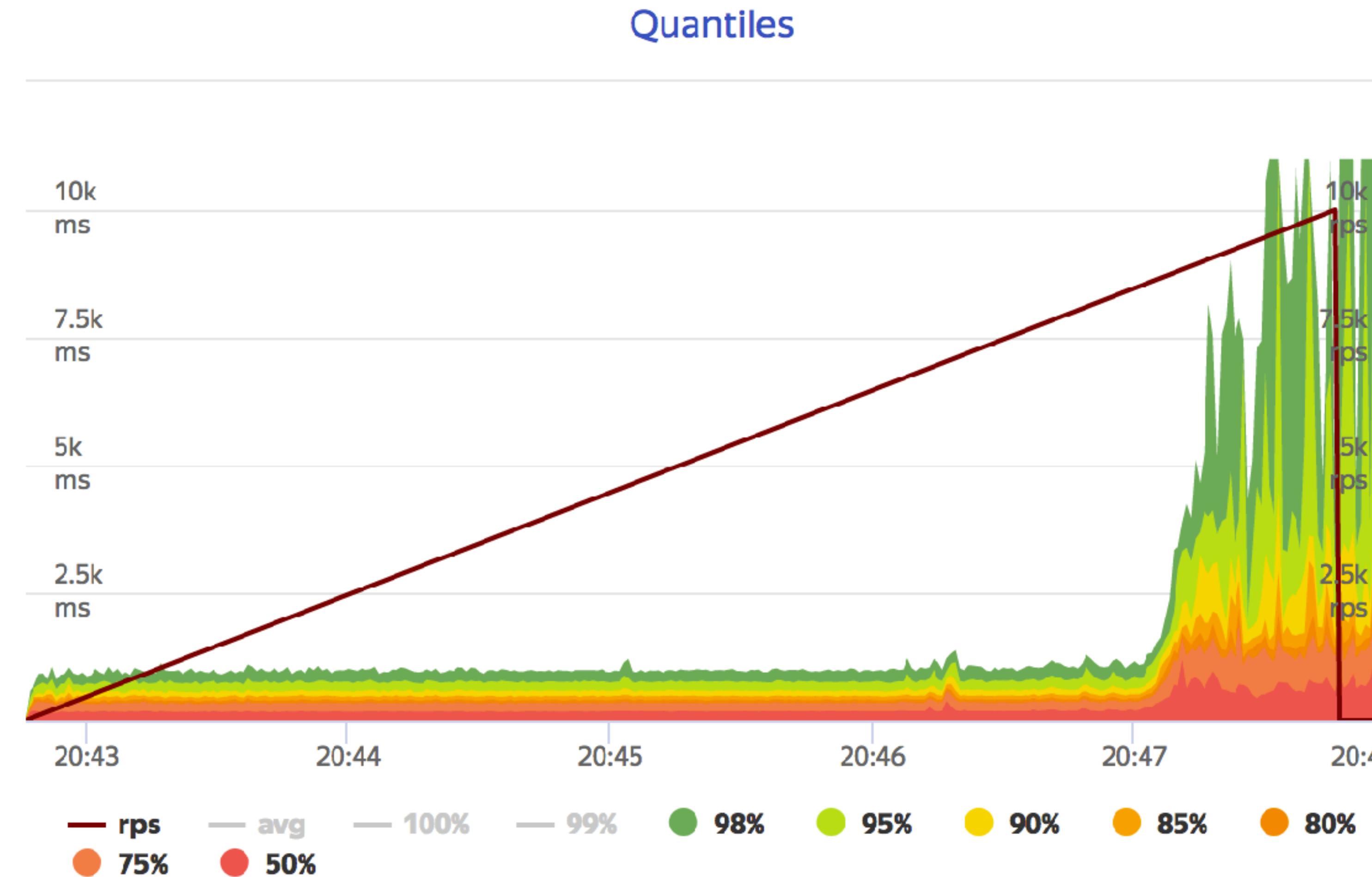
упираемся в число
инстансов генератора

~~Чиним~~ настраиваем генератор

На одной машинке можно запустить несколько десятков тысяч инстансов phantom. Зависит от объема памяти

```
[phantom]  
instances=10000
```

