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

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

Сравним результаты с предыдущей версией сервера.

1 Put

Проведем нагрузку на сервер запросами на вставку данных:

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

    Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     5.86ms    6.00ms  44.10ms   84.72%
    Req/Sec     5.10k     1.18k   10.65k    76.62%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    3.42ms
 75.000%    8.69ms
 90.000%   14.49ms
 99.000%   25.53ms
 99.900%   35.49ms
 99.990%   41.79ms
 99.999%   43.58ms
100.000%   44.13ms

#[Mean    =        5.858, StdDeviation   =        5.999]
#[Max     =       44.096, Total count    =       499740]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  599877 requests in 1.00m, 38.33MB read
  Non-2xx or 3xx responses: 15
Requests/sec:   9997.87
Transfer/sec:    654.16KB
```

Сравним HdrHistogram с диаграммой, получившиейся на прошлой версии сервера:

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

   Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.63ms    3.31ms  45.73ms   96.08%
    Req/Sec     2.65k   419.46     7.55k    85.71%
  Latency Distribution (HdrHistogram - Recorded Latency)
```

```
 50.000%    1.05ms
 75.000%    1.51ms
 90.000%    1.86ms
 99.000%   21.22ms
 99.900%   36.54ms
 99.990%   41.98ms
 99.999%   45.41ms
100.000%   45.76ms

#[Mean    =        1.626, StdDeviation   =        3.315]
#[Max     =       45.728, Total count    =       499946]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  599989 requests in 1.00m, 47.32MB read
Requests/sec:   9999.82
Transfer/sec:    807.51KB
```

Можно увидеть, что после добавления репликации увеличились малые перцентили, а на высоких перцентилях значения не изменились.

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 1) , *alloc* (рис. 2) и *lock* (рис. 3).

На рис. 1 видно, что вставка в каждом воркере заняла не более 0.5%, а все остальное время ушло теперь на работу с сетью: запись ответа в сокет занимает 1.3%, получение ответов от реплик занимает 1.5%, на проксирование тратится 5% времени.

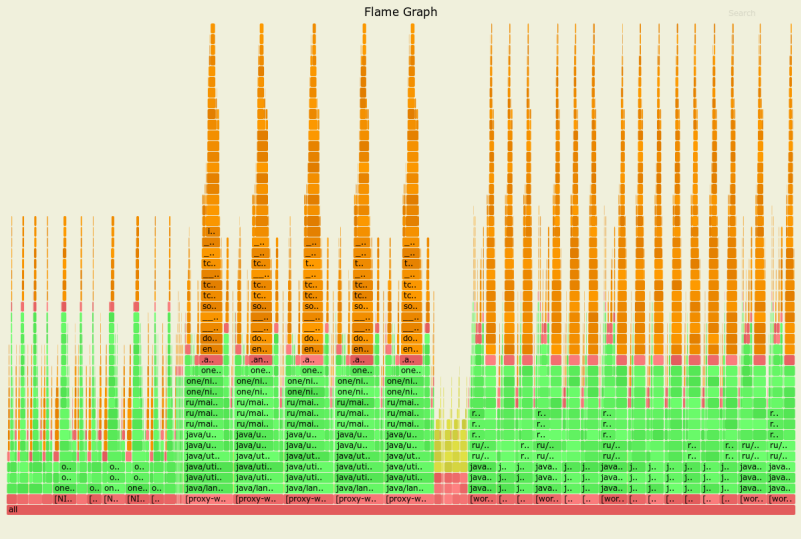

Рис. 1

Что касается аллокации памяти (рис. 2), то большая ее часть выделяется для работы с сетью, например, на запись данных в хранилище отводится не более 0.5%, а при проксировании запроса было выделено 6% памяти.

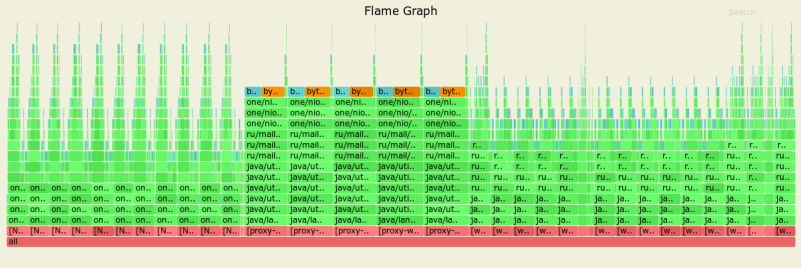

Рис. 2

В режиме *lock* рис. 3 было замечено, что воркеры блокировались при вставке данных (2%), при отпраке ответов на реплики (5%), при работе с *FutureTask* (5%).

