Проведем нагрузочное тестирование нашего потокобезопасного, асинхронного сервера с помощью *wrk* в несколько соединений после переключения внутреннего взаимодействия узлов на асинхронный *java.net.http.HttpClient*. 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 -c 128 -d1m -R1000 -s ./wrk/put.lua -L \
  > http://localhost:8080

    Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.53ms    2.29ms  32.80ms   94.90%
    Req/Sec   527.38    187.54     2.44k    80.24%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    2.08ms
 75.000%    2.47ms
 90.000%    3.02ms
 99.000%   14.57ms
 99.900%   23.97ms
 99.990%   30.32ms
 99.999%   32.83ms
100.000%   32.83ms

#[Mean    =        2.529, StdDeviation   =        2.293]
#[Max     =       32.800, Total count    =        49676]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  59906 requests in 1.00m, 3.83MB read
Requests/sec:    998.42
Transfer/sec:     65.33KB
```

Сравним HdrHistogram с диаграммой, получившиейся на предыдущей версии сервера:

```
  $ wrk -c 128 -d1m -R1000 -s ./wrk/put.lua -L \
  > http://localhost:8080

  Thread calibration: mean lat.: 1.866ms, rate sampling interval: 10ms
  Thread calibration: mean lat.: 5.077ms, rate sampling interval: 14ms
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     3.39ms    2.36ms  25.41ms   82.07%
    Req/Sec   522.15    839.14     4.92k    94.32%
```

```
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    2.57ms
 75.000%    4.80ms
 90.000%    6.11ms
 99.000%   11.73ms
 99.900%   20.33ms
 99.990%   22.38ms
 99.999%   25.42ms
100.000%   25.42ms

