Проведем нагрузочное тестирование нашего потокобезопасного, асинхронного сервера с помощью *wrk* в несколько соединений после реализации горизонтального масштабирования. Проанализируем результаты профилирования под нагрузкой с помощью *async-profiler*. Профилирование будем проводить трех типов:

- *CPU profiling*;
- *Allocation profiling*;
- *Lock profiling*.

1 Put

Проведем нагрузку на сервер запросами на вставку данных, обращаясь к каждому узлу одновременно:

```
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/put.lua -L \
> http://localhost:8080
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/put.lua -L \
> http://localhost:8081
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/put.lua -L \
> http://localhost:8082
```

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 1) , *alloc* (рис. 3) и *lock* (рис. 6).

На рис. 1 видно, что время работы каждого воркера было примерно одинаково, что может говорить о сбалансированной нагрузке на каждый узел. Сама вставка в каждом воркере (рис. 2) заняла не более 1%, и он теперь тратит $\approx 3\%$ времени на проксирование запроса.

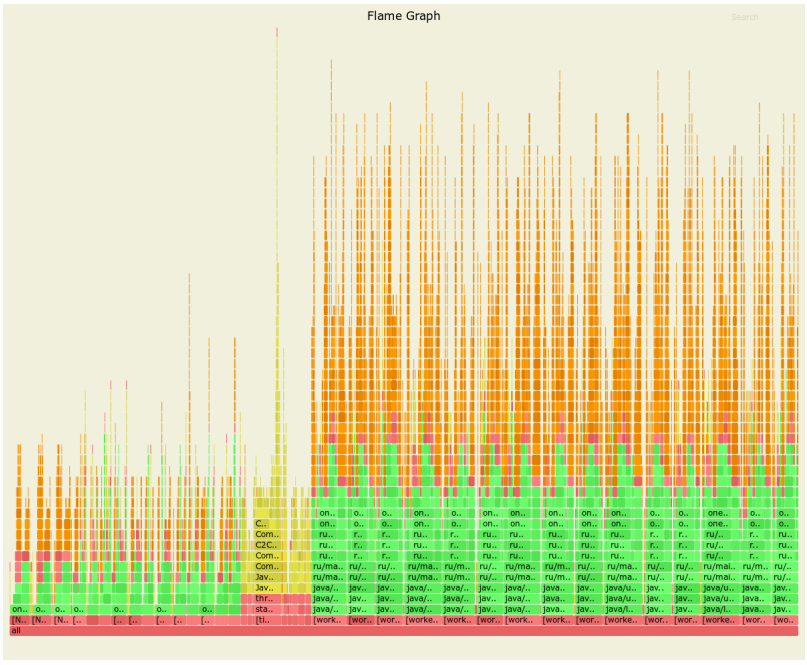

Рис. 1

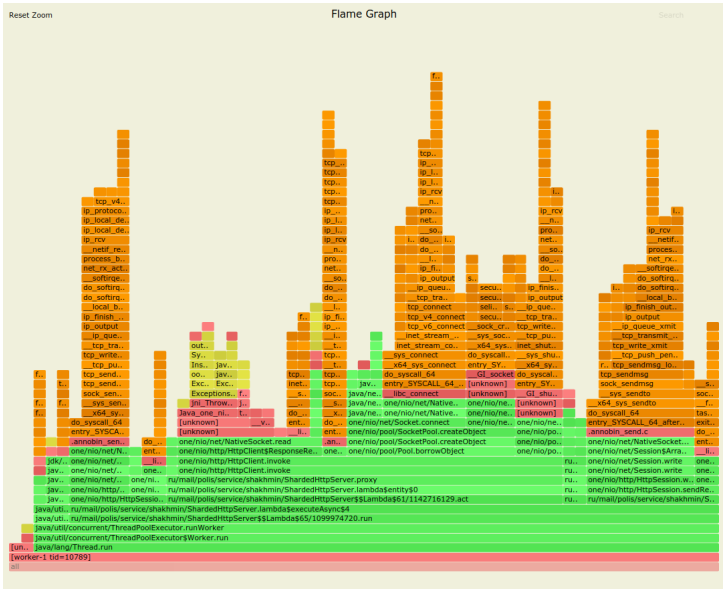

Рис. 2

На рис. 4 можно заметить, что выделение памяти в селекторах в основном происходило при обработке полученного запроса, где менее 1% памяти теперь выделяется на определение принадлежности ключа узлу. На воркере (рис. 5) аллокация большей части памяти происходила при проксировании запроса, а на запись данных в хранилище отводится менее 1%.

