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

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

Сравним результаты профилирования с блокирующей версией сервера.

1 Status and Put

Нагрузим наш сервер запросами на получение статуса:

```
  $ wrk -d20s -R10000 -L http://localhost:8080/v0/status

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.25ms  735.58us  13.80ms   75.62%
    Req/Sec     5.26k   425.81     9.78k    79.68%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.18ms
 75.000%    1.64ms
 90.000%    2.05ms
 99.000%    2.98ms
 99.900%    8.53ms
 99.990%   11.90ms
 99.999%   13.52ms
100.000%   13.81ms

#[Mean    =        1.253, StdDeviation   =        0.736]
#[Max     =       13.800, Total count    =        99743]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  199890 requests in 20.00s, 11.82MB read
Requests/sec:   9994.63
Transfer/sec:    605.14KB
```

Теперь рассмотрим результаты при одновременной нагрузке запросами на запись и на получение статуса:

```
  $ wrk -t4 -c1024 -d3m -R100000 -s ./wrk/put.lua --L http://localhost:8080
  $ wrk -d20s -R10000 -L http://localhost:8080/v0/status

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.20ms    0.92ms  16.34ms   90.42%
    Req/Sec     5.28k   600.44    11.22k    86.39%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.09ms
 75.000%    1.51ms
```

```
 90.000%    1.86ms
 99.000%    4.76ms
 99.900%   11.34ms
 99.990%   14.07ms
 99.999%   15.94ms
100.000%   16.34ms

#[Mean    =        1.201, StdDeviation   =        0.923]
#[Max     =       16.336, Total count    =        99745]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  199888 requests in 20.00s, 11.82MB read
Requests/sec:   9994.50
Transfer/sec:    605.14KB
```

По результатам видно, что пропускная способность практически не изменилась, и 99.9 перцентиль стала медленее ($\approx 3ms$), но незначительно. Сравним эти результы с результатами, которые были при блокирующей версии сервера:

```
  $ wrk -t4 -c1024 -d3m -R100000 -s ./wrk/put.lua --L http://localhost:8080
  $ wrk -d20s -R10000 -L http://localhost:8080/v0/status

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.21ms    1.20ms  28.86ms   98.01%
    Req/Sec     5.27k   642.10    14.40k    91.84%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.08ms
 75.000%    1.48ms
 90.000%    1.76ms
 99.000%    5.13ms
 99.900%   17.47ms
 99.990%   25.18ms
 99.999%   28.83ms
100.000%   28.88ms

#[Mean    =        1.214, StdDeviation   =        1.204]
#[Max     =       28.864, Total count    =        99744]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  199888 requests in 20.00s, 11.82MB read
Requests/sec:   9994.70
Transfer/sec:    605.15KB
```

Видно, что пропусная способность сильно не пострадала, но перцентили, начиная с 99.9, стали значительно медленнее. Поскольку обработка запросов, которые работают с хранилищем, теперь происходит на отдельном пуле потоков, то запрос на получение статуса в новой асинхронной версии сервера работает быстро и при смешанной нагрузке.

2 Get range

Проведем нагрузку на сервер запросами на получение диапозона данных:

```
  $ wrk -t4 -c1024 -d1m -R10000 -s ./wrk/get_range.lua -L /
  > http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     3.04s     4.30s   13.30s    77.02%
    Req/Sec     3.11k   596.01     3.74k    25.00%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   32.90ms
 75.000%    6.76s
 90.000%   10.62s
 99.000%   12.35s
 99.900%   12.82s
 99.990%   12.94s
 99.999%   13.20s
100.000%   13.30s

#[Mean    =     3043.818, StdDeviation   =     4300.866]
#[Max     =    13295.616, Total count    =       581351]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  585324 requests in 1.00m, 221.81MB read
  Socket errors: connect 8, read 0, write 0, timeout 3192
Requests/sec:   9755.20
Transfer/sec:      3.70MB
```

Рассмотрим результаты профилирования в режимах *cpu* (рис. 1) и *alloc* (рис. 2).