overload.yandex.net/14257



Ручка #5

времена ответов





времена ответов – это
то, чего от вас хочет
менеджер

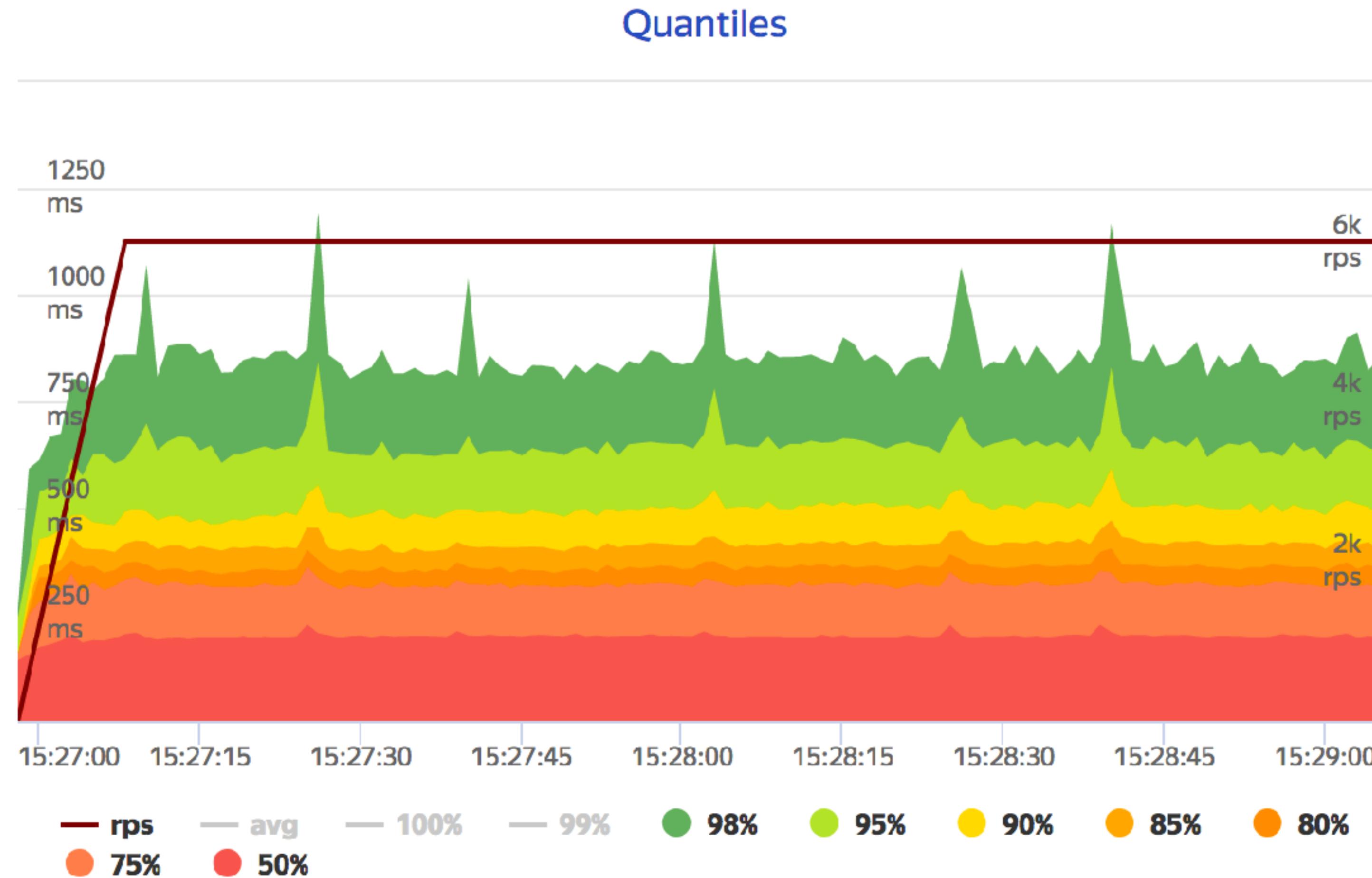
Стреляем постоянной нагрузкой

- › обязательно жестким расписанием
- › в начале прогрев (плавный разгон)
- › убедиться, что инстансов достаточно. Не должны упираться в инстансы ни разу за все время теста (пики)

[phantom]

```
rps_schedule=line(1,6000,30s) const(6000,3m)  
instances=10000
```

overload.yandex.net/16363



Cumulative quantiles

	Quantile
› квантили в отчетах считаются по корзинкам, потому что квантили неаддитивные, а логи не всегда влезают в память	99% 1045
	98% 860
	95% 650
	90% 497
› корзинки маленькие, но не бесконечно маленькие	85% 414
	80% 361
	75% 321
	50% 200

Анализ лога phantom

Можно анализировать сырье данные, чтобы найти точные квантили. Или сделать что-то еще.