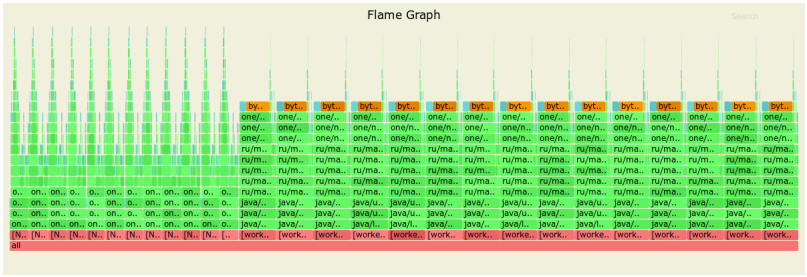

Рис. 3

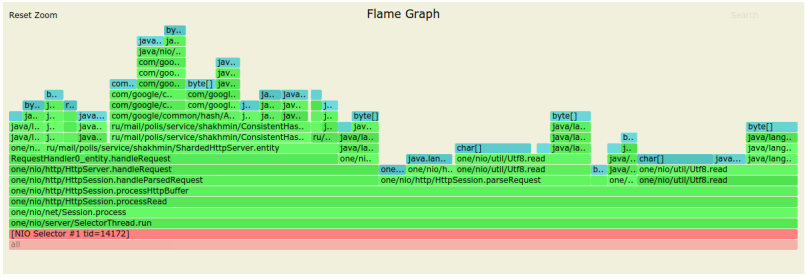

Рис. 4

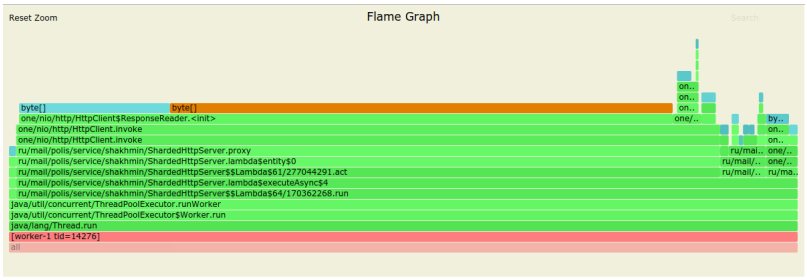

Рис. 5

На рис. 6 и рис. 7 видно, что воркеры блокировались, чтобы взять задачу на обработку ($\approx$ 1%), и, чтобы записать данные в хранилище ($\approx$ 2%). Селекторы блокировались только, чтобы отдать задачу на выполенение в пул потоков.

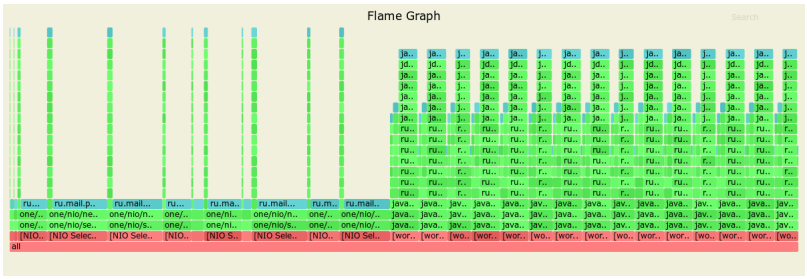

Рис. 6

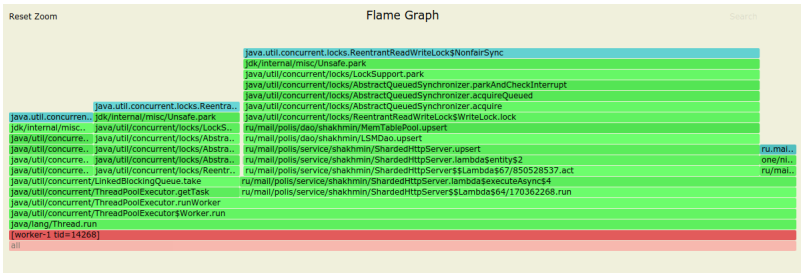

Рис. 7

2 Get

Проведем нагрузку на сервер запросами на получение данных:

```
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/get.lua -L \
> http://localhost:8080
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/get.lua -L \
> http://localhost:8081
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/get.lua -L \
> http://localhost:8082
```

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 8) , *alloc* (рис. 10) и *lock* (рис. 13).