На рис. 1 видно, что запросы обрабатывались в селекторах. В каждом селекторе время ушло на запись в сокет (9%) и работу с итераторами (их создание (6.51%) и взятие следующего значения (4.18%)). Что касается памяти, то согласно рис. 2 можно заметить, что больше всего аллокаций было при создании итератора у *SSTable* ($\approx 15\%$) и при взятии следующего значения у итератора ($\approx 9.5\%$).

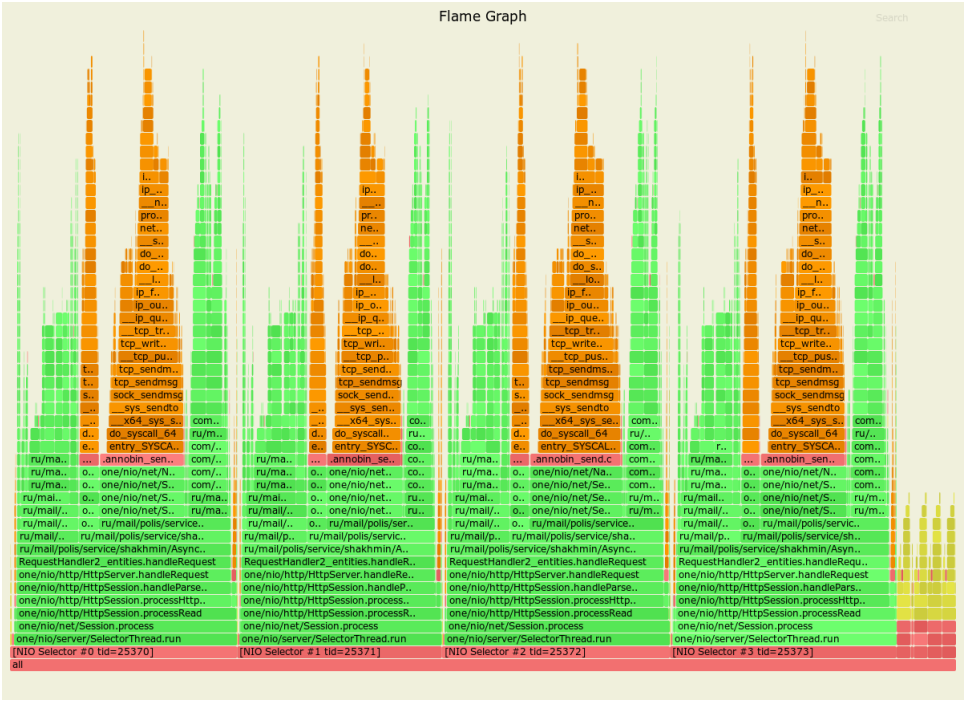
Flame Graph
[NIO Selector #0 tid=25370]
[NIO Selector #1 tid=25371]
[NIO Selector #2 tid=25372]
[NIO Selector #3 tid=25373]
all

Рис. 1

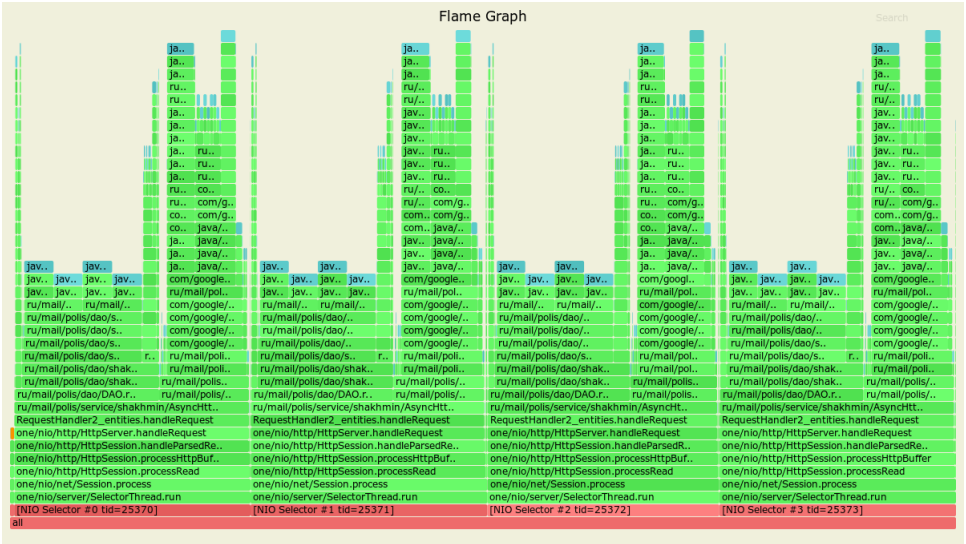
Flame Graph
[NIO Selector #0 tid=25370]
[NIO Selector #1 tid=25371]
[NIO Selector #2 tid=25372]
[NIO Selector #3 tid=25373]
all