Лог phantom – это набор чисел в формате tab separated

для анализа удобно использовать Python, Pandas и Jupyter

1493056685.687	_5#0	11405	36820	11363	26	11390	19	1940	200
1493056685.783	_5#3	29519	31317	29153	36	29346	19	1940	200
1493056685.799	_5#4	15803	2129	15541	41	15709	19	1940	200
1493056685.813	_5#5	21387	2109	21145	23	21288	19	1940	200
1493056685.846	_5#8	11890	1629	11698	21	11823	19	1940	200

Гистограмма из сырых данных

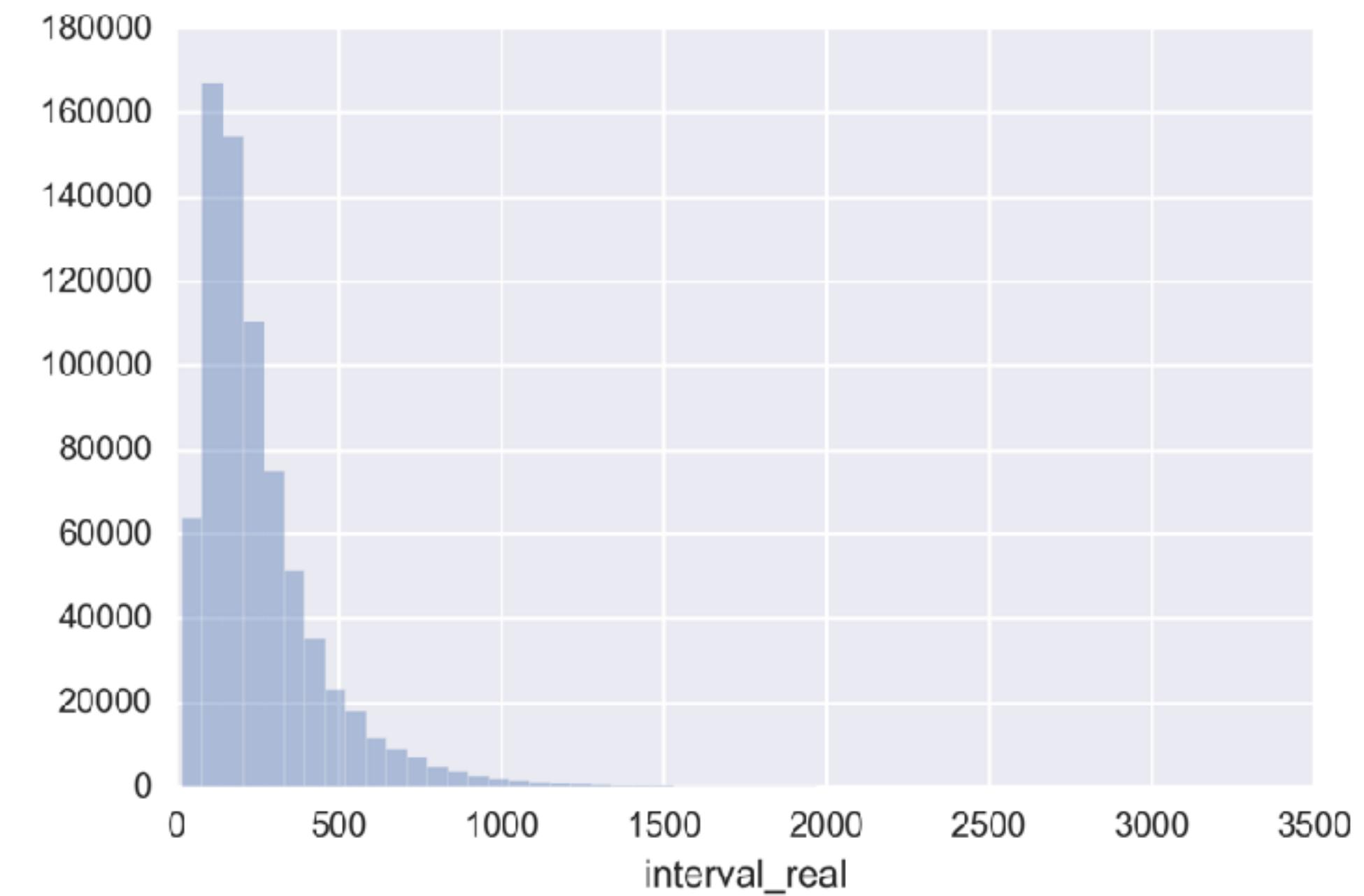
```
import pandas as pd
import seaborn as sns

phout_columns = [
    'time', 'tag', 'interval_real', 'connect_time', 'send_time',
    'latency', 'receive_time', 'interval_event', 'size_out',
    'size_in', 'net_code', 'proto_code']
ph_16363 = pd.read_csv("./16363_phout.log", sep='\t', names=phout_columns)

sns.distplot(ph_16363["interval_real"]/1000.0, kde=False)
```

