Building Scalable Real-Time Internet Technology

Scaling to 12 Million Concurrent Connections: How MigratoryData Did It

Massive scalability is the biggest challenge we undertake at MigratoryData, a provider of an enterprise publish-subscribe messaging system for building very scalable real-time web and mobile applications. We recently published a blog post demonstrating 12 million concurrent connections with MigratoryData WebSocket Server running on a single 1U server. I am going to share some lessons learned while pushing the boundaries of scalability with MigratoryData WebSocket Server.

Objective

The goal of our benchmark is to **demonstrate the vertical scalability of the MigratoryData server using commodity hardware** usually found in data centers. The server used for the benchmark was a Dell PowerEdge R610 (see the specs in the initial blog post) running CentOS 6.4 out of the box — with no Linux kernel recompilation.

Opening 12 Million Sockets

Let's discuss the server side and the client side separately.

Server-Side Considerations

Server Port Numbers

A common misunderstanding is that a server cannot accept more than 65,536 (2^{16}) TCP sockets because TCP ports are 16-bit integer numbers.

First, the number of ports is limited to `65,536`, but this limitation applies only to a single IP address. Supposing that we are limited by the number of ports to have more than 65,536 clients, then adding more IP addresses to the server machine (either by adding new network cards, or simply by using IP aliasing for the existing network card) would solve the problem (even if, for opening 12 million client would need 184 network cards or IP aliases on the server machine).

In fact, the misunderstanding comes from the fact that the server does not use its listening IP address and a different ephemeral port for each new socket to distinguish among the sockets, but it uses the same listening IP address and the same listening port for all sockets and it distinguishes among sockets by using the IP address and the ephemeral port of each client. Therefore, **MigratoryData Server uses a single port to accept any number of clients** and optionally it uses another few ports for JMX monitoring, HTTP monitoring, etc

Server Socket Descriptors

While the MigratoryData server uses a single port to accept any number of clients, it uses a different socket descriptor for each client. So, **to open 12 million sockets, the process of the MigratoryData server should be able to use 12 million socket descriptors**. Increasing the maximum number of socket descriptors per process is possible using the command `ulimit`. Consequently, we increased this limit to about 20 million socket descriptors as follows:

```
ulimit -n 20000500
```

Because one cannot increase the maximum number of socket descriptors per process to a value larger than the current kernel maximum (`fs.nr_open`) and because the kernel maximum defaults to `1048576` (1024^2), prior to running the `ulimit` command, we increased the kernel maximum accordingly as follows:

```
echo 20000500 > /proc/sys/fs/nr_open
```

Follow

Client-Side Considerations

We developed a tool named **MigratoryData Client Benchmark** able to open a configurable number of connections to the MigratoryData server. MigratoryData Client Benchmark is also able to subscribe to a configurable number of subjects for each client connection (where each subject is randomly selected from a configurable set of subjects) and compute various statistics for the received messages.

We used ten servers Dell PowerEdge SC1435 having 16 GB RAM and 2 dual-core AMD Opteron CPU @2.0 GHz to run ten instances of MigratoryData Client Benchmark. Hence, from each client machine we opened 1.2 million sockets.

Client Port Numbers

Each socket connection uses a new ephemeral port number on the client machine. Therefore, we extended the range of the ephemeral ports from `500` to the maximum theoretical limit `65,536` – keeping only the ports from 1 to `500` reserved for the operating system:

```
sysctl -w net.ipv4.ip_local_port_range="500   65535"
```

Using this extension of the ephemeral port range, one can open up to `65,035` sockets from a client machine per each IP address of the client machine.

In order to be able to open `1.2` million sockets from a client machine, we had to create 19 IP aliases for the network interface of the client machine. In this way, about `65,000` sockets are opened to the MigratoryData server from each of the 19 IP addresses.

> ***Note*** *— For each socket, MigratoryData Client Benchmark uses precisely one of the 19 IP addresses and a successive port starting with the port `500` and ending with the port `65,535`, thus avoiding the random allocation of the ephemeral port numbers.*

Follow

Client Socket Descriptors

Each socket connection uses a socket descriptor on the client machine. Thus, to open 1.2 million sockets from a client machine, MigratoryData Client Benchmark should be able to use 1.2 million socket descriptors. As discussed in the section “Server Socket Descriptors” above, on each client machine we had to increase the maximum number of socket descriptors per process as follows:

```
echo 3000000 > /proc/sys/fs/nr_open
ulimit -n 2000000
```

Other Client Tuning Tips

To avoid delays between successive benchmark test rounds, we configured the kernel to reuse the sockets in state TIME_WAIT as follows:

```
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
```

Another problem difficult to debug was when the kernel maximum number of memory pages allocated to the TCP was reached. When, this happened the MigratoryData Client Benchmark remained up, but the client machine did not accept any new TCP connection. We figured out that we have to increase the maximum number of memory pages allocated to the TCP from 767586 (about 3 GB) to 2303190 (about 8.7 GB) as follows:

```
sysctl -w net.ipv4.tcp_mem="383865   511820   2303190"
```

Linux Kernel Tuning

A number of kernel tuning suggestions, mainly in terms of ports and socket descriptors, were already provided above. Now, we focus on kernel tuning suggestions for memory and performance optimization.

Follow

Kernel Memory Tuning

We used the Linux kernel version `3.9.4-1.el6.elrepo.x86_64` which consumed about 36 GB of kernel memory for the 12 million open sockets as shown by `slabtop` in one of the screenshots below.

> ***Note*** *— We started the benchmark tests using the default kernel 2.6.x of Centos 6.4. In version 2.6.x and other older kernel versions (until kernel version 3.7?), the kernel uses, besides the 36 GB memory for 12 million open sockets, another memory page (i.e.* `4096` *bytes) for each open socket. Even worse, it looks like there is a bug in the Linux kernel because this memory page used per open socket is not reported in* `/proc/meminfo` *which usually is not a big deal, but having lots of sockets one can observe a lot of memory vanishing in an inexplicable way. Therefore,* ***a recent Linux kernel such as*** `3.9` ***optimizes the kernel memory with about*** `46` ***GB for 12 million sockets.***

We used the following tuning related to the socket buffer sizes for our benchmark scenario:

```
sysctl -w net.ipv4.tcp_rmem="1024   4096   16384"
sysctl -w net.ipv4.tcp_wmem="1024   4096   16384"
sysctl -w net.ipv4.tcp_moderate_rcvbuf="0"
```

Balancing Hardware Interrupts

The server machine used a 10 Gbps network card Intel X520-DA2 having 24 tx/rx queues. We assigned each tx/rx queue to a different CPU core using `smp_affinity`. First, we identified the interrupts of our network interface named `p1p1` as follows:

```
[root@t15 migratorydata]# grep p1p1 /proc/interrupts
```

Follow

```
 77: 1684 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 78: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 79: 1089 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 80: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 81: 1080 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 82: 1076 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 83: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 84: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 85: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 86: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 87: 1105 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 88: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 89: 1073 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 90: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 91: 1073 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 92: 1073 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 93: 1075 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 94: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 95: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 96: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 97: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 98: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
 99: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
100: 1072 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1-
101:    2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge  p1p1
```