Рис. 2

Рассмотрим теперь как изменились результаты профилирования на асинхронном сервере у других запросов.

3 PUT

Проведем нагрузку на сервер запросами на вставку данных:

```
  $ wrk -t 4 -c 1024 -d1m -R10000 -s ./wrk/put.lua -L http://localhost:8080

    Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.80ms    3.14ms  55.49ms   91.13%
    Req/Sec     2.60k     1.79k   17.56k    66.08%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.81ms
 75.000%    3.47ms
 90.000%    5.65ms
 99.000%   15.57ms
 99.900%   36.03ms
 99.990%   49.63ms
 99.999%   53.47ms
100.000%   55.52ms

#[Mean    =        2.796, StdDeviation   =        3.143]
#[Max     =       55.488, Total count    =       483210]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  587469 requests in 1.00m, 37.54MB read
  Socket errors: connect 8, read 0, write 0, timeout 232
Requests/sec:   9790.71
Transfer/sec:    640.60KB
```

Сравним *HdrHistogram* с диаграммой, получившиейся при той же нагрузке на блокирующейся версии сервера:

```
  $ wrk -t 4 -c 1024 -d1m -R10000 -s ./wrk/put.lua -L http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     6.80ms    5.65ms  68.99ms   78.16%
    Req/Sec     2.52k     2.46k   11.74k    82.31%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    5.57ms
 75.000%    9.52ms
 90.000%   13.27ms
 99.000%   26.88ms
 99.900%   47.33ms
 99.990%   64.10ms
 99.999%   68.16ms
100.000%   69.06ms
```

```
#[Mean    =        6.795, StdDeviation   =        5.650]
#[Max     =       68.992, Total count    =       483017]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  585432 requests in 1.00m, 37.41MB read
  Socket errors: connect 8, read 0, write 0, timeout 232
Requests/sec:   9756.65
Transfer/sec:    638.37KB
```

Можно увидеть, что асинхронная реализация дала прирост на всех перцентилях и небольшой прирост пропускной способности.

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 3) , *alloc* (рис. 4) и *lock* (рис. 5).

На (рис. 3) видно, что теперь запросы обрабатываются отдельным пулом потоков, а не на селекторах. Сама вставка в каждом воркере заняла не более 2%, большую часть времени заняла запись в сокет (8%).

