Проведем нагрузочное тестирование с помощью *wrk* в одно соединие. Проанализируем результаты профилирования под нагрузкой, проведенное с помощью *async-profiler*.

В начале заполним наше хранилище небольшим количеством данных (25M):

```
$ wrk -t1 -c1 -d60s -R10000 -s ./wrk/put.lua --latency
```

Если попрофилировать под данной нагрузкой в режиме *cpu*, то наибольшее количество времени занимает передача данных по сети. Сама вставка данных занимает $\approx 11\%$ всего времени (рис. 1).

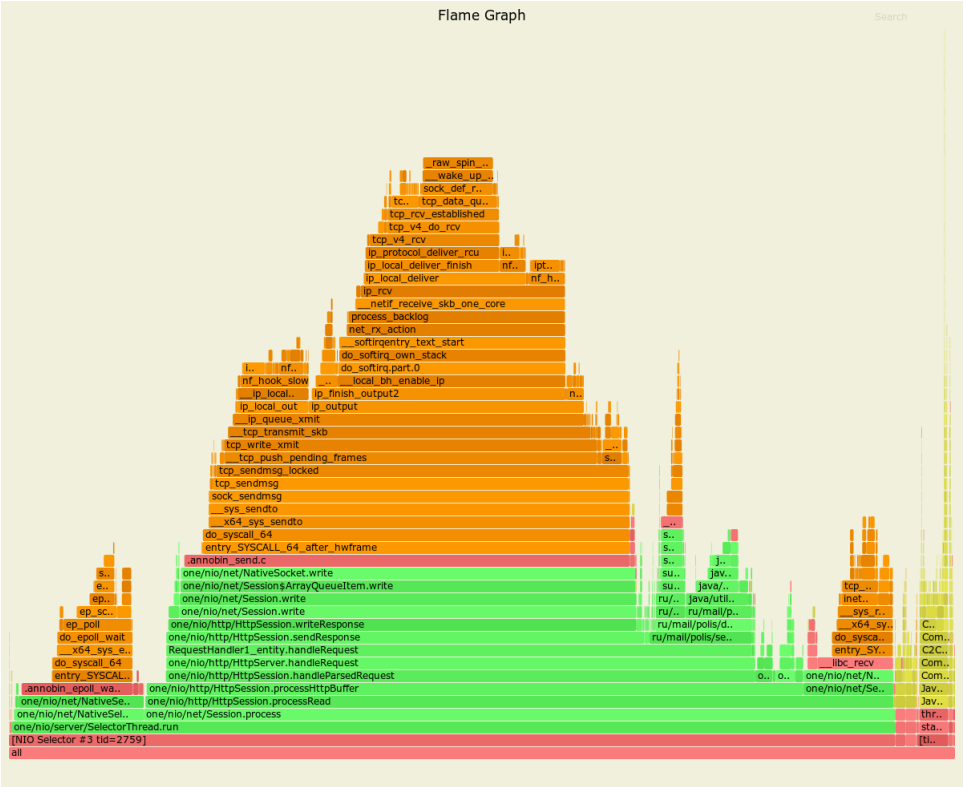

Рис. 1

Теперь проверим, как наш сервер будет обрабатывать 10000 запросов на чтение в секунду в течение минуты:

```
$ wrk -t1 -c1 -d60s -R10000 -s ./wrk/get.lua --latency
```

Как видно на рис. 2 и на рис. 3 на ряду с сетевыми вызовами большую часть времени ($\approx 31\%$) занимает вызов метода *iterator* класса *SSTable*.