By default, the interrupts are assigned to the first CPU core. To balance the hardware interrupts of each tx/rx queue to a different CPU core, we used:

```
[root@t15 ~]# cat smp_affinity.sh
echo "1" >/proc/irq/77/smp_affinity
echo "2" >/proc/irq/78/smp_affinity
echo "4" >/proc/irq/79/smp_affinity
echo "8" >/proc/irq/80/smp_affinity
echo "10" >/proc/irq/81/smp_affinity
echo "20" >/proc/irq/82/smp_affinity
echo "40" >/proc/irq/83/smp_affinity
echo "80" >/proc/irq/84/smp_affinity
echo "100" >/proc/irq/85/smp_affinity
echo "200" >/proc/irq/86/smp_affinity
echo "400" >/proc/irq/87/smp_affinity
echo "800" >/proc/irq/88/smp_affinity
echo "1000" >/proc/irq/89/smp_affinity
echo "2000" >/proc/irq/90/smp_affinity
echo "4000" >/proc/irq/91/smp_affinity
echo "8000" >/proc/irq/92/smp_affinity
echo "10000" >/proc/irq/93/smp_affinity
echo "20000" >/proc/irq/94/smp_affinity
echo "40000" >/proc/irq/95/smp_affinity
echo "80000" >/proc/irq/96/smp_affinity
echo "100000" >/proc/irq/97/smp_affinity
echo "200000" >/proc/irq/98/smp_affinity
echo "400000" >/proc/irq/99/smp_affinity
echo "800000" >/proc/irq/100/smp_affinity
```

After running this `smp_affinity` script, one can see that the interrupts are now balanced on all 24 CPU cores:

```
[root@t15 ~]# grep p1p1 /proc/interrupts
 77:  3002 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1-T
 78:  1453 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 79:  1475 0 437 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 80:  1453 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 81:  1462 0 0 0 127 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 82:  1457 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 83:  1456 0 0 0 0 0 125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 84:  1453 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 85:  1453 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 86:  1453 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 87:  1497 0 0 0 0 0 0 0 0 0 130 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 88:  1453 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 89:  1454 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 90:  1453 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 91:  1454 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 92:  1454 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 93:  1456 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
 94:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 0 PCI-MSI-edge p1p1
 95:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 0 PCI-MSI-edge p1p1
 96:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 0 PCI-MSI-edge p1p1
```

Follow

```
 97:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 0 PCI-MSI-edge p1p1
 98:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 0 PCI-MSI-edge p1p1
 99:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 0 PCI-MSI-edge p1p1
100:  1453 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 123 PCI-MSI-edge p1p1
101:     2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 PCI-MSI-edge p1p1
```

Benchmark Configuration

Please refer to the sections "Hardware and Setup" and "The Benchmark Scenario" in the initial blog post for precise details about the benchmark scenario and setup used by the involved components: MigratoryData Server, MigratoryData Client Benchmark, and MigratoryData Publisher Benchmark. Here we detail their configurations.

MigratoryData Server

We used the following configuration of the MigratoryData server:

```
Memory = 55296 MB                       # Allocate 54 GB to the JVM
Listen = *:8800                         # Accept clients on port 8800
Monitor = JMX                           # Enabled JMX monitoring
MonitorJMX.Listen = 192.168.3.115:3000
MonitorJMX.Authentication = false
BufferLimit.Send = 800                  # Set the initial buffer sizes to 80
BufferLimit.Receive = 800               # that we publish messages of 512 by
PubSubTightlyCoupled = false            # Disable subscribe notifications to
CacheSize = 0                           # Disable the cache
BenchmarkMode = true                    # See details below
```

The parameter `BenchmarkMode` is set on `true` to handle the clients in a realistic way. In fact, in order to take advantage of multiprocessor servers, the incoming users are separated in parallel internal groups. Each client is assigned to a group based on its IP address. Because, in our benchmark a lot of clients come from the same IP address, we use the parameter `BenchmarkMode` to distribute them to different internal groups of the MigratoryData server, just as it would happen in production.

Follow

JVM Tuning

Tuning the JVM and especially the Garbage Collector related parameters is outside the scope of this blog post. We recommend the book Java Performance on JVM tuning. In a screenshot below we provide all the JVM parameters used in the benchmark.

The JVM parameter `UseCompressedOops` compresses the 64-bit pointers and offers non-negligible memory optimization. The recent versions of JVM enable this parameter by default for the JVM heap sizes smaller than about 30 GB. But `UseCompressedOops` can be used only for a JVM heap size of maximum 32 GB. The JVM parameter `ObjectAlignmentInBytes` extends the compression to the JVM heap sizes larger than 32 GB. **Using the parameters `ObjectAlignmentInBytes` and `UseCompressedOops` allowed us to benefit from pointer compression for the entire memory allocated to the MigratoryData server without the need to run two instances of the MigratoryData server** each one using less than 32 GB. Therefore, we included the following JVM parameters:

```
-XX:ObjectAlignmentInBytes=16 -XX:+UseCompressedOops
```

MigratoryData Client Benchmark

We used 10 instances of MigratoryData Client Benchmark running on 10 client machines Dell SC1435 as described above. Each instance of MigratoryData Client Benchmark opened 1.2 million connections to the MigratoryData server. Along each connection MigratoryData Client Benchmark subscribed to a different subject. Hence, we had 12 million concurrent clients subscribing to 12 million concurrent subjects.

For example, the 1st of the 10 instances of MigratoryData Client Benchmark used the following configuration:

```
Memory = 9000                       # Allocate 8.7 GB to the JVM
ServerAddresses = 192.168.3.115:8800  # The address of the MigratoryData ser
```

Follow

```
Clients = 1200000                     # Open 1.2 million subjects
Subjects = /p/s{1..1200000}/-         # Subscribe to /p/s1/-, /p/s2/-, ...,
SubjectsPerClient = 1                 # Each client subscribes to one subjec
LocalAddresses = 192.168.{10..30}.101 # Use these IP addresses for client so
BufferSize = 1024                     # Set the initial buffer size to 1024
OneToOne = true                       # client 1 subscribes to /p/s1,
                                      # client 2 subscribes to /p/s2,
                                      # ...
                                      # client 1200000 subscribes to /p/s120
```

In the same way:

- The 2nd instance of the MigratoryData Client Benchmark has been configured to open another 1.2 million concurrent connections and subscribed to the subjects from `/p/s1200001/-` to `/p/s2400000/-`
- The 3rd instance of the MigratoryData Client Benchmark has been configured to open another 1.2 million concurrent connections and subscribed to the subjects from `/p/s2400001/-` to `/p/s4800000/-`
- ...
- The 10th instance of the MigratoryData Client Benchmark has been configured to open another 1.2 million concurrent connections and subscribed to the subjects from `/p/s10800001/-` to `/p/s12000000/-`

Each instance of MigratoryData Client Benchmark displays every 5 seconds various information such as: the number of seconds since the start time, the minimum, maximum, mean, standard deviation for all messages received, the total number of messages received, and the frequency of the messages received during the last 5 seconds, as in the example below:

```
[2013-06-18, 15:03:11] [INFO] Uptime: 1,607 s
[2013-06-18, 15:03:11] [INFO] Latency min: 0 ms
[2013-06-18, 15:03:11] [INFO] Latency max: 2,024 ms
[2013-06-18, 15:03:11] [INFO] Latency mean: 269.68 ms
[2013-06-18, 15:03:11] [INFO] Latency standard deviation: 420.36 ms
[2013-06-18, 15:03:11] [INFO] 90% CI for the mean: [269.42 - 269.942] ms
[2013-06-18, 15:03:11] [INFO] 95% CI for the mean: [269.37 - 269.991] ms
[2013-06-18, 15:03:11] [INFO] Total messages: 7,089,699
```

Follow

```
[2013-06-18, 15:03:11] [INFO] Frequency: 20,905.80 messages/sec
```

The latency statistics are detailed in the section “Latency” below.