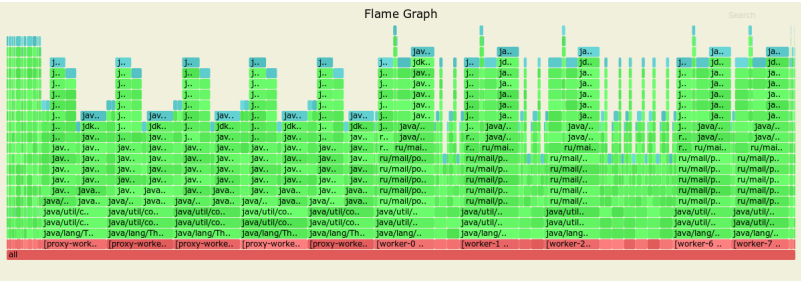

Рис. 3

2 Delete

Проведем нагрузку на сервер запросами на удаление данных:

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

    Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.26ms    2.86ms  47.84ms   92.01%
    Req/Sec     4.09k   544.90     8.58k    87.25%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.50ms
 75.000%    2.24ms
 90.000%    4.30ms
 99.000%   16.07ms
 99.900%   26.69ms
 99.990%   43.65ms
 99.999%   47.17ms
100.000%   47.87ms

#[Mean    =        2.263, StdDeviation   =        2.861]
#[Max     =       47.840, Total count    =       399779]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  492988 requests in 1.00m, 31.97MB read
```

```
  Socket errors: connect 0, read 0, write 0, timeout 50
  Non-2xx or 3xx responses: 3
Requests/sec:   8216.58
Transfer/sec:    545.63KB
```

Сравним HdrHistogram с диаграммой, получившиейся на прошлой версии сервера:

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

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.75ms    3.80ms  41.82ms   95.99%
    Req/Sec     2.64k   407.98     7.22k    86.86%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.06ms
 75.000%    1.52ms
 90.000%    1.87ms
 99.000%   26.00ms
 99.900%   36.03ms
 99.990%   41.02ms
 99.999%   41.60ms
100.000%   41.85ms

#[Mean    =        1.747, StdDeviation   =        3.795]
#[Max     =       41.824, Total count    =       499942]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  599980 requests in 1.00m, 47.89MB read
Requests/sec:   9999.87
Transfer/sec:    817.28KB
```

Можно увидеть, добавление репликации немного увеличило значения на всех перцентилях, при этом также пострадала и пропускная способность.

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 4) , *alloc* (рис. 5) и *lock* (рис. 6).

На рис. 4 видно, что удаление в каждом воркере заняло не более 0.4%, а все остальное время ушло на работу с сетью ($\approx 8\%$).

Что касается аллокации памяти (рис. 5), то большая ее часть выделяется для работы с сетью, например, при удалении данных из хранилища отводится не более 0.6%, а при проксировании запроса было выделено 6% памяти.

В режиме *lock* (рис. 6) было замечено, что воркеры блокировались при удалении данных (3%), при отпраке запросов на реплики (4.3%), при работе с *FutureTask* (4%).