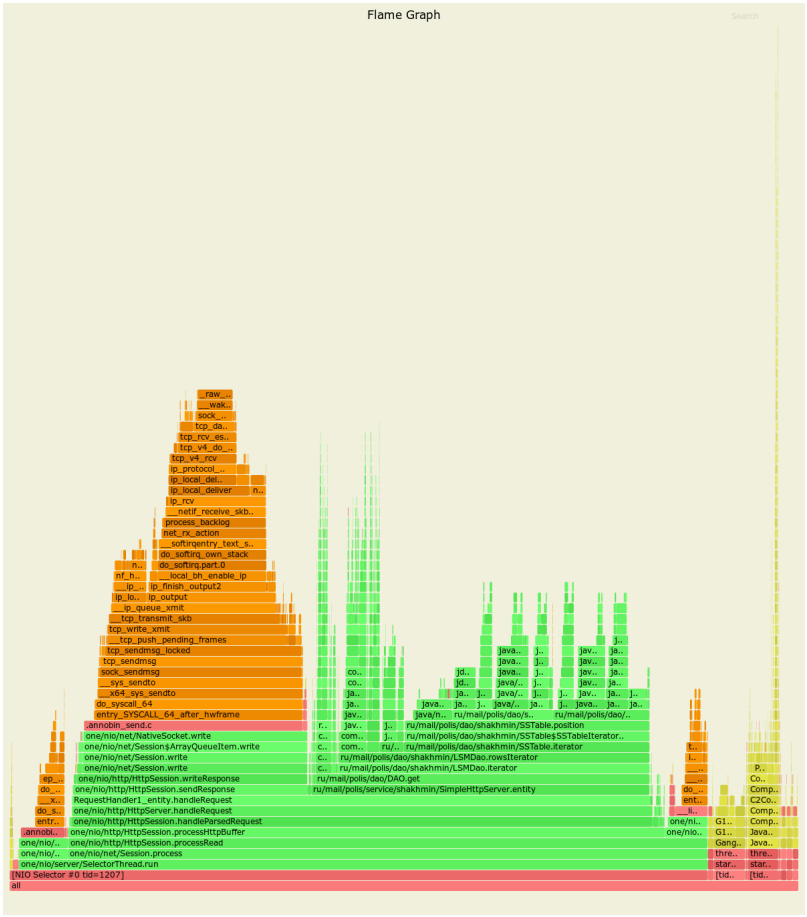
Flame Graph
Search
[NIO Selector #0 tid=1207]
all

Рис. 2

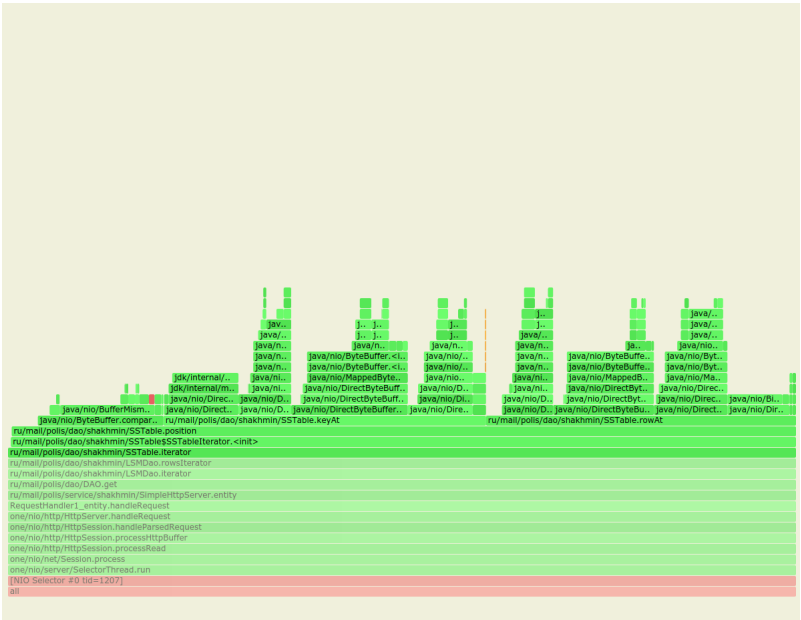
ru/mail/polis/dao/shakhmin/SSTable.position
ru/mail/polis/dao/shakhmin/SSTable$SSTableIterator.<init>
ru/mail/polis/dao/shakhmin/SSTable.iterator
ru/mail/polis/dao/shakhmin/LSMDao.rowsIterator
ru/mail/polis/dao/shakhmin/LSMDao.iterator
ru/mail/polis/dao/DAO.get
ru/mail/polis/service/shakhmin/SimpleHttpServer.entity
RequestHandler1_entity.handleRequest
one/nio/http/HttpServer.handleRequest
one/nio/http/HttpSession.handleParsedRequest
one/nio/http/HttpSession.processHttpBuffer
one/nio/http/HttpSession.processRead
one/nio/net/Session.process
one/nio/server/SelectorThread.run
[NIO Selector #0 tid=1207]
all

Рис. 3

Проверим, какие результаты профилирования под этой же нагрузкой будут в режиме *alloc*. На рис. 4 видно, что в куче большую часть памяти занимают объекты класса *DirectByteBuffer*. Просмотрев методы *keyAt* и *rowAt* класса *SSTable* было замечено, что вызов метода *ByteBuffer.dublicate* в них был лишним, поскольку уже был вызван метод *ByteBuffer.asReadOnly*, который внутри себя вызывает метод *ByteBuffer.dublicate*.