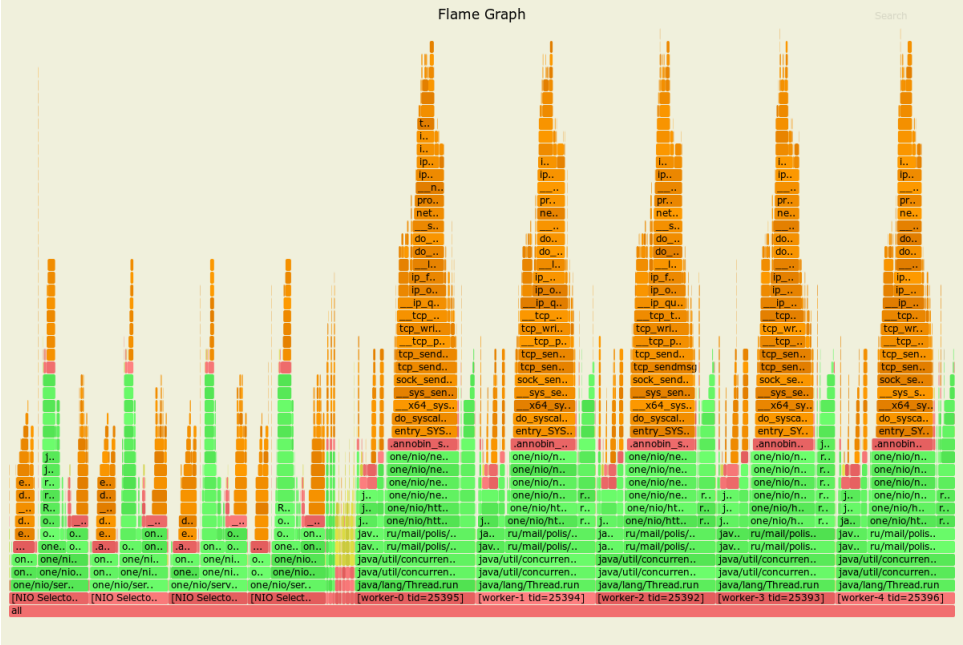

Рис. 3

На (рис. 4) можно заметить, что выделение памяти в селекторах в основном происходило при обработке полученного запроса. На воркере память была выделена при записи данных в хранилище ($\approx 6\%$) и при отправке ответа в сокет (2.5%).

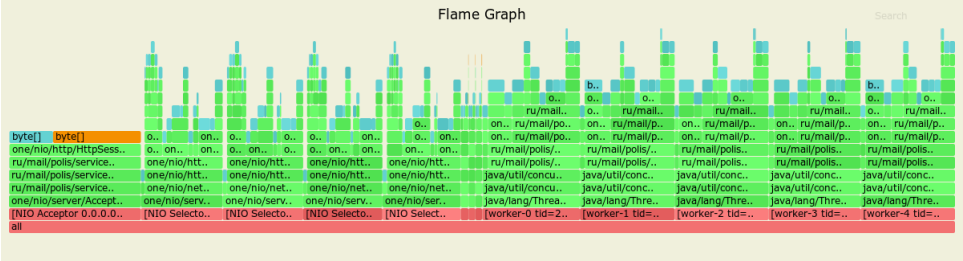

Рис. 4

На (рис. 5) видно, что воркеры блокировались, чтобы взять задачу на обработку (4.5%), и, чтобы записать данные в хранилище ($\approx 9\%$). Селекторы блокировались только, чтобы отдать задачу на выполенение в пул потоков ($\approx 4\%$).

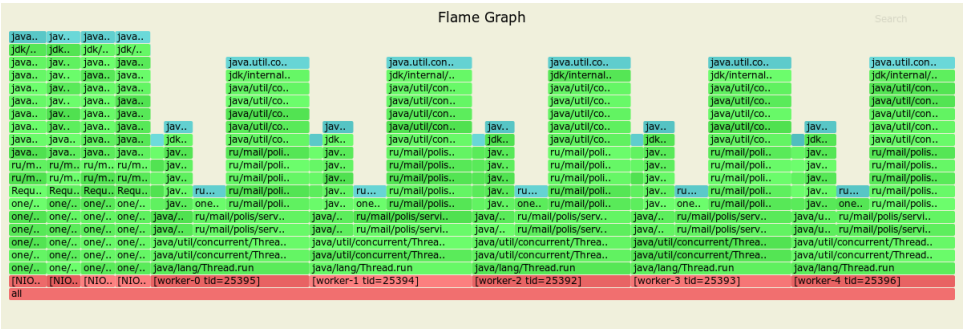

Рис. 5

4 GET

Проведем нагрузку на сервер запросами на получение данных:

```
$ wrk -t 4 -c 1024 -d1m -R10000 -s ./wrk/get.lua -L http://localhost:8080

Thread Stats   Avg      Stdev     Max   +/- Stdev
  Latency    21.79ms   16.57ms  98.69ms   63.01%
  Req/Sec     2.50k   773.01     5.26k    72.75%
Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   19.10ms
 75.000%   33.63ms
 90.000%   45.06ms
 99.000%   64.42ms
 99.900%   78.97ms
 99.990%   93.06ms
 99.999%   97.54ms
100.000%   98.75ms
```

```
#[Mean    =       21.793, StdDeviation   =       16.574]
#[Max     =       98.688, Total count    =       483401]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  585302 requests in 1.00m, 40.84MB read
  Socket errors: connect 8, read 0, write 0, timeout 232
Requests/sec:   9754.44
Transfer/sec:    697.03KB
```