MigratoryData Publisher Benchmark

In our benchmark scenario, there are 200,000 messages per second published to the MigratoryData server. The payload of each message is a 512-byte string of random characters. We have developed a tool named **MigratoryData Publisher Benchmark** which is able to publish messages of a configurable size at a configurable frequency, for subjects selected randomly from a configurable set of subjects.

We ran 8 instances of MigratoryData Publisher Benchmark on 4 machines Dell SC1435 having the same specs as the client machines, two instances per machine. To achieve 200,000 messages per second, each instance published 25,000 messages per second.

For example, the 1st of the 8 instances of MigratoryData Publisher Benchmark used the following configuration:

```
Memory = 7700                        # Allocate 7.5 GB to the JVM
ServerAddresses = 192.168.3.115:8800 # The address of the MigratoryData ser
PublisherType = p                    # The first prefix of the subjects
Subjects = /s{1..1500000}/-          # Publish messages for /p/s1/-, /p/s2/
Frequency = 25000                    # Number of messages per second
MessageSize = 512                    # The number of bytes per message
```

In the same way:

- The 2nd instance of the MigratoryData Publisher Benchmark has been configured to publish 25,000 messages per second where the subject of each message is randomly selected from the subjects between `/p/s1500001/-` to `/p/s3000000/-`
- The 3rd instance of the MigratoryData Publisher Benchmark has

Follow

been configured to publish 25,000 messages per second where the subject of each message is randomly selected from the subjects between `/p/s3000001/-` to `/p/s4500000/-`
- ...
- The 8th instance of the MigratoryData Publisher Benchmark has been configured to publish 25,000 messages per second where the subject of each message is randomly selected from the subjects between `/p/s10500000/-` to `/p/s12000000/-`

Latency

Latency is defined here as the time needed for a message to propagate from the publisher to the client, via the MigratoryData server. In other words, the latency of a message is the difference between the time at which the message is sent by MigratoryData Publisher Benchmark to the MigratoryData server and the time at which the message is received by MigratoryData Client Benchmark from the MigratoryData server. All the client machines and publisher machines have the clocks synchronized using `ntp` (a few milliseconds difference might exist between the clocks, so you can observe a minimum latency of -1 millisecond due to such small clock differences).

We calculate the following latency statistics: maximum, mean, and standard deviation.

In order to calculate the latency statistics, one can use all latency values or use a reasonable sample size. The sampling approach provides statistically accurate results but it should be typically used when it is not possible to obtain all the values (e.g. election polling). Another disadvantage is that sampling cannot be used to compute the absolute maximum latency.

Because for certain systems, having a predictable latency – under a certain maximum – is more valuable than having low latency values in average, **we chosen to calculate the latency statistics for all values**, and so also obtaining the true maximum latency. While it might appear that computing the latency statistics for all values would add a certain performance overhead, in fact, one needs not to store the latency values, because **the statistics can be calculated on-the-fly for all messages** as follows:

Follow

```
numberOfMessages++;
maxLatency = maximum{maxLatency, newLatency}; /* max  */
deltaLatency = newLatency - meanLatency;
meanLatency = meanLatency + (deltaLatency / numberOfMessages); /* mean */
sumOfSquaredDelta += deltaLatency * deltaLatency;
stdevLatency = sqrt(sumOfSquaredDelta / numberOfMessages); /* stdev */
```

> ***Note*** *— Note that we have ten instances of MigratoryData Client Benchmark and each instance computes the latency statistics only for the messages it processes. However, the latency values of each instance of MigratoryData Client Benchmark represent an entire fraction of the population (10%), not only a typical sample. Also, as shown in the screenshot below, all statistics are practically identical for all ten instances of MigratoryData Client Benchmark.*

Bandwidth

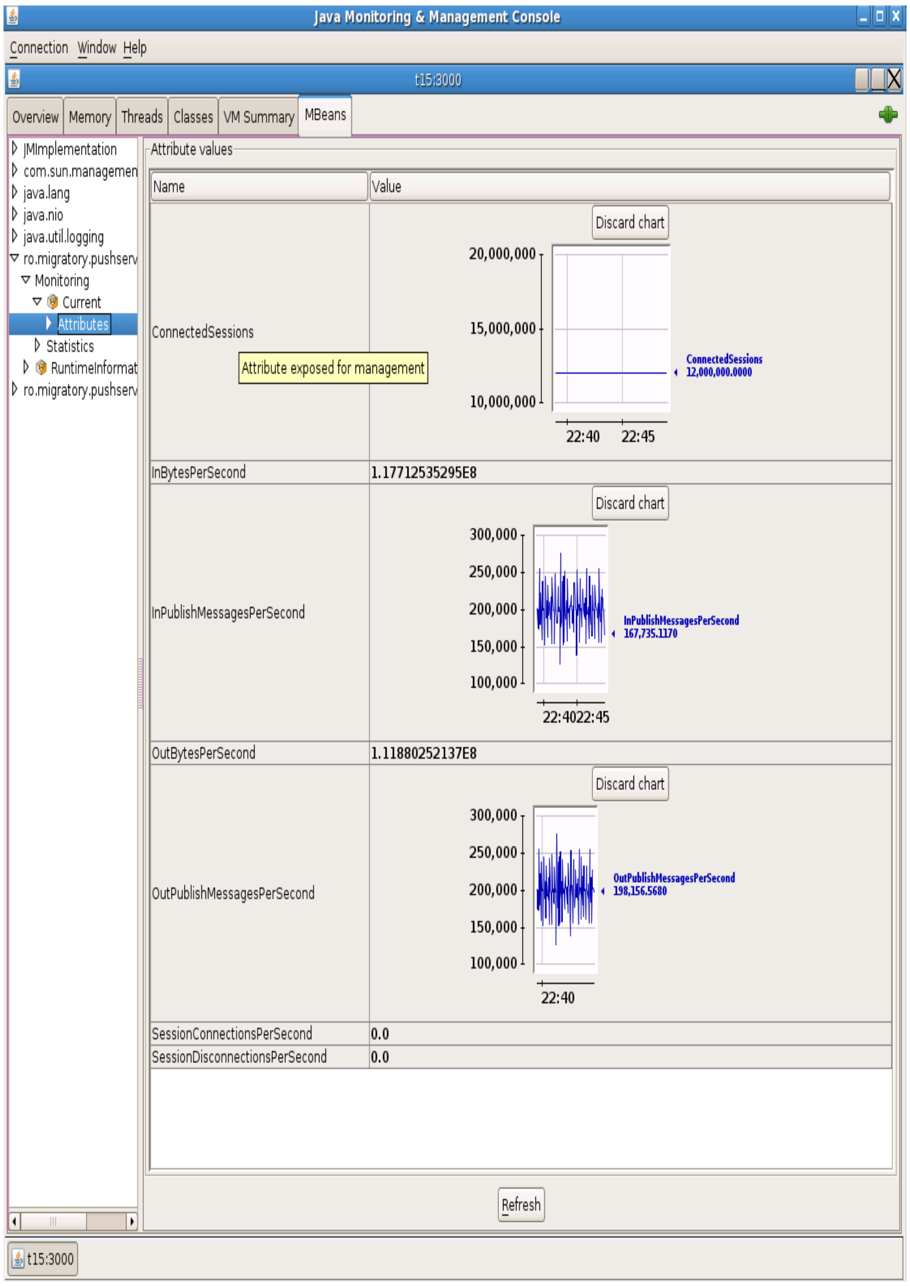
The Linux tools available for bandwidth calculation usually attempt to compute the bandwidth by capturing TCP packets. This approach does not scale for high data throughput and the bandwidth values produced are usually not accurate.