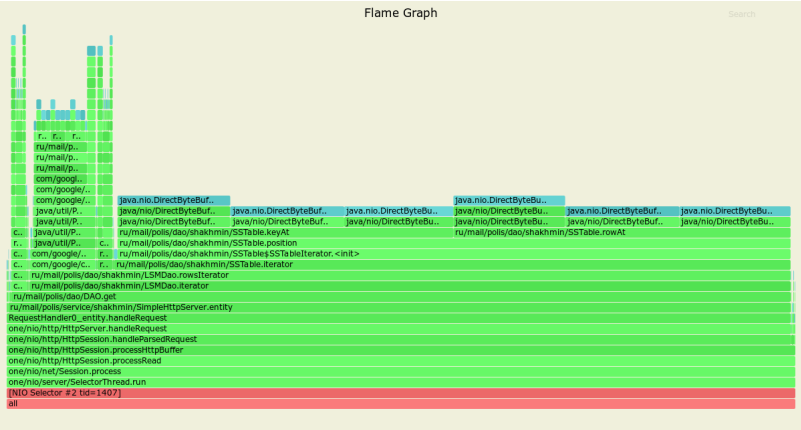

Рис. 4

Удалив лишний вызов метода (рис. 5), проверим: улучшилась ли ситуация при обрабатывании запросов на чтение. На рис. 6 и рис. 7 видны небольшие улучшения. Если раньше метод *SSTable.iterator* занимал $\approx 31\%$, то теперь он занимает $\approx 26\%$.

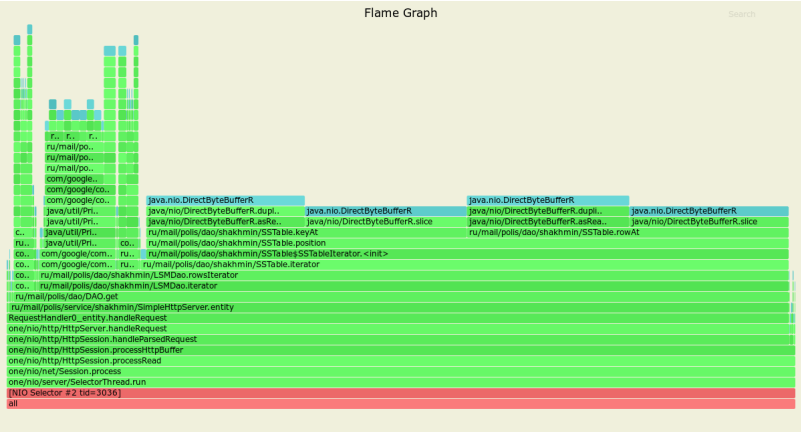

Рис. 5

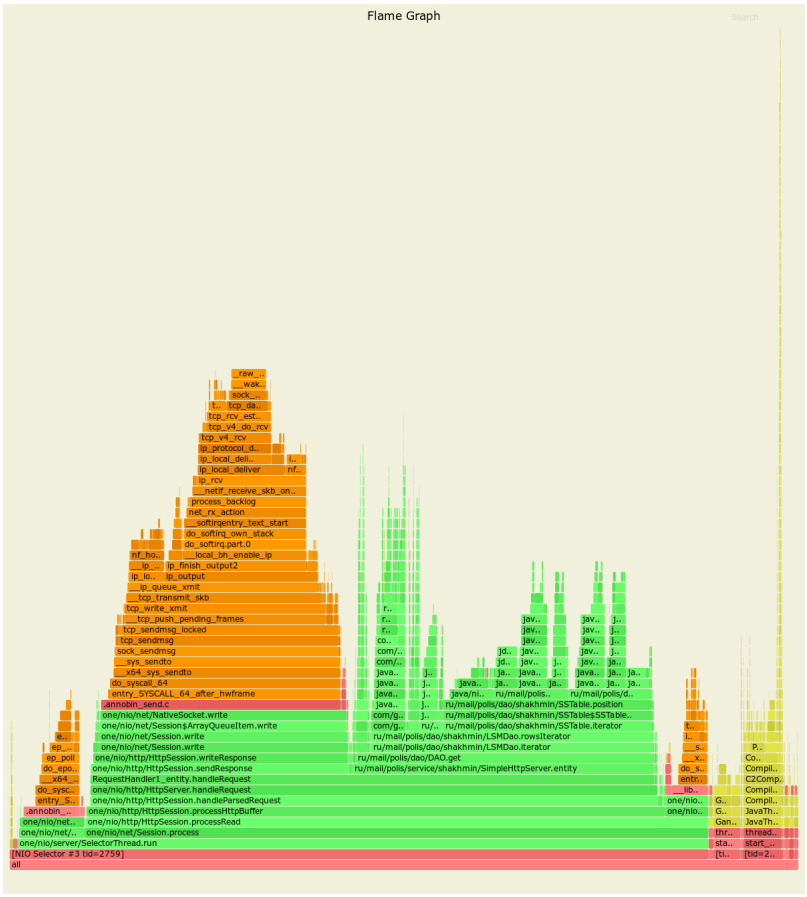
Flame Graph
Search
[NIO Selector #3 tid=2759]
all