Гистограмма времен ответа 16363

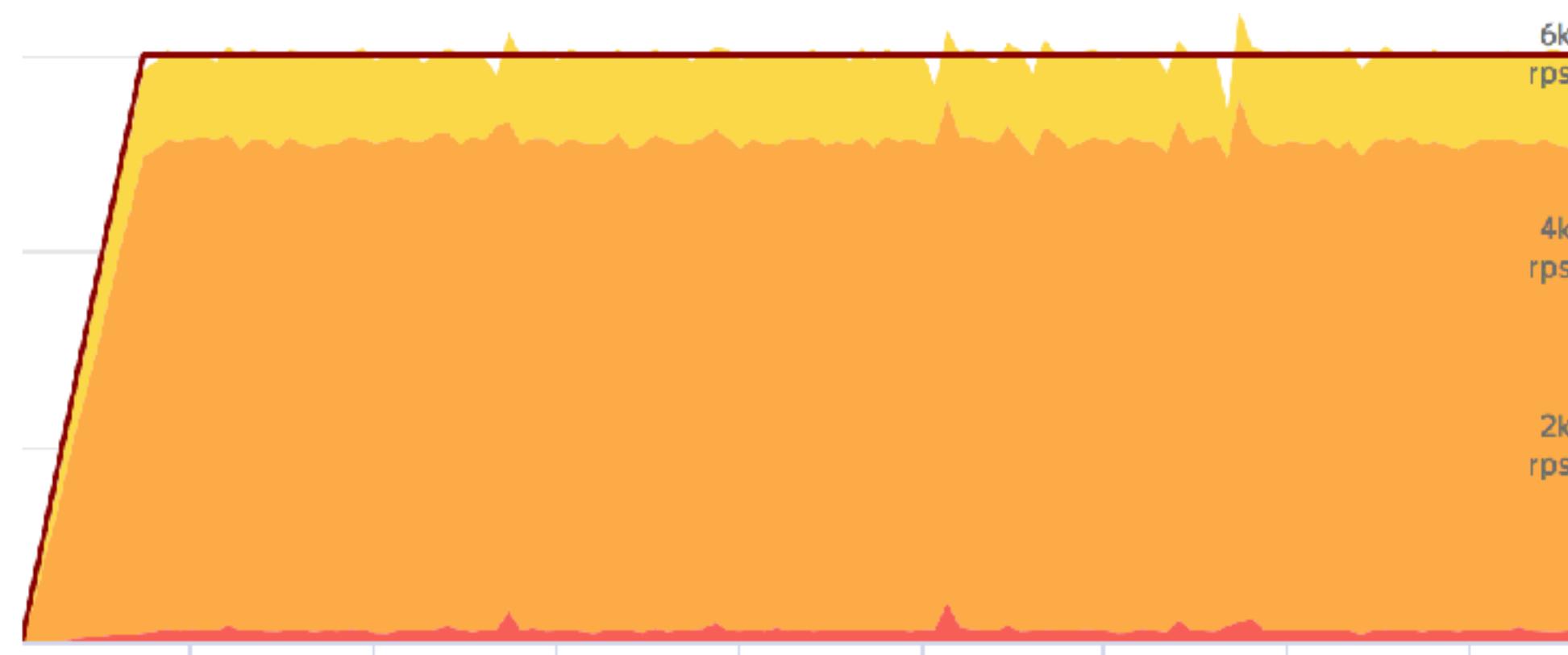
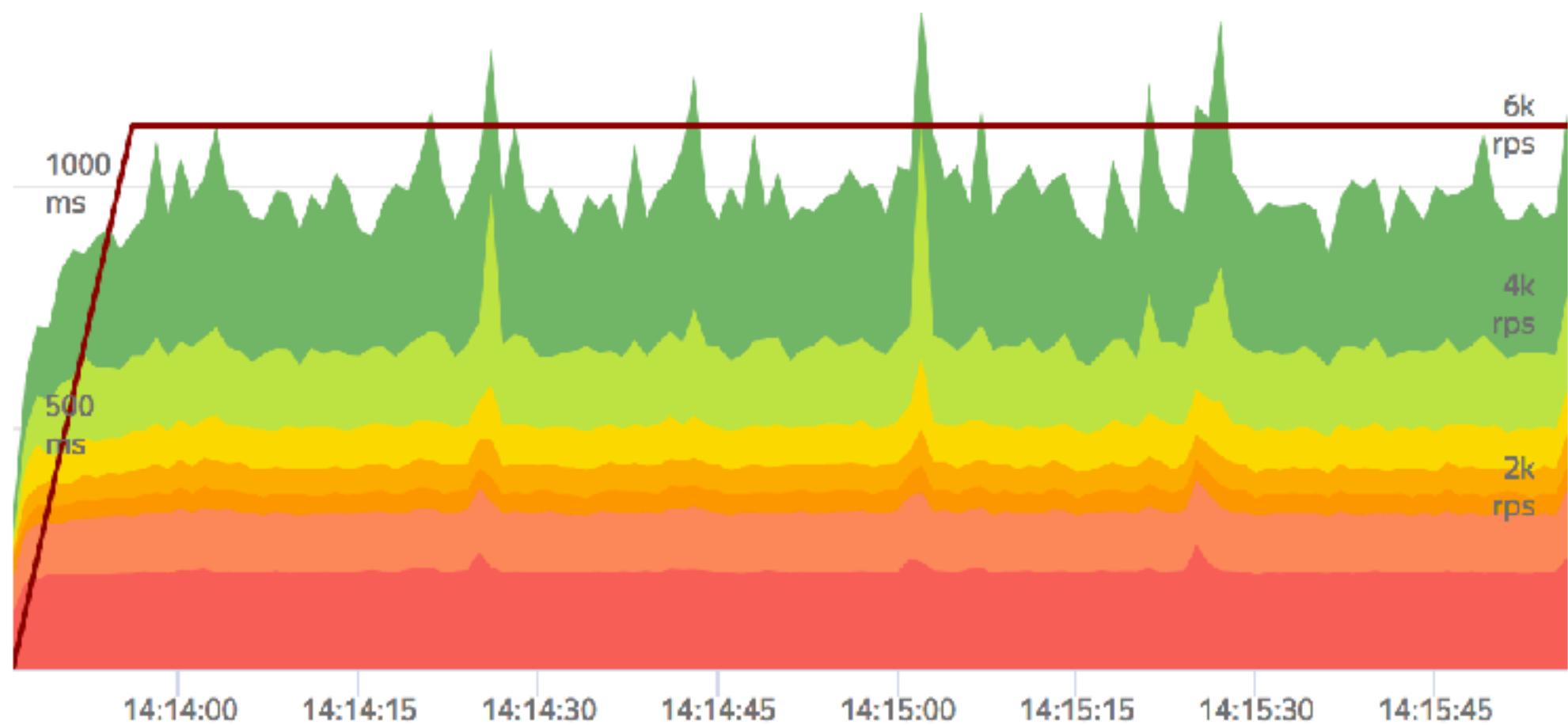