На рис. 9 видно, что как и раньше при обработке запроса на чтение в каждом воркере большую часть времени заняло получение итератора у *LSMDao* ($\approx 3.5\%$), а проксирование заняло $\approx 1\%$ времени.

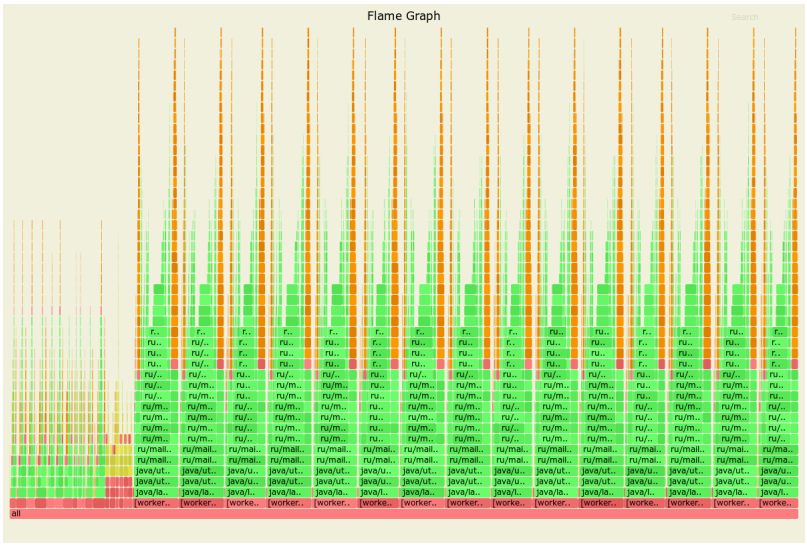

Рис. 8

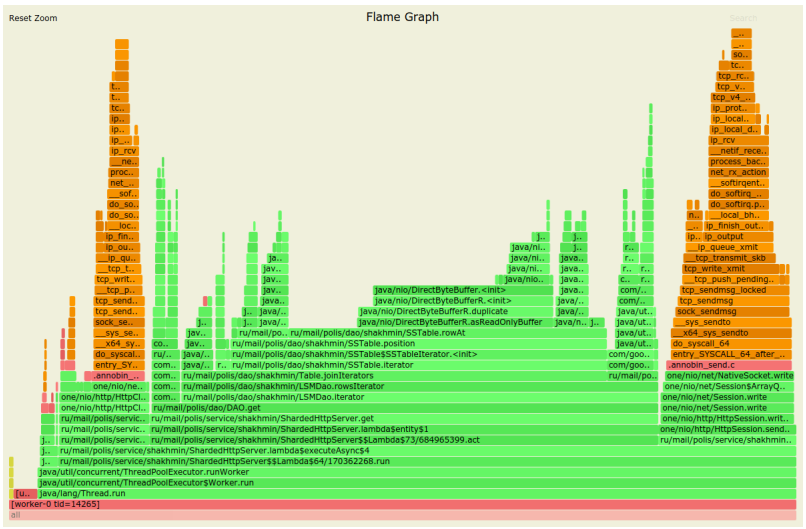

Рис. 9

На рис. 11 можно заметить, что выделение памяти в селекторах в основном происходило при обработке полученного запроса, где около 1% памяти теперь выделяется на определение принадлежности ключа узлу. На воркере (рис. 12) большая часть памяти была выделена объектам класса *DirectByteBuffer*, которые создавались при вызове метода *SSTable.iterator*