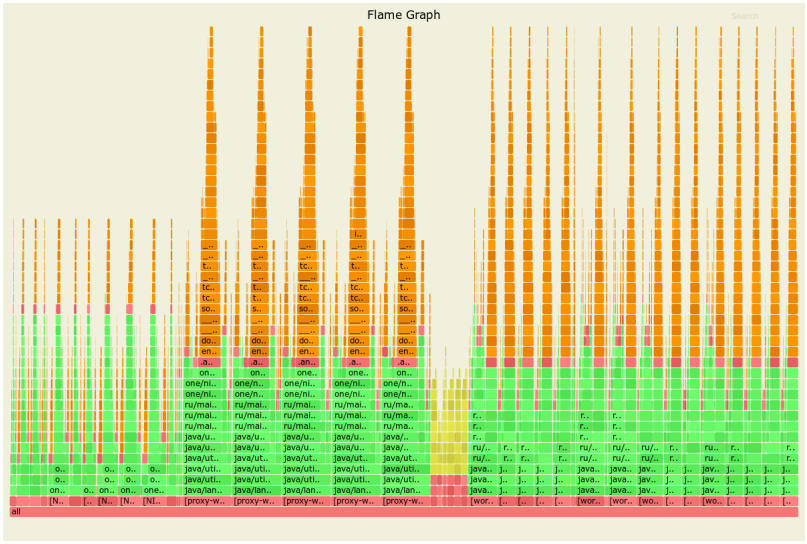

Рис. 4

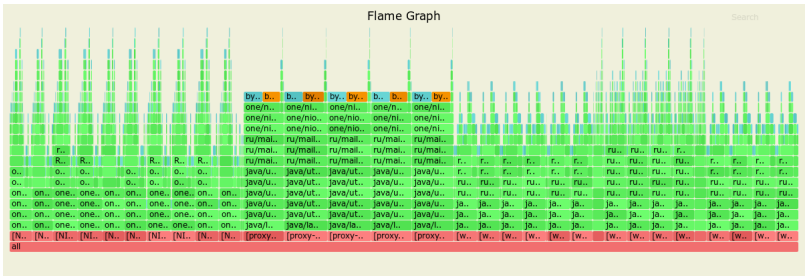

Рис. 5

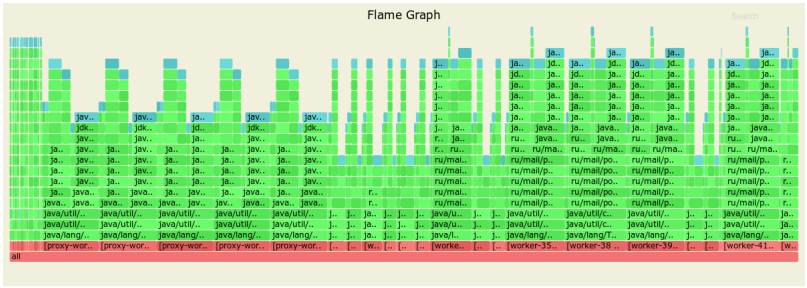

Рис. 6

3 Get

Проведем нагрузку на сервер запросами на получение данных (хранилище заполнено на 100M):

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

Thread Stats   Avg      Stdev     Max   +/- Stdev
  Latency     8.10s      3.28s   13.95s    58.72%
```

```
    Req/Sec     3.86k   120.71     4.10k    68.18%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    8.18s
 75.000%   10.97s
 90.000%   12.63s
 99.000%   13.70s
 99.900%   13.90s
 99.990%   13.95s
 99.999%   13.96s
100.000%   13.96s

#[Mean    =     8103.865, StdDeviation   =     3275.039]
#[Max     =    13950.976, Total count    =       386315]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  462502 requests in 1.00m, 32.41MB read
Requests/sec:   7708.41
Transfer/sec:    553.12KB
```

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

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

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     9.73ms   12.24ms  72.19ms   82.40%
    Req/Sec     5.04k     1.12k    9.22k    67.27%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    2.45ms
 75.000%   16.40ms
 90.000%   28.93ms
 99.000%   48.42ms
 99.900%   61.95ms
 99.990%   67.58ms
 99.999%   71.55ms
100.000%   72.25ms

#[Mean    =        9.730, StdDeviation   =       12.237]
#[Max     =       72.192, Total count    =       499947]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  599886 requests in 1.00m, 51.10MB read
Requests/sec:   9997.93
Transfer/sec:      0.85MB
```

Можно увидеть, что добавление репликации значительно увеличило значения на всех перцентилях, при этом также пострадала и пропускная способность.

Рассмотрим результаты профилирования трех режимов: *cpu* (рис. 7) , *alloc* (рис. 8) и *lock* (рис. 9).

На рис. 7 видно, что как и раньше при обработке запроса на чтение в каждом воркере большую часть времени заняло получение итератора у *LSMDao* ($\approx 3.3\%$), а работа с сетью $\approx 4\%$ времени.

Рис. 7

Что касается аллокации памяти (рис. 8), большая часть памяти была выделена объектам класса *DirectByteBuffer*, которые создавались при вызове метода *SSTable.iterator* (2%), а также при работе с *FutureTask* (6%). В селекторах при обработке запроса было выделено $\approx 3\%$ памяти.

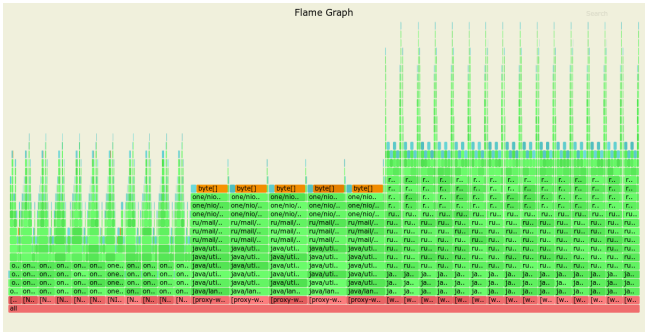

Рис. 8

На рис. 9 видно, что поскольку мы использовали *ReadWriteLock*, то воркеры блокировались только при работе с FutureTask ($\approx 0.1\%$), и при отпраке запросов на реплики (0.25%).

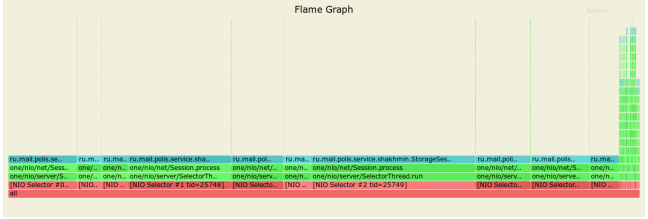

Рис. 9