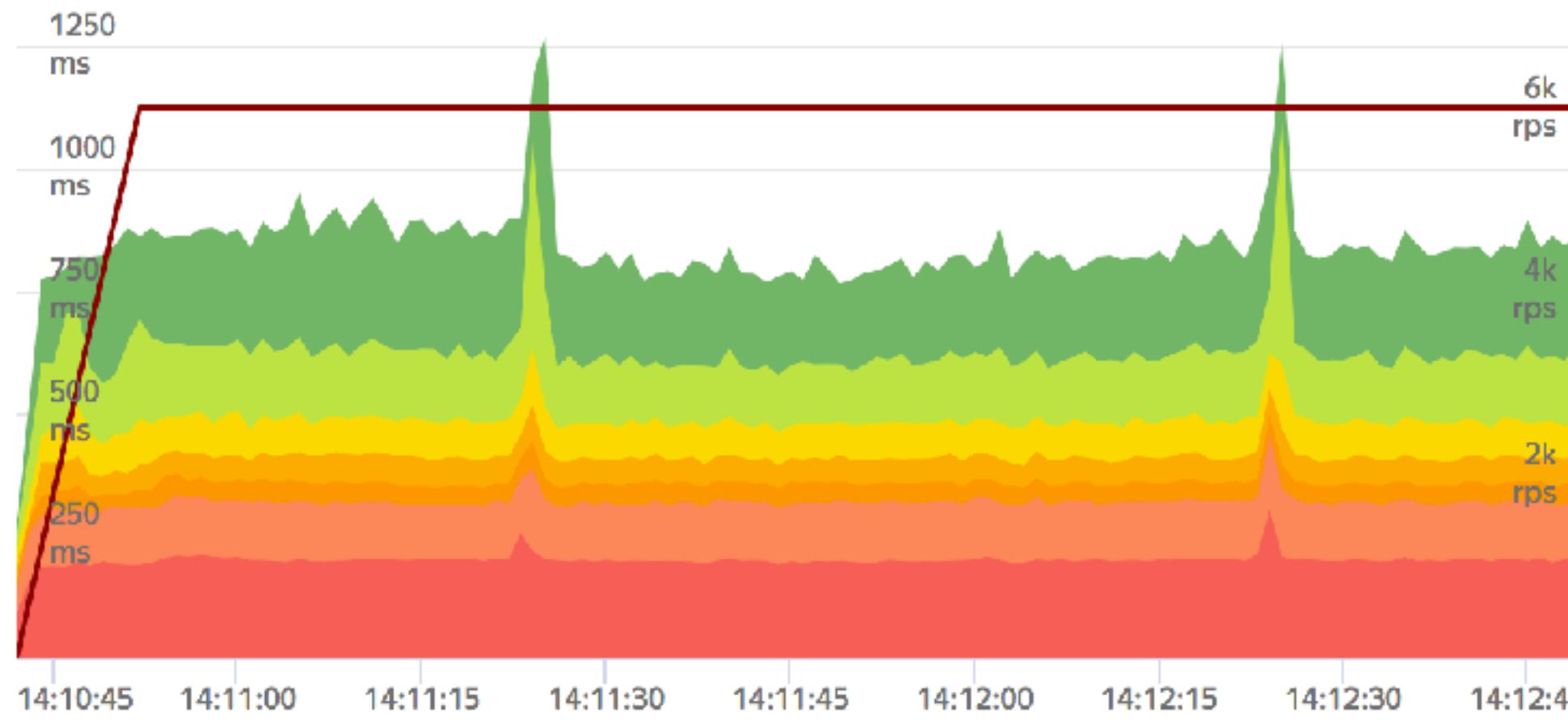
- › визуальный анализ распределения
- › можно увидеть артефакты, например, пики в хвосте