An accurate method for bandwidth calculation is to use the Receive Bytes / Packets and Transmit Bytes / Packets written by the Linux kernel to `/proc/net/dev` for the network interface used by the MigratoryData server. We compute the bandwidth by reading `/proc/net/dev` every 10 seconds. You can see the bandwidth transmitted by the MigratoryData server in one of the screenshots below.

Results

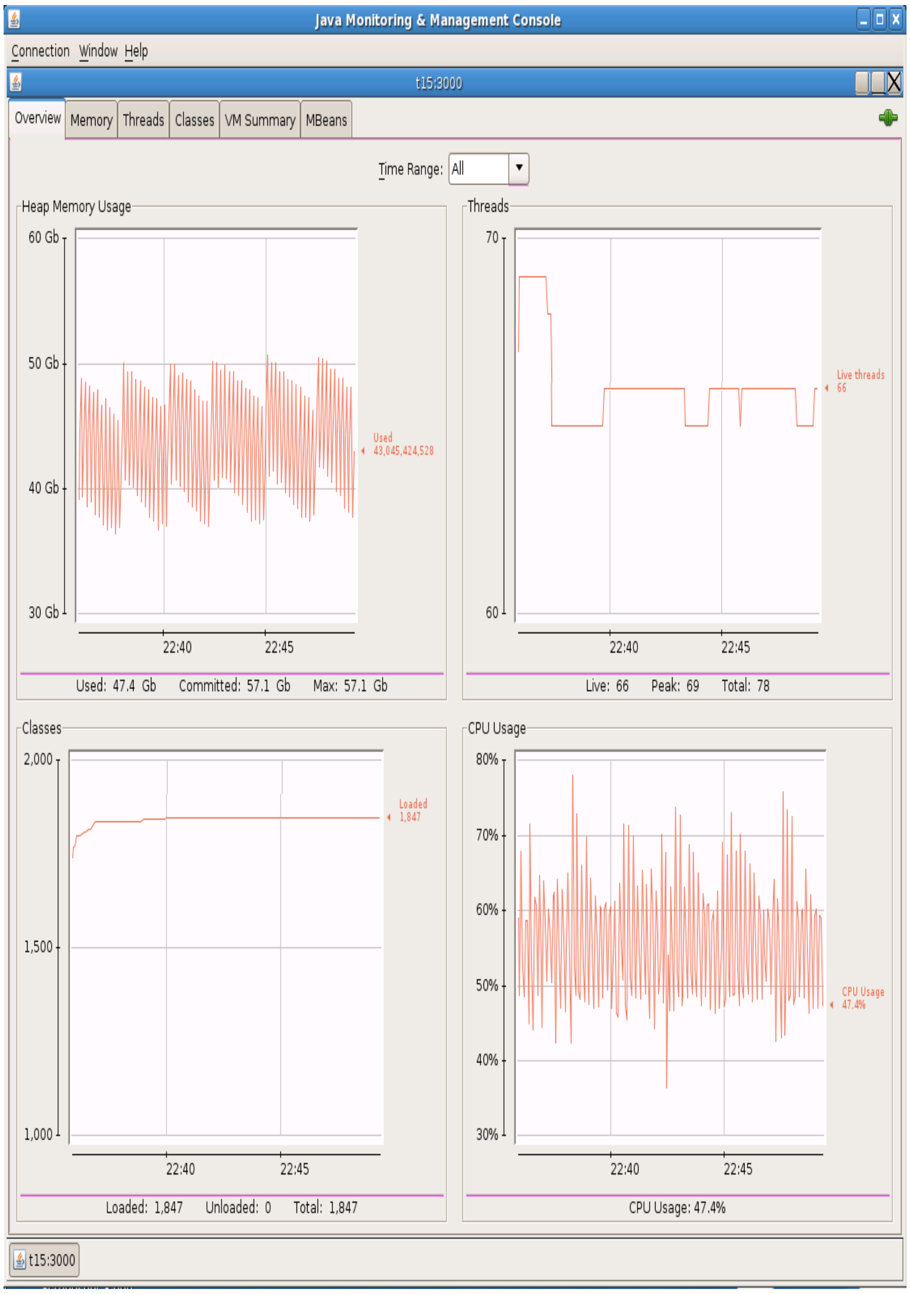
These are the screenshots with the results obtained which have been detailed above.

Follow



Name	Value
ConnectedSessions	
InBytesPerSecond	1.17712535295E8
InPublishMessagesPerSecond	
OutBytesPerSecond	1.11880252137E8
OutPublishMessagesPerSecond	
SessionConnectionsPerSecond	0.0
SessionDisconnectionsPerSecond	0.0

Follow



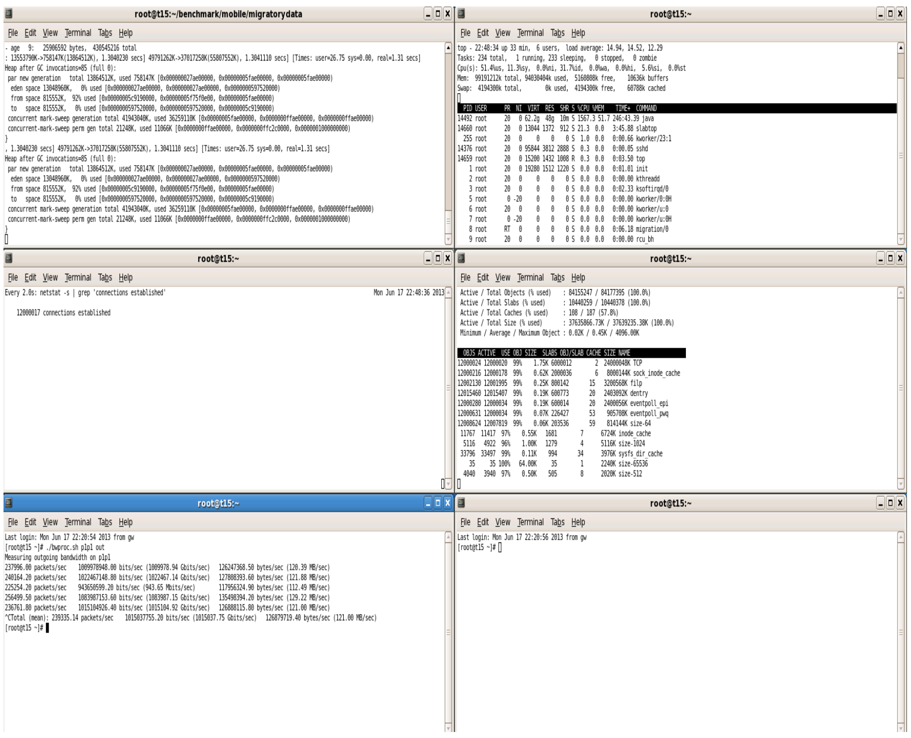
Used: 47.4 Gb Committed: 57.1 Gb Max: 57.1 Gb