Сравним *HdrHistogram* с диаграммой, получившиейся при той же нагрузке на блокирующейся версии сервера:

```
  $ wrk -t 4 -c 1024 -d1m -R10000 -s ./wrk/get.lua -L http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    22.87ms   19.20ms 142.98ms   70.50%
    Req/Sec     2.49k   642.45     4.52k    66.67%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   18.78ms
 75.000%   32.38ms
 90.000%   49.73ms
 99.000%   81.92ms
 99.900%  109.44ms
 99.990%  125.44ms
 99.999%  135.29ms
100.000%  143.10ms

#[Mean    =       22.868, StdDeviation   =       19.204]
#[Max     =      142.976, Total count    =       483561]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  585648 requests in 1.00m, 40.87MB read
  Socket errors: connect 7, read 0, write 0, timeout 203
Requests/sec:   9760.24
Transfer/sec:    697.45KB
```

Можно увидеть, что на асинхронной реализции произошел прирост с 99.9 перцентили.

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 6) , *alloc* (рис. 7) и *lock* (рис. 8).

На (рис. 6) видно, что как и раньше при обработке запроса на чтение в каждом воркере большую часть времени заняло получение итератора у *LSMDao* (≈ 13%).

Рис. 6

На (рис. 7) можно заметить, что выделение памяти в селекторах в основном происходило при обработке полученного запроса. На воркере большая часть памяти была выделена объектам класса *DirectByteBuffer*, которые создавались при вызове метода *SSTable.iterator*

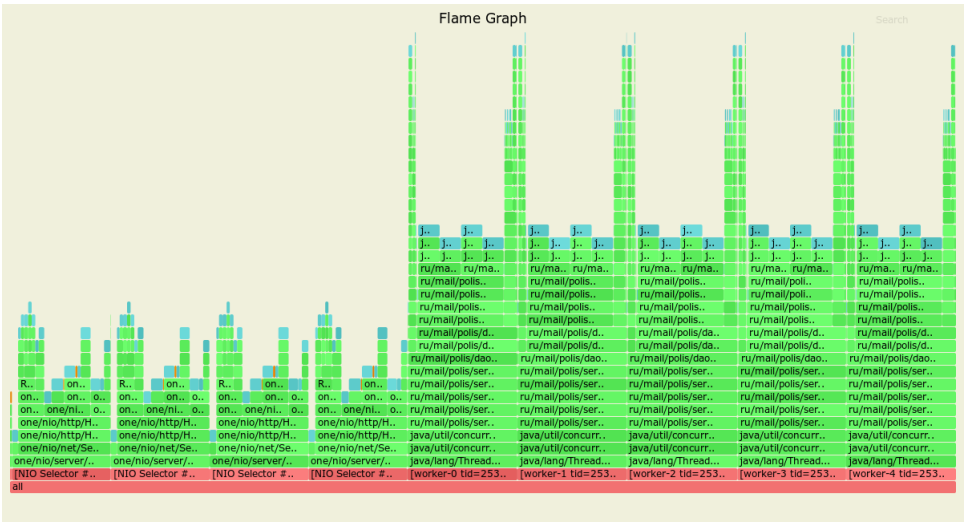

Рис. 7

На (рис. 8) видно, что поскольку мы использовали *ReadWriteLock* , то воркеры блокировались только, чтобы взять задачу на обработку ($\approx$ 17%), а селекторы блокировались, чтобы отдать задачу на выполенение в пул потоков ($\approx$ 2%).