Сравнение времен ответов



17742 vs 17743



17742 vs 17743



лучше



лучше... наверное



17742 vs 17743

Кумулятивные квантили

	17742	17743
99%	1065.000 ms	1715.000 (+650.000)
98%	855.000 ms	1010.000 (+155.000)
95%	630.000 ms	675.000 (+45.000)
90%	487.000 ms	510.000 (+23.000)
85%	412.000 ms	424.000 (+12.000)
80%	359.000 ms	369.000 (+10.000)
75%	320.000 ms	327.000 (+7.000)
50%	200.000 ms	204.000 (+4.000)

Еще более спящий хэндлер

в одном проценте случаев сервис отвечает дольше

```
if np.random.rand() < 0.99:  
    yield tornado.gen.sleep(  
        np.random.lognormal(mean=-1.63, sigma=0.7))  
  
else:  
    yield tornado.gen.sleep(  
        np.random.lognormal(mean=1, sigma=0.25))
```

Пробуйте это дома



Продолжайте стрелять!

- › не смотрите в код =) `pip install target`
- › попытайтесь понять, что делают остальные ручки
- › почему нелинейно растет потребление CPU в ручке №2?
- › почему ручка №5 так бессовестно разваливается при стрельбе по всем ручкам?
- › закомитьте свои собственные ручки (github.com/yandex-load/target)



вдумчиво смотрите
на результаты тестов

bravenewgeek.com/everything-you-know-about-latency-is-wrong

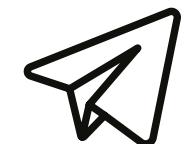
Спасибо за внимание!

Алексей Лавренюк

старший инженер по
тестированию



direvius@yandex-team.ru



[@direvius](https://t.me/direvius)