Рис. 6

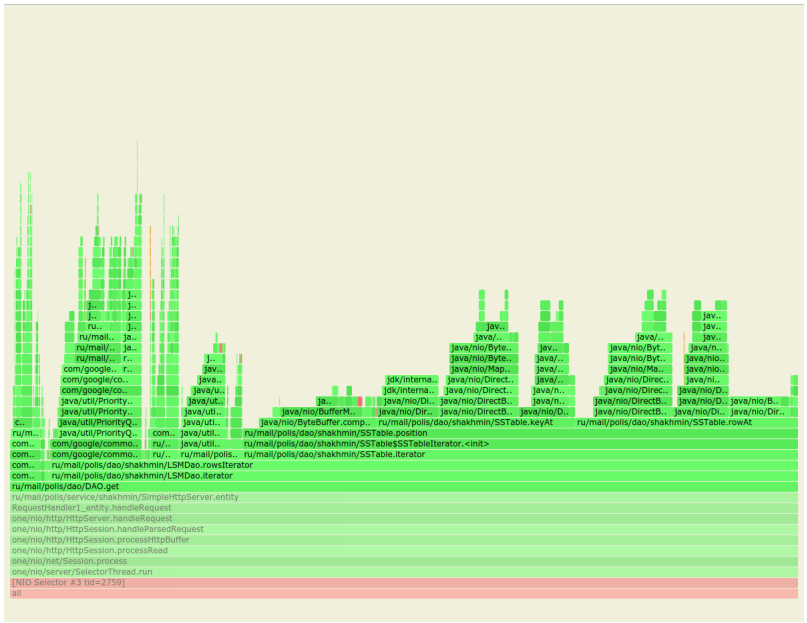
ru/mail/polis/dao/shakhmin/SSTable.position
ru/mail/polis/dao/shakhmin/SSTable$SSTableIterator.<init>
ru/mail/polis/dao/shakhmin/SSTable.iterator
ru/mail/polis/dao/shakhmin/LSMDao.rowsIterator
ru/mail/polis/dao/shakhmin/LSMDao.iterator
ru/mail/polis/dao/DAO.get
ru/mail/polis/service/shakhmin/SimpleHttpServer.entity
RequestHandler1_entity.handleRequest
one/nio/http/HttpServer.handleRequest
one/nio/http/HttpSession.handleParsedRequest
one/nio/http/HttpSession.processHttpBuffer
one/nio/http/HttpSession.processRead
one/nio/net/Session.process
one/nio/server/SelectorThread.run
[NIO Selector #3 tid=2759]
all

Рис. 7

Попробуем улучшить результаты за счет увеличения числа потоков нашего веб-сервера до 4-ех:

```
$ wrk -t4 -c4 -d60s -R10000 -s ./wrk/get.lua --latency
```

В результате получим, что обработка запросов на чтение при той же нагрузке в каждом потоке не превышало 12% (рис. 8).