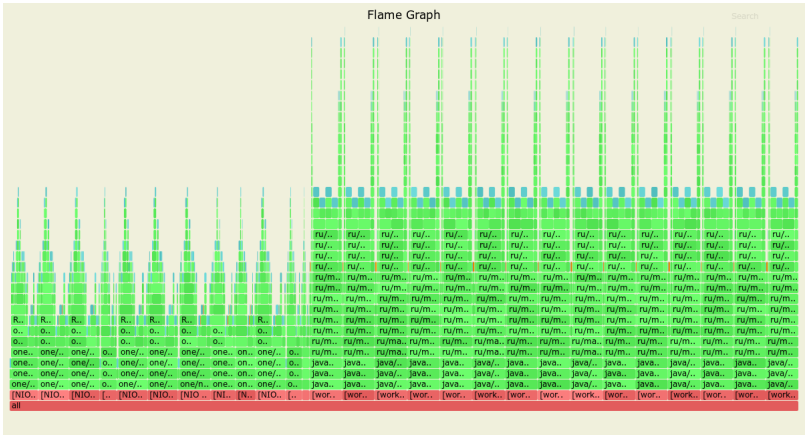

Рис. 10

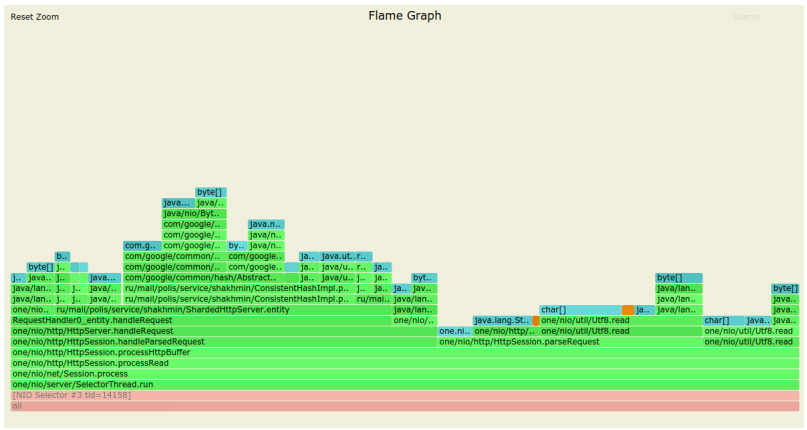

Рис. 11

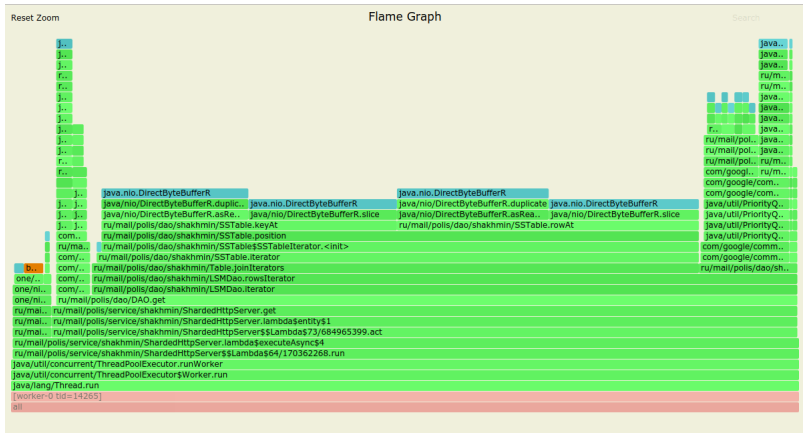

Рис. 12

На рис. 13 и рис. 14 видно, что поскольку мы использовали *ReadWriteLock*, то воркеры блокировались только, чтобы взять задачу на обработку ($\approx$ 0.2%), а селекторы блокировались, чтобы отдать задачу на выполенение в пул потоков.