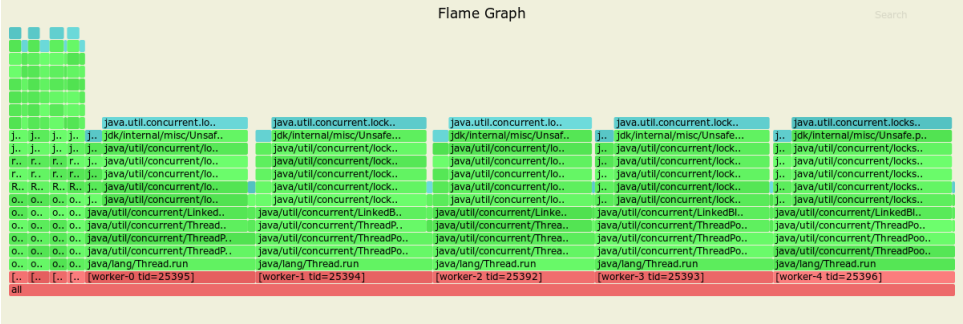

Рис. 8

5 Delete

Проведем нагрузку на сервер запросами на удаление данных:

```
  $ wrk -t 4 -c 1024 -d1m -R10000 -s ./wrk/delete.lua -L http://localhost
      :8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     3.51ms    4.36ms  66.24ms   90.30%
    Req/Sec     2.59k     1.74k   15.30k    68.86%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.88ms
 75.000%    4.28ms
 90.000%    7.73ms
 99.000%   21.63ms
 99.900%   40.61ms
 99.990%   58.21ms
 99.999%   64.13ms
100.000%   66.30ms


#[Mean    =        3.506, StdDeviation   =        4.360]
#[Max     =       66.240, Total count    =       483247]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  587504 requests in 1.00m, 38.10MB read
  Socket errors: connect 8, read 0, write 0, timeout 232
Requests/sec:   9791.19
Transfer/sec:    650.20KB
```

Сравним *HdrHistogram* с диаграммой, получившиейся при той же нагрузке на блокирующейся версии сервера:

```
  $ wrk -t 4 -c 1024 -d1m -R10000 -s ./wrk/delete.lua -L http://localhost
      :8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     8.97ms    7.41ms  63.84ms   72.02%
    Req/Sec     2.53k     2.39k    8.33k    77.93%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    7.60ms
 75.000%   12.26ms
 90.000%   18.66ms
 99.000%   34.43ms
 99.900%   51.10ms
 99.990%   62.17ms
 99.999%   63.71ms
100.000%   63.87ms

#[Mean    =        8.974, StdDeviation   =        7.412]
#[Max     =       63.840, Total count    =       483466]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  586041 requests in 1.00m, 38.00MB read
  Socket errors: connect 7, read 0, write 0, timeout 203
Requests/sec:   9766.68
Transfer/sec:    648.57KB
```

Можно увидеть, что на асинхронной реализции прирост в основном получился на малых перцентилях.

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 9) , *alloc* (рис. 10) и *lock* (рис. 11).

На (рис. 9) видно, что как и раньше при обработке запроса на удаление в каждом воркере большую часть времени заняла запись в сокет (7%), а само удаление было быстрым ($\approx 2.4\%$)