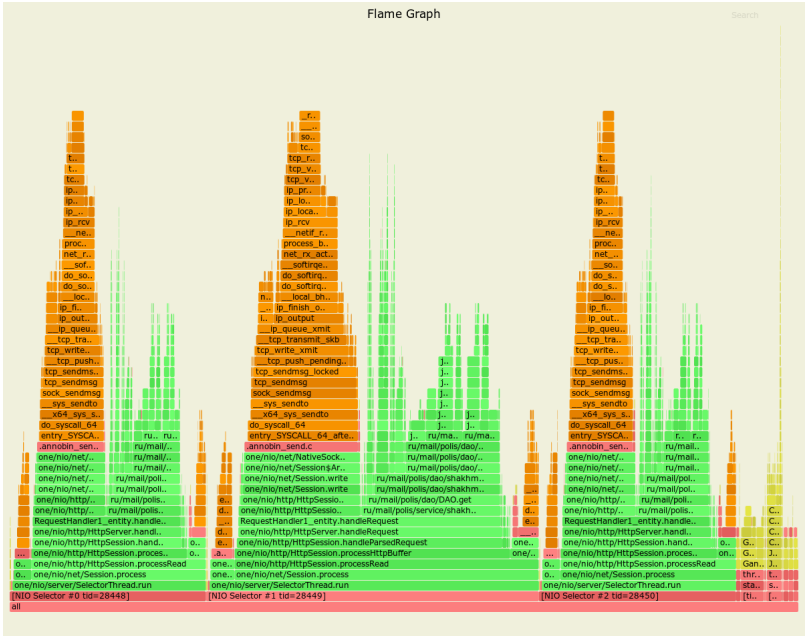

Рис. 8

Рассмотрим теперь результаты профилирования, когда сервер был нагружен запросами на удаление:

```
$ wrk -t1 -c1 -d60s -R10000 -s ./wrk/delete.lua --latency
```

На рис. 9 видно, что удаление данных занимает небольшое количество времени ($\approx 10\%$). Большую часть времени занял метод отправки данных по сети.

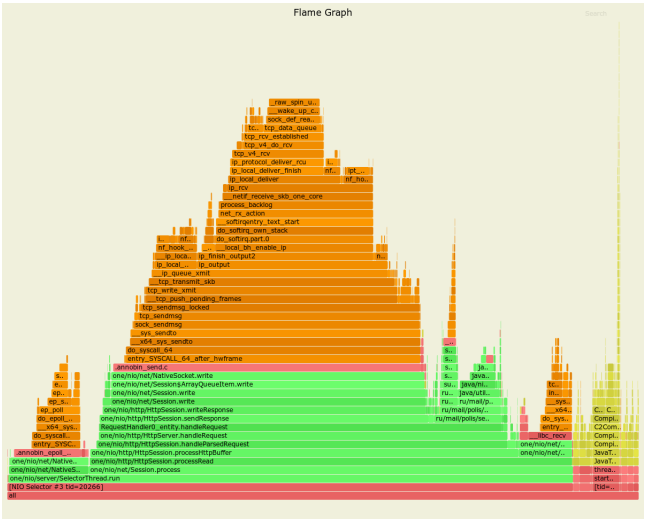

Рис. 9

Если заполнить хранилище до 100M и провести профилирование, то как и при малых данных запросы на запись (рис. 10) и на удаление (рис. 11) не сильно нагружают веб-сервер. При запросах на чтение большую часть времени также занимает метод *SSTable.iterator* (рис. 12), но увеличение числа потоков нашего веб-сервера помогает улучшить ситуацию (рис. 13).