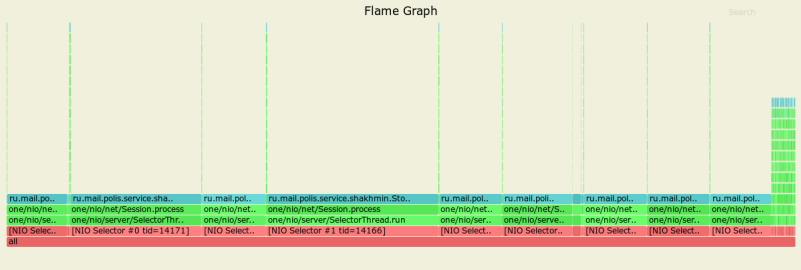

Рис. 13

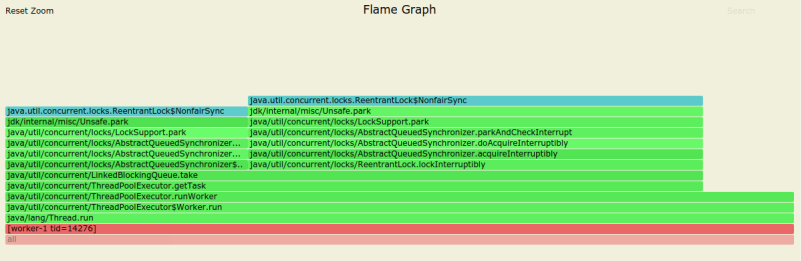

Рис. 14

3 Delete

Проведем нагрузку на сервер запросами на удаление данных:

```
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/delete.lua -L \
> http://localhost:8080
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/delete.lua -L \
> http://localhost:8081
$ wrk -t4 -c4 -d1m -R10000 -s ./wrk/delete.lua -L \
> http://localhost:8082
```

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 15) , *alloc* (рис. 17) и *lock* (рис. 20).

На рис. 16 видно, что как и раньше при обработке запроса на удаление в каждом воркере большую часть времени заняли сетевые вызовы, а само удаление было быстрым ($\approx 0.3\%$).

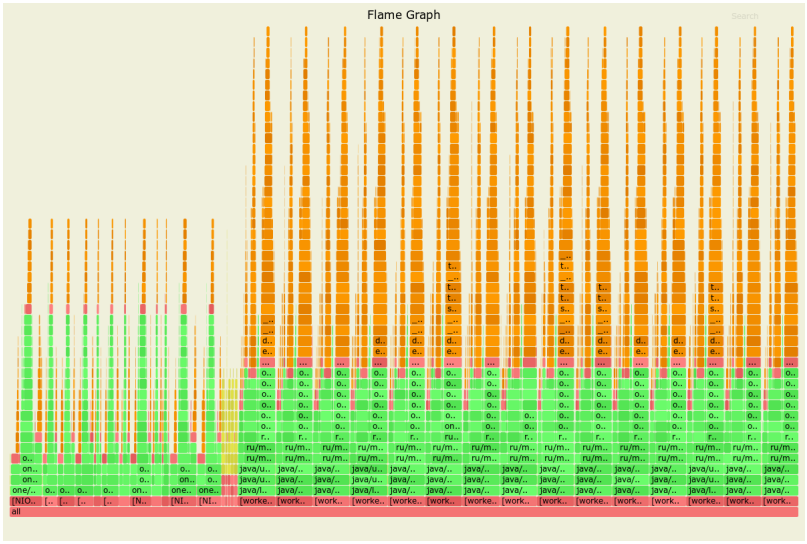

Рис. 15

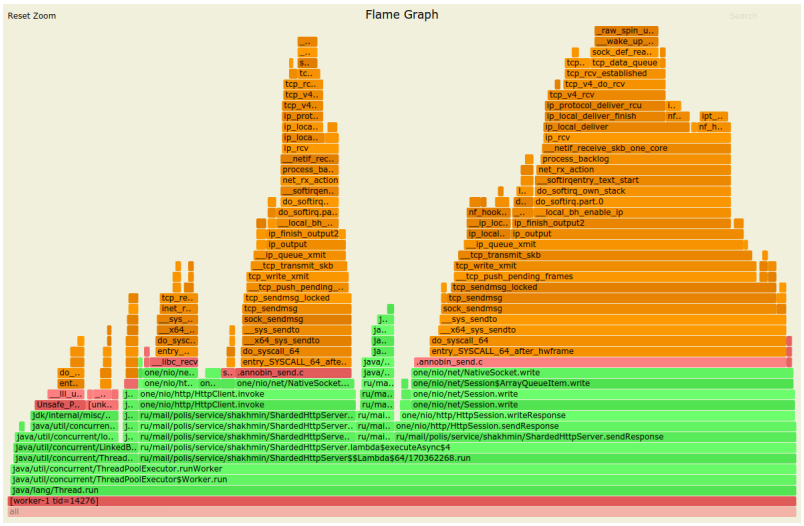

Рис. 16

На рис. 18 можно заметить, что выделение памяти в селекторах в основном происходило при обработке полученного запроса, где около 1% памяти теперь выделяется на определение принадлежности ключа узлу. На воркере (рис. 19) аллокация большей части памяти происходила при проксировании запроса, а на удаление данных в хранилище отводится менее 1%.