Live: 66 Peak: 69 Total: 78

Loaded: 1,847 Unloaded: 0 Total: 1,847

CPU Usage: 47.4%

Follow


root@t15:~/benchmark/mobile/migratorydata
root@t15:~
File Edit View Terminal Tabs Help

```
- age   9:   25906592 bytes,  430545216 total
: 13553790K->758147K(13864512K), 1.3040230 secs] 49791262K->37017258K(55807552K), 1.3041110 secs] [Times: user=26.75 sys=0.00, real=1.31 secs]
Heap after GC invocations=85 (full 0):
 par new generation   total 13864512K, used 758147K [0x000000027ae00000, 0x00000005fae00000, 0x00000005fae00000)
  eden space 13048960K,   0% used [0x000000027ae00000, 0x000000027ae00000, 0x0000000597520000)
  from space 815552K,  92% used [0x00000005c9190000, 0x00000005f75f0e00, 0x00000005fae00000)
  to   space 815552K,   0% used [0x0000000597520000, 0x0000000597520000, 0x00000005c9190000)
 concurrent mark-sweep generation total 41943040K, used 36259110K [0x00000005fae00000, 0x0000000ffae00000, 0x0000000ffae00000)
 concurrent-mark-sweep perm gen total 21248K, used 11066K [0x0000000ffae00000, 0x0000000ffc2c0000, 0x0000001000000000)
}
, 1.3040230 secs] 49791262K->37017258K(55807552K), 1.3041110 secs] [Times: user=26.75 sys=0.00, real=1.31 secs]
Heap after GC invocations=85 (full 0):
 par new generation   total 13864512K, used 758147K [0x000000027ae00000, 0x00000005fae00000, 0x00000005fae00000)
  eden space 13048960K,   0% used [0x000000027ae00000, 0x000000027ae00000, 0x0000000597520000)
  from space 815552K,  92% used [0x00000005c9190000, 0x00000005f75f0e00, 0x00000005fae00000)
  to   space 815552K,   0% used [0x0000000597520000, 0x0000000597520000, 0x00000005c9190000)
 concurrent mark-sweep generation total 41943040K, used 36259110K [0x00000005fae00000, 0x0000000ffae00000, 0x0000000ffae00000)
 concurrent-mark-sweep perm gen total 21248K, used 11066K [0x0000000ffae00000, 0x0000000ffc2c0000, 0x0000001000000000)
}
```

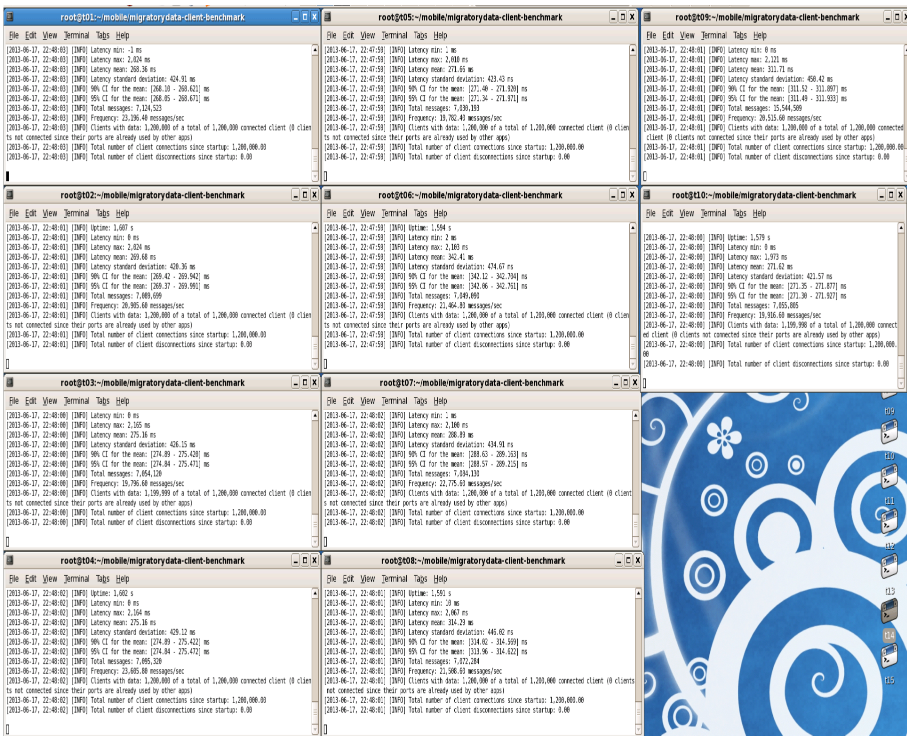
```
top - 22:48:34 up 33 min,  6 users,  load average: 14.94, 14.52, 12.29
Tasks: 234 total,   1 running, 233 sleeping,   0 stopped,   0 zombie
Cpu(s): 51.4%us, 11.3%sy,  0.0%ni, 31.7%id,  0.0%wa,  0.0%hi,  5.6%si,  0.0%st
Mem:  99191212k total, 94030404k used,  5160808k free,    10636k buffers
Swap:  4194300k total,        0k used,  4194300k free,    60788k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14492 root      20   0 62.2g  48g  10m S 1567.3 51.7 246:43.39 java
14660 root      20   0 13044 1372  912 S 21.3  0.0   3:45.88 slabtop
  255 root      20   0     0    0    0 S  1.0  0.0   0:00.66 kworker/23:1
14376 root      20   0 95844 3812 2888 S  0.3  0.0   0:00.05 sshd
14659 root      20   0 15200 1432 1008 R  0.3  0.0   0:03.50 top
    1 root      20   0 19280 1512 1220 S  0.0  0.0   0:01.01 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:02.33 ksoftirqd/0
    5 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/u:0
    7 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/u:0H
    8 root      RT   0     0    0    0 S  0.0  0.0   0:06.18 migration/0
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 rcu_bh
```

```
Every 2.0s: netstat -s | grep 'connections established'            Mon Jun 17 22:48:36 2013


    12000017 connections established
```

```
 Active / Total Objects (% used)    : 84155247 / 84177395 (100.0%)
 Active / Total Slabs (% used)      : 10440259 / 10440378 (100.0%)
 Active / Total Caches (% used)     : 108 / 187 (57.8%)
 Active / Total Size (% used)       : 37635866.73K / 37639235.38K (100.0%)
 Minimum / Average / Maximum Object : 0.02K / 0.45K / 4096.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
12000024 12000020  99%    1.75K 6000012        2  24000048K TCP
12000216 12000178  99%    0.62K 2000036        6   8000144K sock_inode_cache
12002130 12001995  99%    0.25K 800142        15   3200568K filp
12015460 12015407  99%    0.19K 600773        20   2403092K dentry
12000280 12000034  99%    0.19K 600014        20   2400056K eventpoll_epi
12000631 12000034  99%    0.07K 226427        53    905708K eventpoll_pwq
12008624 12007819  99%    0.06K 203536        59    814144K size-64
 11767  11417  97%    0.55K   1681        7      6724K inode_cache
  5116   4922  96%    1.00K   1279        4      5116K size-1024
 33796  33497  99%    0.11K    994       34      3976K sysfs_dir_cache
    35     35 100%   64.00K     35        1      2240K size-65536
  4040   3940  97%    0.50K    505        8      2020K size-512
```

```
Last login: Mon Jun 17 22:20:54 2013 from gw
[root@t15 ~]# ./bwproc.sh p1p1 out
Measuring outgoing bandwidth on p1p1
237996.00 packets/sec    1009978948.00 bits/sec (1009978.94 Gbits/sec)   126247368.50 bytes/sec (120.39 MB/sec)
240164.20 packets/sec    1022467148.80 bits/sec (1022467.14 Gbits/sec)   127808393.60 bytes/sec (121.88 MB/sec)
225254.20 packets/sec    943650599.20 bits/sec (943.65 Mbits/sec)        117956324.90 bytes/sec (112.49 MB/sec)
256499.50 packets/sec    1083987153.60 bits/sec (1083987.15 Gbits/sec)   135498394.20 bytes/sec (129.22 MB/sec)
236761.80 packets/sec    1015104926.40 bits/sec (1015104.92 Gbits/sec)   126888115.80 bytes/sec (121.00 MB/sec)
^CTotal (mean): 239335.14 packets/sec    1015037755.20 bits/sec (1015037.75 Gbits/sec)   126879719.40 bytes/sec (121.00 MB/sec)
[root@t15 ~]#
```

```
Last login: Mon Jun 17 22:20:56 2013 from gw
[root@t15 ~]#
```