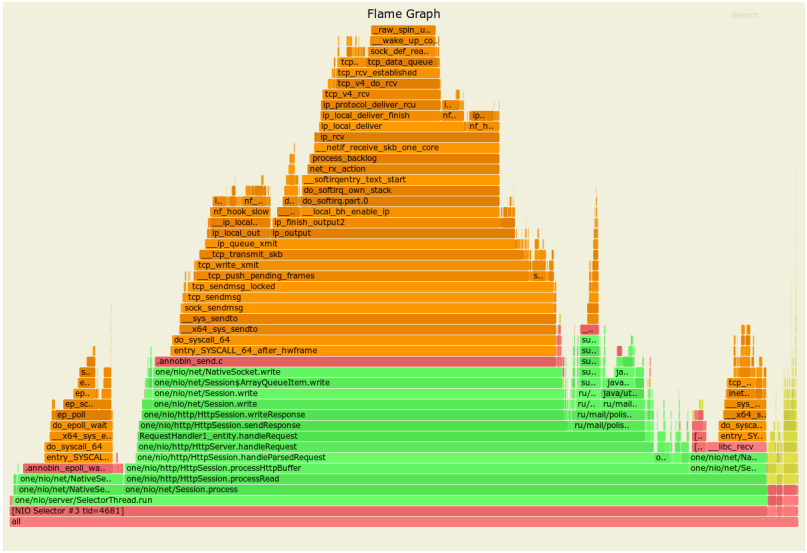

Рис. 10

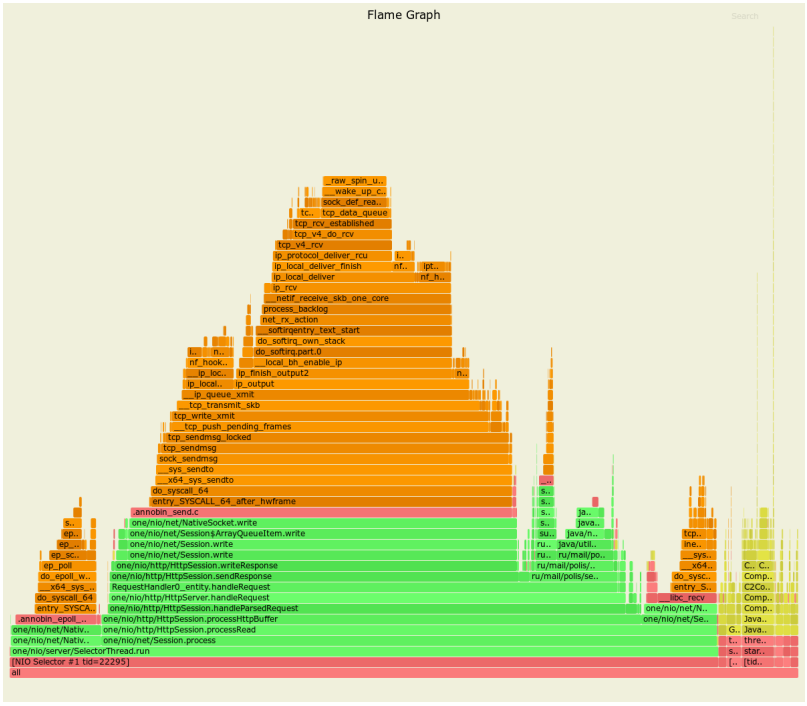

Рис. 11

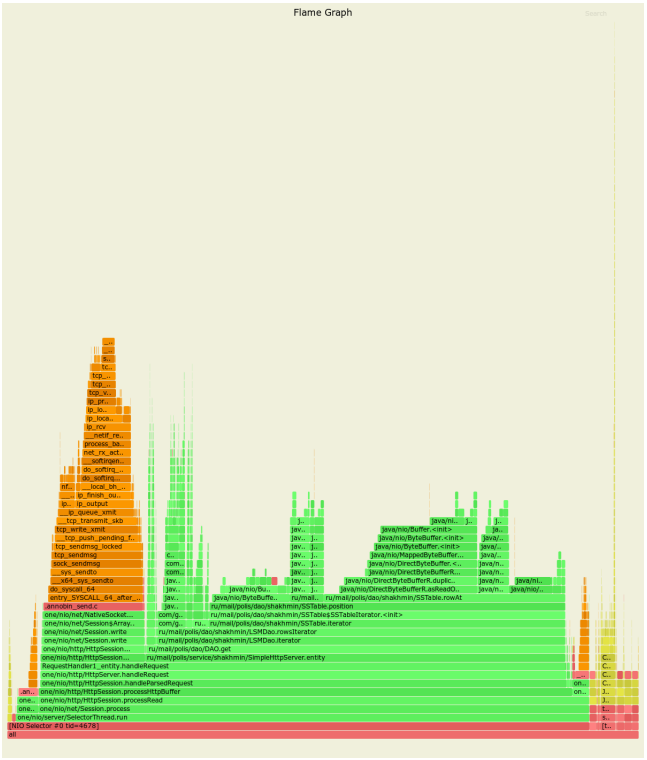
Flame Graph
Search
[NIO Selector #0 tid=4678]
all

Рис. 12


Flame Graph
Search
[NIO Selector #0 tid=29053]
[NIO Selector #1 tid=29054]
[NIO Selector #2 tid=29055]
[NIO Selector #3 tid=29056]
all

Рис. 13