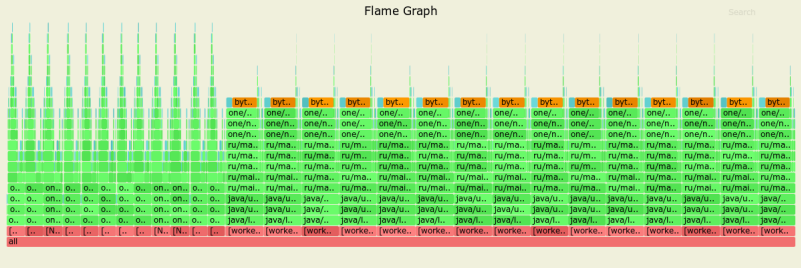

Рис. 17

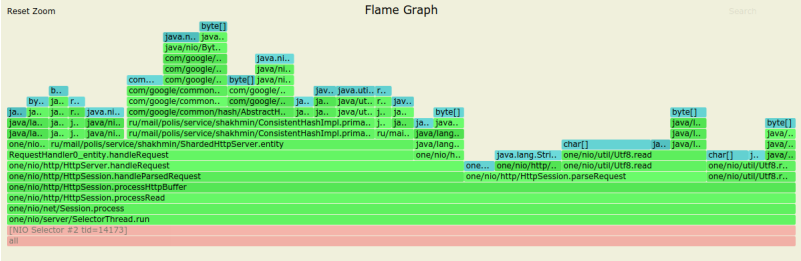

Рис. 18

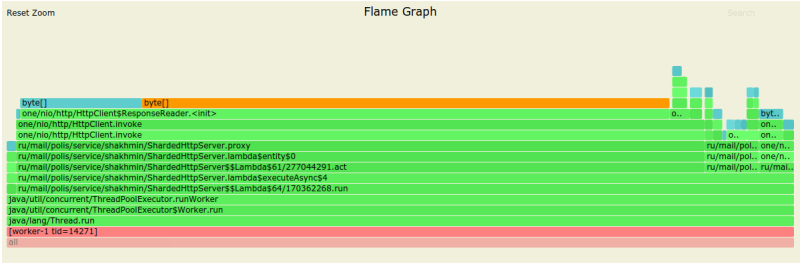

Рис. 19

На рис. 20 и рис. 21 видно, что воркеры блокировались, чтобы взять задачу на обработку ($\approx$ 2%), и, чтобы удалить данные из хранилища ($\approx$ 2%). Селекторы блокировались только, чтобы отдать задачу на выполенение в пул потоков.

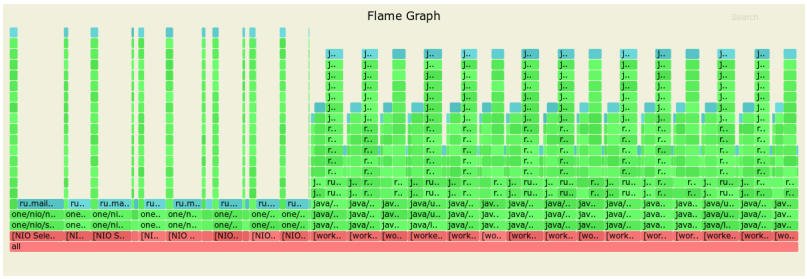

Рис. 20

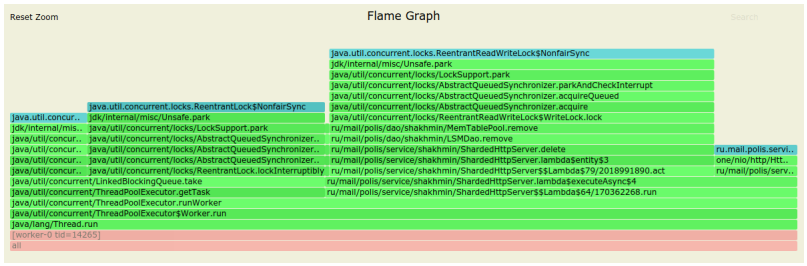

Рис. 21