#[Mean    =        3.387, StdDeviation   =        2.357]
#[Max     =       25.408, Total count    =        49670]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  59458 requests in 1.00m, 3.80MB read
Requests/sec:    990.80
Transfer/sec:     64.83KB
```

Можно увидеть, что на новой версии сервера увеличились как малые, так и высокие перцентили, но незначительно, например, 99 перцентиль увеличился на 7 миллисекунд. Стоит отметить, что на новой версии наблюдается прирост пропускной способности.

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

На рис. 1 видно, что больше всего времени заняла работа воркеров: вызовы методов у *CompletableFuture*, работа с сетью. Так же видна работа *ForkJoinPool*, поскольку он обрабатывал задачи, которые были прокинуты в *CompletableFuture*. На селекторах, как и раньше, происходит только обработка запросов.

Что касается аллокации памяти (рис. 2), то большая ее часть выделяется для работы с сетью и с *CompletableFuture*.

В режиме *lock* (рис. 3) видно, что блокировки происходили при работе с *CompletableFuture*, поскольку это инструмент межпотокового взаимодействия, также блокировки возникали с добавленным *HttpClient*, например, при обработке запросов и ответов. Блокировка, связанная с записью в хранилище, заняла малую часть времени (1%).

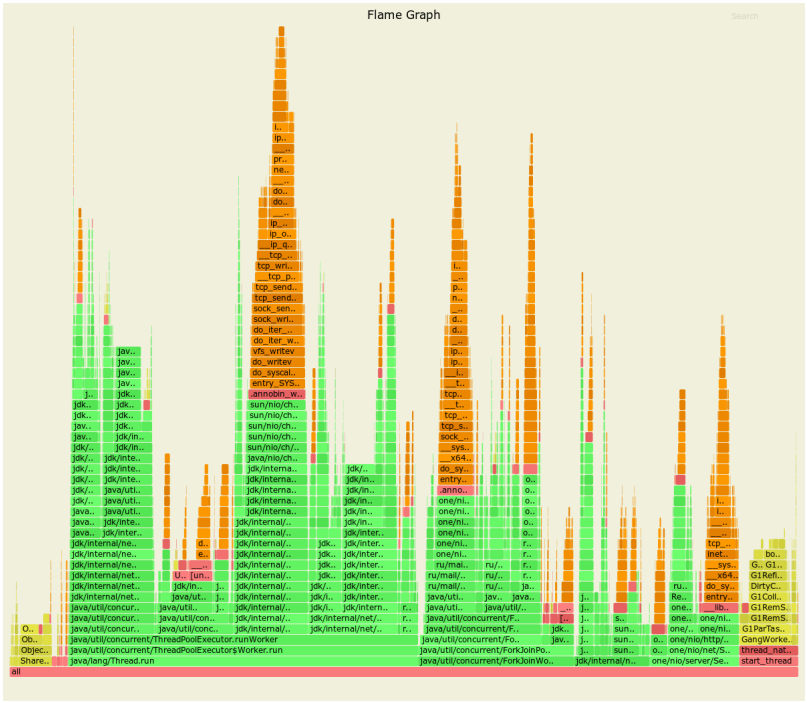

Рис. 1

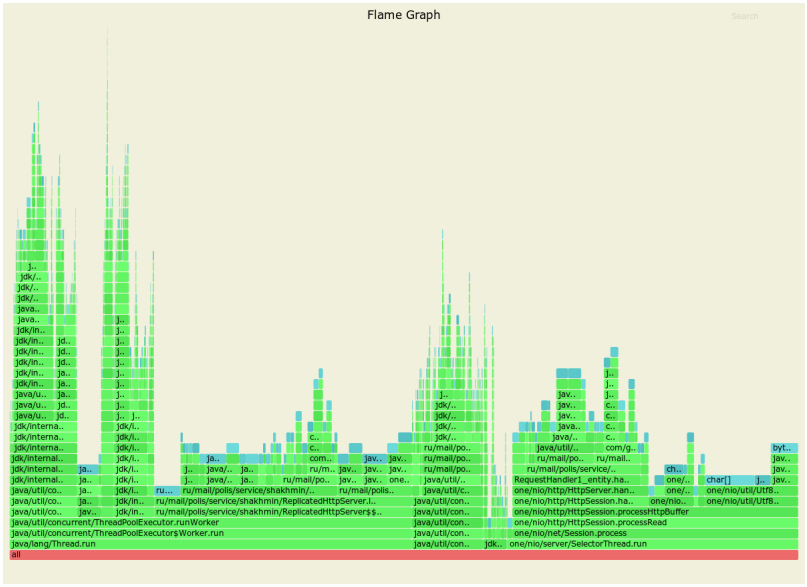

Рис. 2

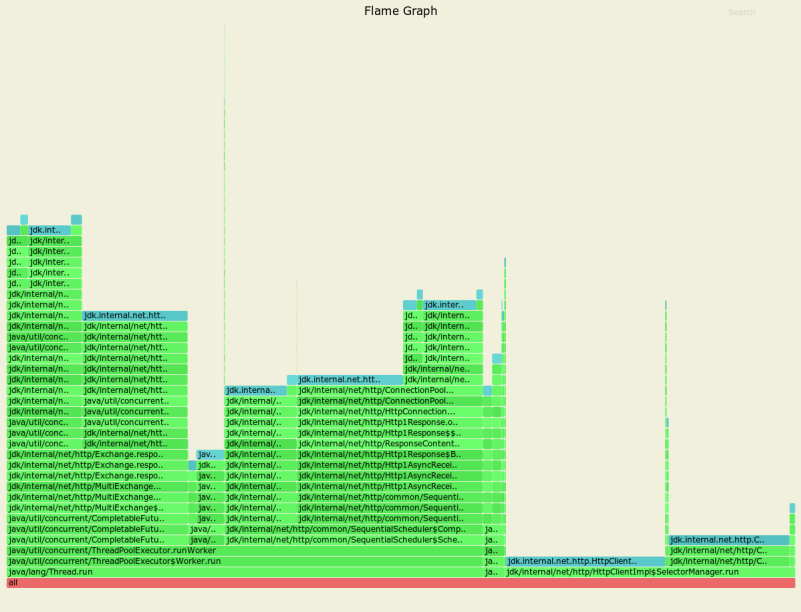

Рис. 3

2 Get

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

```
  $ wrk -c 128 -d1m -R1000 -s ./wrk/get.lua -L \
  > http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    18.65ms   14.74ms  71.68ms   51.47%
    Req/Sec   504.02    229.83     1.15k    69.61%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%   20.29ms
 75.000%   29.02ms
 90.000%   38.56ms
 99.000%   54.14ms
 99.900%   64.38ms
 99.990%   68.74ms
 99.999%   71.74ms
100.000%   71.74ms

#[Mean    =       18.653, StdDeviation   =       14.742]
#[Max     =       71.680, Total count    =        49670]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  59458 requests in 1.00m, 4.10MB read
Requests/sec:    990.64
Transfer/sec:     69.93KB
```

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