root@t01:~/mobile/migratorydata-client-benchmark
root@t05:~/mobile/migratorydata-client-benchmark
root@t09:~/mobile/migratorydata-client-benchmark
root@t02:~/mobile/migratorydata-client-benchmark
root@t06:~/mobile/migratorydata-client-benchmark
root@t10:~/mobile/migratorydata-client-benchmark
root@t03:~/mobile/migratorydata-client-benchmark
root@t07:~/mobile/migratorydata-client-benchmark
root@t04:~/mobile/migratorydata-client-benchmark
root@t08:~/mobile/migratorydata-client-benchmark
File Edit View Terminal Tabs Help
t09
t10
t11
t12
t13
t14
t15

```
[2013-06-17, 22:48:03] [INFO] Latency min: -1 ms
[2013-06-17, 22:48:03] [INFO] Latency max: 2,024 ms
[2013-06-17, 22:48:03] [INFO] Latency mean: 268.36 ms
[2013-06-17, 22:48:03] [INFO] Latency standard deviation: 424.91 ms
[2013-06-17, 22:48:03] [INFO] 90% CI for the mean: [268.10 - 268.621] ms
[2013-06-17, 22:48:03] [INFO] 95% CI for the mean: [268.05 - 268.671] ms
[2013-06-17, 22:48:03] [INFO] Total messages: 7,124,523
[2013-06-17, 22:48:03] [INFO] Frequency: 23,196.40 messages/sec
[2013-06-17, 22:48:03] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 clien
ts not connected since their ports are already used by other apps)
[2013-06-17, 22:48:03] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:03] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:47:59] [INFO] Latency min: 1 ms
[2013-06-17, 22:47:59] [INFO] Latency max: 2,010 ms
[2013-06-17, 22:47:59] [INFO] Latency mean: 271.66 ms
[2013-06-17, 22:47:59] [INFO] Latency standard deviation: 423.43 ms
[2013-06-17, 22:47:59] [INFO] 90% CI for the mean: [271.40 - 271.920] ms
[2013-06-17, 22:47:59] [INFO] 95% CI for the mean: [271.34 - 271.971] ms
[2013-06-17, 22:47:59] [INFO] Total messages: 7,030,193
[2013-06-17, 22:47:59] [INFO] Frequency: 19,782.40 messages/sec
[2013-06-17, 22:47:59] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 clien
ts not connected since their ports are already used by other apps)
[2013-06-17, 22:47:59] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:47:59] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:48:01] [INFO] Latency min: 0 ms
[2013-06-17, 22:48:01] [INFO] Latency max: 2,121 ms
[2013-06-17, 22:48:01] [INFO] Latency mean: 311.71 ms
[2013-06-17, 22:48:01] [INFO] Latency standard deviation: 450.42 ms
[2013-06-17, 22:48:01] [INFO] 90% CI for the mean: [311.52 - 311.897] ms
[2013-06-17, 22:48:01] [INFO] 95% CI for the mean: [311.49 - 311.933] ms
[2013-06-17, 22:48:01] [INFO] Total messages: 15,544,509
[2013-06-17, 22:48:01] [INFO] Frequency: 20,515.60 messages/sec
[2013-06-17, 22:48:01] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected
 client (0 clients not connected since their ports are already used by other apps)
[2013-06-17, 22:48:01] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:01] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:48:01] [INFO] Uptime: 1,607 s
[2013-06-17, 22:48:01] [INFO] Latency min: 0 ms
[2013-06-17, 22:48:01] [INFO] Latency max: 2,024 ms
[2013-06-17, 22:48:01] [INFO] Latency mean: 269.68 ms
[2013-06-17, 22:48:01] [INFO] Latency standard deviation: 420.36 ms
[2013-06-17, 22:48:01] [INFO] 90% CI for the mean: [269.42 - 269.942] ms
[2013-06-17, 22:48:01] [INFO] 95% CI for the mean: [269.37 - 269.991] ms
[2013-06-17, 22:48:01] [INFO] Total messages: 7,089,699
[2013-06-17, 22:48:01] [INFO] Frequency: 20,905.60 messages/sec
[2013-06-17, 22:48:01] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 clien
ts not connected since their ports are already used by other apps)
[2013-06-17, 22:48:01] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:01] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:47:59] [INFO] Uptime: 1,594 s
[2013-06-17, 22:47:59] [INFO] Latency min: 2 ms
[2013-06-17, 22:47:59] [INFO] Latency max: 2,103 ms
[2013-06-17, 22:47:59] [INFO] Latency mean: 342.41 ms
[2013-06-17, 22:47:59] [INFO] Latency standard deviation: 474.67 ms
[2013-06-17, 22:47:59] [INFO] 90% CI for the mean: [342.12 - 342.704] ms
[2013-06-17, 22:47:59] [INFO] 95% CI for the mean: [342.06 - 342.761] ms
[2013-06-17, 22:47:59] [INFO] Total messages: 7,049,090
[2013-06-17, 22:47:59] [INFO] Frequency: 21,464.80 messages/sec
[2013-06-17, 22:47:59] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 clien
ts not connected since their ports are already used by other apps)
[2013-06-17, 22:47:59] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:47:59] [INFO] Total number of client disconnections since startup: 0.00
```

```

[2013-06-17, 22:48:00] [INFO] Uptime: 1,579 s
[2013-06-17, 22:48:00] [INFO] Latency min: 0 ms
[2013-06-17, 22:48:00] [INFO] Latency max: 1,973 ms
[2013-06-17, 22:48:00] [INFO] Latency mean: 271.62 ms
[2013-06-17, 22:48:00] [INFO] Latency standard deviation: 421.57 ms
[2013-06-17, 22:48:00] [INFO] 90% CI for the mean: [271.35 - 271.877] ms
[2013-06-17, 22:48:00] [INFO] 95% CI for the mean: [271.30 - 271.927] ms
[2013-06-17, 22:48:00] [INFO] Total messages: 7,055,805
[2013-06-17, 22:48:00] [INFO] Frequency: 19,916.60 messages/sec
[2013-06-17, 22:48:00] [INFO] Clients with data: 1,199,998 of a total of 1,200,000 connect
ed client (0 clients not connected since their ports are already used by other apps)
[2013-06-17, 22:48:00] [INFO] Total number of client connections since startup: 1,200,000.
00
[2013-06-17, 22:48:00] [INFO] Total number of client disconnections since startup: 0.00
```