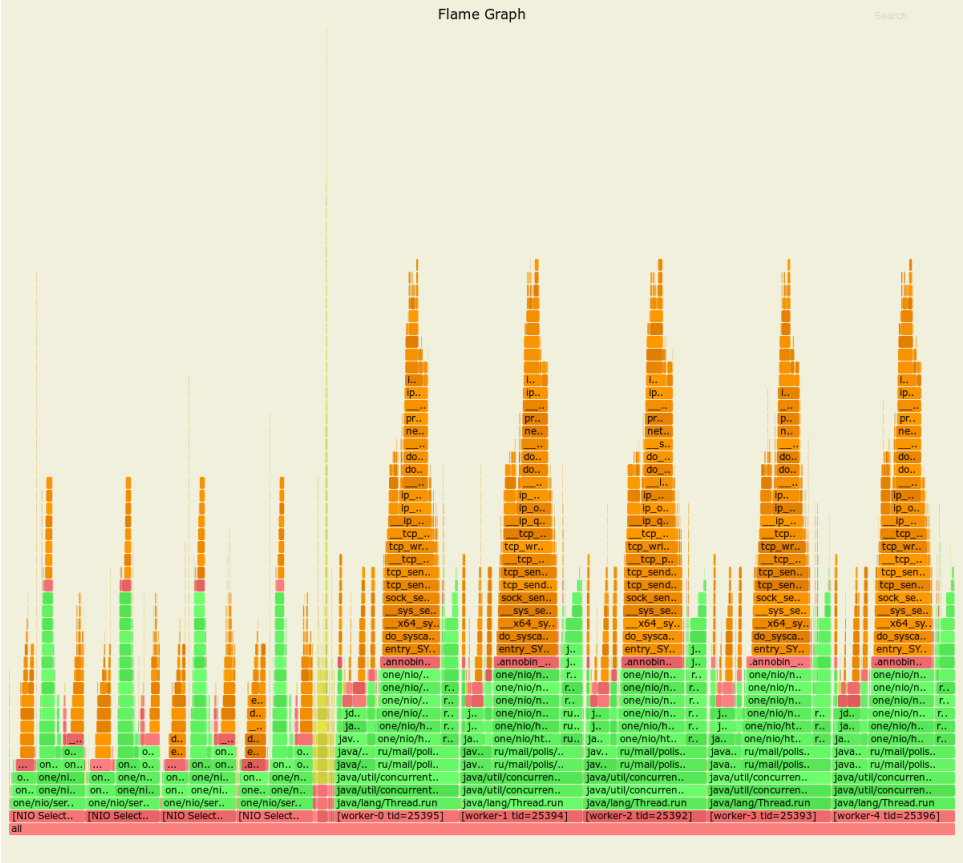

Рис. 9

На (рис. 10) можно заметить, что выделение памяти в селекторах в основном происходило при обработке полученного запроса. На воркере память была выделена при удалении данных из хранилища ($\approx 6.3\%$) и при отправке ответа в сокет ($\approx 2.7\%$).

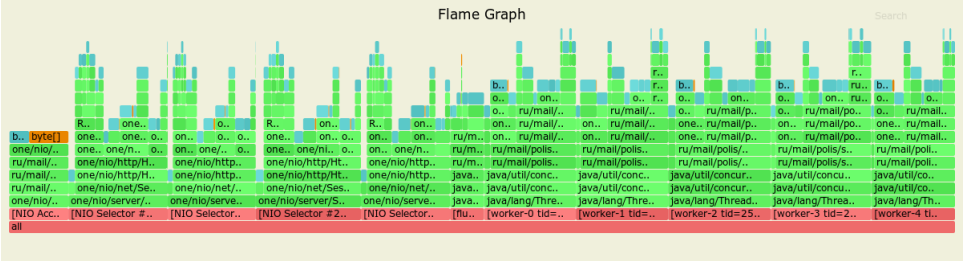

Рис. 10

На (рис. 11) видно, что воркеры блокировались, чтобы взять задачу на обработку ($\approx 3.8\%$), и, чтобы записать данные в хранилище ($\approx 12.4\%$). Селекторы блокировались только, чтобы отдать задачу на выполенение в пул потоков ($\approx 2.3\%$).

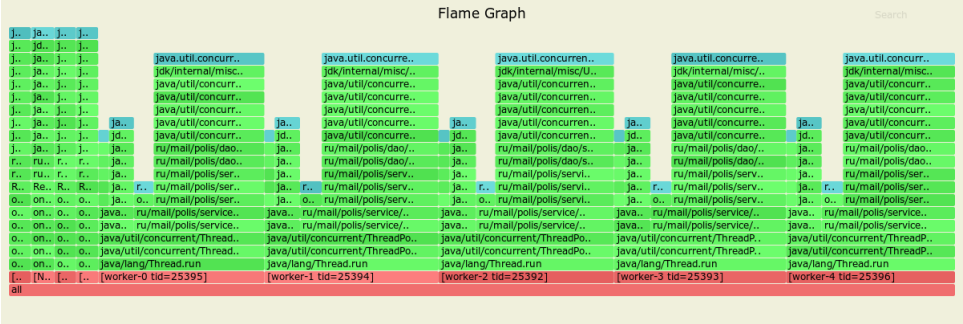

Рис. 11