```
  $ wrk -c 128 -d1m -R1000 -s ./wrk/get.lua -L \
  > http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     4.81ms    4.03ms  31.98ms   81.00%
    Req/Sec   521.90    509.35     2.67k    73.35%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    2.74ms
 75.000%    7.48ms
 90.000%   10.38ms
 99.000%   18.21ms
 99.900%   25.17ms
 99.990%   29.04ms
 99.999%   32.00ms
100.000%   32.00ms

#[Mean    =        4.811, StdDeviation   =        4.027]
#[Max     =       31.984, Total count    =        49670]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  59462 requests in 1.00m, 4.10MB read
Requests/sec:    990.82
Transfer/sec:     69.95KB
```

Можно увидеть, что значительно увеличило значения на всех перцентилях, а пропускная способность не изменилась.

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

На рис. 4 видно, что больше всего времени заняла работа воркеров: вызовы методов у *CompletableFuture*, работа с сетью. Так же видна работа *ForkJoinPool*, обрабатывающий задачи *CompletableFuture*. Большую часть времени в *ForkJoinPool* заняло получение итератора у *LSMDao*.

Что касается аллокации памяти (рис. 5), то большая ее часть в *ForkJoinPool* была выделена объектам класса *DirectByteBuffer*, которые создавались при вызове метода *SSTable.iterator*. Остальная аллокация памяти происходила при работе с сетью и с *CompletableFuture*.

На рис. 6 видно, что блокировки были также связаны с межпотоковым взаимодействием *CompletableFuture* и с обработкой запросов/ответов у *HttpClient*.

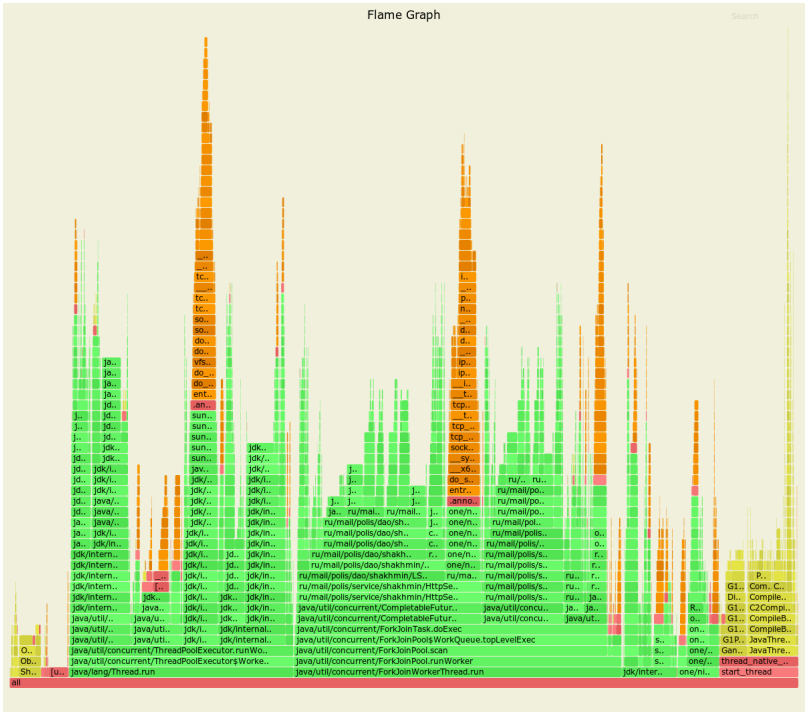

Рис. 4

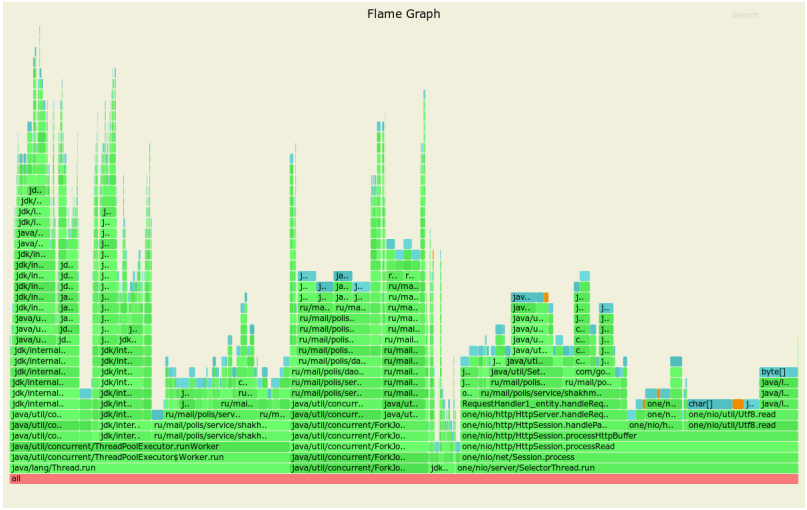

Рис. 5

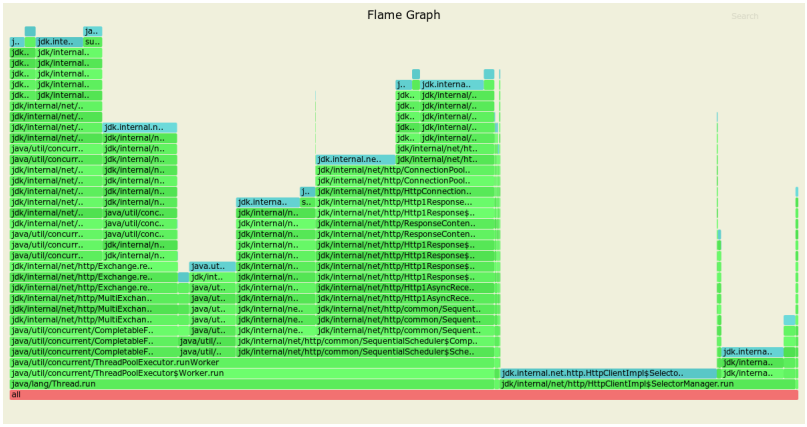

Рис. 6

3 Delete

Проведем нагрузку на сервер запросами на удаление данных:

```
  $ wrk -c 128 -d1m -R1000 -s ./wrk/delete.lua -L \
  > http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.93ms    1.75ms  57.82ms   97.87%
    Req/Sec   527.63    146.88     3.40k    80.55%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.74ms
 75.000%    2.10ms
 90.000%    2.51ms
 99.000%    7.13ms
 99.900%   26.91ms
 99.990%   54.27ms
 99.999%   57.85ms
100.000%   57.85ms

#[Mean    =        1.929, StdDeviation   =        1.749]
#[Max     =       57.824, Total count    =        49676]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  59906 requests in 1.00m, 3.88MB read
Requests/sec:    998.41
Transfer/sec:     66.30KB
```

Сравним HdrHistogram с диаграммой, получившиейся на прошлой версии сервера:

```
  $ wrk -c 128 -d1m -R1000 -s ./wrk/delete.lua \
  > http://localhost:8080

  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     3.04ms    2.17ms  37.86ms   85.17%
```

```
    Req/Sec   523.14      0.89k     5.33k     94.27%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    2.42ms
 75.000%    4.19ms
 90.000%    5.46ms
 99.000%    8.63ms
 99.900%   24.06ms
 99.990%   36.35ms
 99.999%   37.89ms
100.000%   37.89ms

#[Mean    =        3.040, StdDeviation   =        2.168]
#[Max     =       37.856, Total count    =        49670]
#[Buckets =           27, SubBuckets     =         2048]
----------------------------------------------------------
  59459 requests in 1.00m, 3.86MB read
Requests/sec:    990.82
Transfer/sec:     65.80KB
```

Можно увидеть, что на новой версии произошел прирост пропускной способности, уменьшение значений на малых перцентилях и увеличение высоких перцентилей.

Результаты профилирования (*cpu* (рис. 7) , *alloc* (рис. 8) и *lock* (рис. 9)) схожи с результатами, которые были получены при нагрузке сервера запросами на запись.

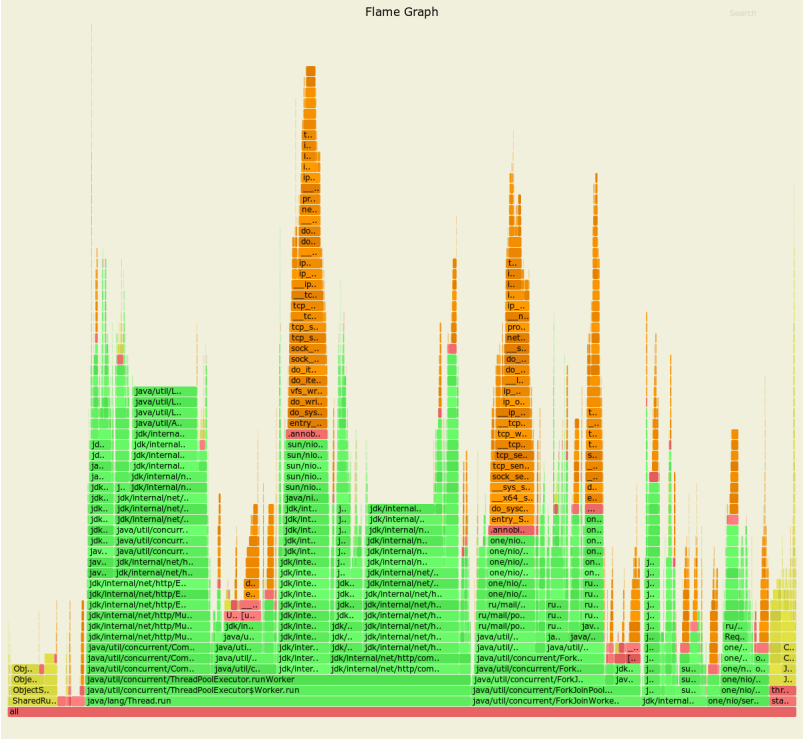

Рис. 7

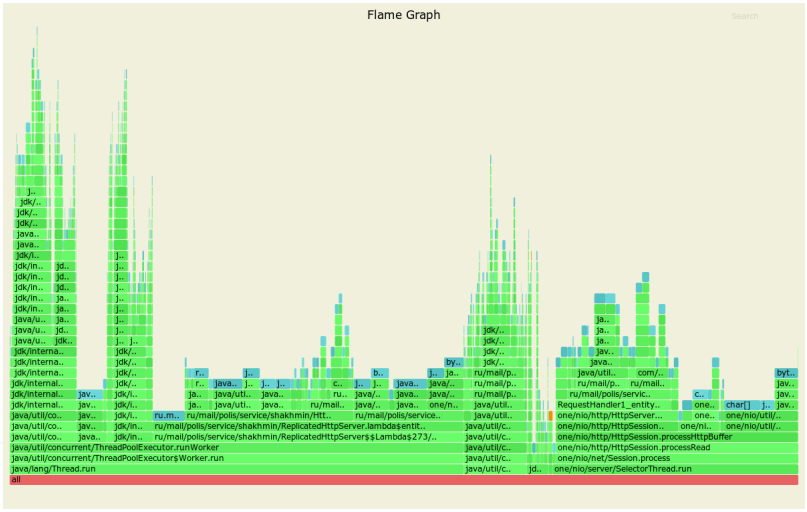
Flame Graph
Search
java/lang/Thread.run
all

Рис. 8

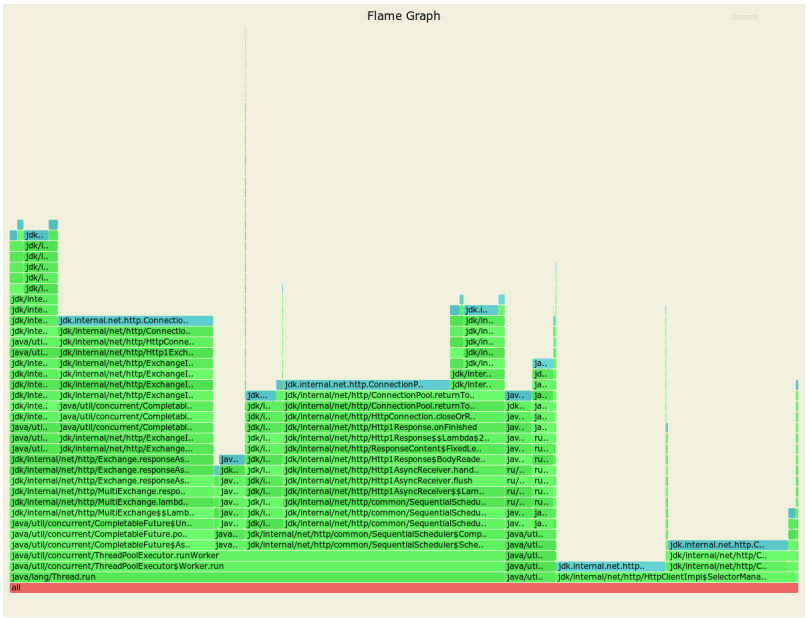
Flame Graph
Search
java/lang/Thread.run
all

Рис. 9