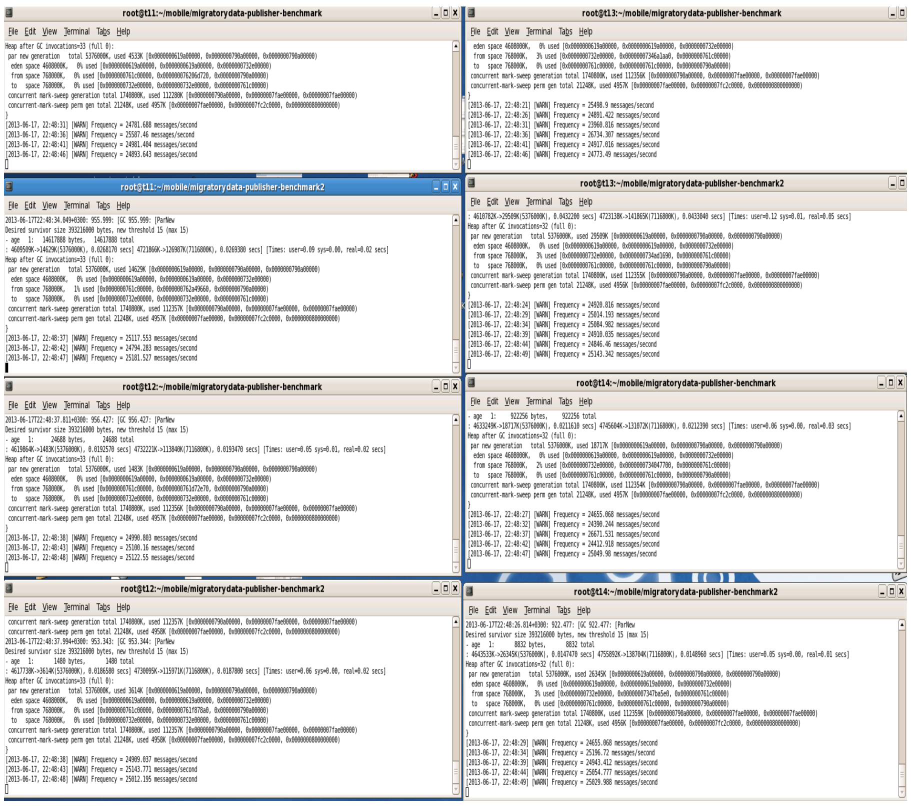
```
[2013-06-17, 22:48:00] [INFO] Latency min: 0 ms
[2013-06-17, 22:48:00] [INFO] Latency max: 2,165 ms
[2013-06-17, 22:48:00] [INFO] Latency mean: 275.16 ms
[2013-06-17, 22:48:00] [INFO] Latency standard deviation: 426.15 ms
[2013-06-17, 22:48:00] [INFO] 90% CI for the mean: [274.89 - 275.420] ms
[2013-06-17, 22:48:00] [INFO] 95% CI for the mean: [274.84 - 275.471] ms
[2013-06-17, 22:48:00] [INFO] Total messages: 7,054,120
[2013-06-17, 22:48:00] [INFO] Frequency: 19,796.60 messages/sec
[2013-06-17, 22:48:00] [INFO] Clients with data: 1,199,999 of a total of 1,200,000 connected client (0 clien
ts not connected since their ports are already used by other apps)
[2013-06-17, 22:48:00] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:00] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:48:02] [INFO] Latency min: 1 ms
[2013-06-17, 22:48:02] [INFO] Latency max: 2,100 ms
[2013-06-17, 22:48:02] [INFO] Latency mean: 288.89 ms
[2013-06-17, 22:48:02] [INFO] Latency standard deviation: 434.91 ms
[2013-06-17, 22:48:02] [INFO] 90% CI for the mean: [288.63 - 289.163] ms
[2013-06-17, 22:48:02] [INFO] 95% CI for the mean: [288.57 - 289.215] ms
[2013-06-17, 22:48:02] [INFO] Total messages: 7,084,130
[2013-06-17, 22:48:02] [INFO] Frequency: 22,775.60 messages/sec
[2013-06-17, 22:48:02] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 client
s not connected since their ports are already used by other apps)
[2013-06-17, 22:48:02] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:02] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:48:02] [INFO] Uptime: 1,602 s
[2013-06-17, 22:48:02] [INFO] Latency min: 0 ms
[2013-06-17, 22:48:02] [INFO] Latency max: 2,164 ms
[2013-06-17, 22:48:02] [INFO] Latency mean: 275.16 ms
[2013-06-17, 22:48:02] [INFO] Latency standard deviation: 429.12 ms
[2013-06-17, 22:48:02] [INFO] 90% CI for the mean: [274.89 - 275.422] ms
[2013-06-17, 22:48:02] [INFO] 95% CI for the mean: [274.84 - 275.472] ms
[2013-06-17, 22:48:02] [INFO] Total messages: 7,095,320
[2013-06-17, 22:48:02] [INFO] Frequency: 23,605.80 messages/sec
[2013-06-17, 22:48:02] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 clien
ts not connected since their ports are already used by other apps)
[2013-06-17, 22:48:02] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:02] [INFO] Total number of client disconnections since startup: 0.00
```

```
[2013-06-17, 22:48:01] [INFO] Uptime: 1,591 s
[2013-06-17, 22:48:01] [INFO] Latency min: 10 ms
[2013-06-17, 22:48:01] [INFO] Latency max: 2,067 ms
[2013-06-17, 22:48:01] [INFO] Latency mean: 314.29 ms
[2013-06-17, 22:48:01] [INFO] Latency standard deviation: 446.02 ms
[2013-06-17, 22:48:01] [INFO] 90% CI for the mean: [314.02 - 314.569] ms
[2013-06-17, 22:48:01] [INFO] 95% CI for the mean: [313.96 - 314.622] ms
[2013-06-17, 22:48:01] [INFO] Total messages: 7,072,284
[2013-06-17, 22:48:01] [INFO] Frequency: 21,508.60 messages/sec
[2013-06-17, 22:48:01] [INFO] Clients with data: 1,200,000 of a total of 1,200,000 connected client (0 clients
 not connected since their ports are already used by other apps)
[2013-06-17, 22:48:01] [INFO] Total number of client connections since startup: 1,200,000.00
[2013-06-17, 22:48:01] [INFO] Total number of client disconnections since startup: 0.00
```

Follow


root@t11:~/mobile/migratorydata-publisher-benchmark
File Edit View Terminal Tabs Help

```
Heap after GC invocations=33 (full 0):
 par new generation   total 5376000K, used 4533K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   0% used [0x0000000761c00000, 0x000000076206d720, 0x0000000790a00000)
  to   space 768000K,   0% used [0x0000000732e00000, 0x0000000732e00000, 0x0000000761c00000)
 concurrent mark-sweep generation total 1740800K, used 112280K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4957K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:31] [WARN] Frequency = 24781.688 messages/second
[2013-06-17, 22:48:36] [WARN] Frequency = 25587.46 messages/second
[2013-06-17, 22:48:41] [WARN] Frequency = 24981.404 messages/second
[2013-06-17, 22:48:46] [WARN] Frequency = 24893.643 messages/second
```

root@t13:~/mobile/migratorydata-publisher-benchmark
File Edit View Terminal Tabs Help

```
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   3% used [0x0000000732e00000, 0x00000007346a1aa0, 0x0000000761c00000)
  to   space 768000K,   0% used [0x0000000761c00000, 0x0000000761c00000, 0x0000000790a00000)
 concurrent mark-sweep generation total 1740800K, used 112356K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4957K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:21] [WARN] Frequency = 25498.9 messages/second
[2013-06-17, 22:48:26] [WARN] Frequency = 24891.422 messages/second
[2013-06-17, 22:48:31] [WARN] Frequency = 23960.816 messages/second
[2013-06-17, 22:48:36] [WARN] Frequency = 26734.307 messages/second
[2013-06-17, 22:48:41] [WARN] Frequency = 24917.016 messages/second
[2013-06-17, 22:48:46] [WARN] Frequency = 24773.49 messages/second
```

root@t11:~/mobile/migratorydata-publisher-benchmark2
File Edit View Terminal Tabs Help

```
2013-06-17T22:48:34.049+0300: 955.999: [GC 955.999: [ParNew
Desired survivor size 393216000 bytes, new threshold 15 (max 15)
- age   1:   14617888 bytes,   14617888 total
: 4609509K->14629K(5376000K), 0.0268170 secs] 4721866K->126987K(7116800K), 0.0269380 secs] [Times: user=0.09 sys=0.00, real=0.02 secs]
Heap after GC invocations=33 (full 0):
 par new generation   total 5376000K, used 14629K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   1% used [0x0000000761c00000, 0x0000000762a49660, 0x0000000790a00000)
  to   space 768000K,   0% used [0x0000000732e00000, 0x0000000732e00000, 0x0000000761c00000)
 concurrent mark-sweep generation total 1740800K, used 112357K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4957K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:37] [WARN] Frequency = 25117.553 messages/second
[2013-06-17, 22:48:42] [WARN] Frequency = 24794.283 messages/second
[2013-06-17, 22:48:47] [WARN] Frequency = 25181.527 messages/second
```

root@t13:~/mobile/migratorydata-publisher-benchmark2
File Edit View Terminal Tabs Help

```
: 4610782K->29509K(5376000K), 0.0432200 secs] 4723138K->141865K(7116800K), 0.0433040 secs] [Times: user=0.12 sys=0.01, real=0.05 secs]
Heap after GC invocations=32 (full 0):
 par new generation   total 5376000K, used 29509K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   3% used [0x0000000732e00000, 0x0000000734ad1690, 0x0000000761c00000)
  to   space 768000K,   0% used [0x0000000761c00000, 0x0000000761c00000, 0x0000000790a00000)
 concurrent mark-sweep generation total 1740800K, used 112355K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4956K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:24] [WARN] Frequency = 24920.816 messages/second
[2013-06-17, 22:48:29] [WARN] Frequency = 25014.193 messages/second
[2013-06-17, 22:48:34] [WARN] Frequency = 25084.982 messages/second
[2013-06-17, 22:48:39] [WARN] Frequency = 24910.035 messages/second
[2013-06-17, 22:48:44] [WARN] Frequency = 24846.46 messages/second
[2013-06-17, 22:48:49] [WARN] Frequency = 25143.342 messages/second
```

root@t12:~/mobile/migratorydata-publisher-benchmark
File Edit View Terminal Tabs Help

```
2013-06-17T22:48:37.811+0300: 956.427: [GC 956.427: [ParNew
Desired survivor size 393216000 bytes, new threshold 15 (max 15)
- age   1:      24688 bytes,      24688 total
: 4619864K->1483K(5376000K), 0.0192570 secs] 4732221K->113840K(7116800K), 0.0193470 secs] [Times: user=0.05 sys=0.01, real=0.02 secs]
Heap after GC invocations=33 (full 0):
 par new generation   total 5376000K, used 1483K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   0% used [0x0000000761c00000, 0x0000000761d72e70, 0x0000000790a00000)
  to   space 768000K,   0% used [0x0000000732e00000, 0x0000000732e00000, 0x0000000761c00000)
 concurrent mark-sweep generation total 1740800K, used 112356K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4957K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:38] [WARN] Frequency = 24990.803 messages/second
[2013-06-17, 22:48:43] [WARN] Frequency = 25100.16 messages/second
[2013-06-17, 22:48:48] [WARN] Frequency = 25122.55 messages/second
```

root@t14:~/mobile/migratorydata-publisher-benchmark
File Edit View Terminal Tabs Help

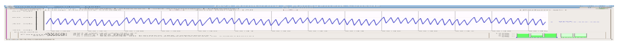
```
- age   1:     922256 bytes,     922256 total
: 4633249K->18717K(5376000K), 0.0211610 secs] 4745604K->131072K(7116800K), 0.0212390 secs] [Times: user=0.06 sys=0.00, real=0.03 secs]
Heap after GC invocations=32 (full 0):
 par new generation   total 5376000K, used 18717K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   2% used [0x0000000732e00000, 0x0000000734047700, 0x0000000761c00000)
  to   space 768000K,   0% used [0x0000000761c00000, 0x0000000761c00000, 0x0000000790a00000)
 concurrent mark-sweep generation total 1740800K, used 112354K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4957K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:27] [WARN] Frequency = 24655.068 messages/second
[2013-06-17, 22:48:32] [WARN] Frequency = 24390.244 messages/second
[2013-06-17, 22:48:37] [WARN] Frequency = 26671.531 messages/second
[2013-06-17, 22:48:42] [WARN] Frequency = 24412.918 messages/second
[2013-06-17, 22:48:47] [WARN] Frequency = 25049.98 messages/second
```

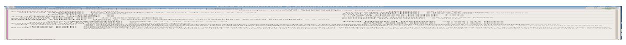
root@t12:~/mobile/migratorydata-publisher-benchmark2
File Edit View Terminal Tabs Help

```
 concurrent mark-sweep generation total 1740800K, used 112357K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4958K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
2013-06-17T22:48:37.994+0300: 953.343: [GC 953.344: [ParNew
Desired survivor size 393216000 bytes, new threshold 15 (max 15)
- age   1:       1480 bytes,       1480 total
: 4617738K->3614K(5376000K), 0.0186580 secs] 4730095K->115971K(7116800K), 0.0187800 secs] [Times: user=0.06 sys=0.00, real=0.02 secs]
Heap after GC invocations=33 (full 0):
 par new generation   total 5376000K, used 3614K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   0% used [0x0000000761c00000, 0x0000000761f878a0, 0x0000000790a00000)
  to   space 768000K,   0% used [0x0000000732e00000, 0x0000000732e00000, 0x0000000761c00000)
 concurrent mark-sweep generation total 1740800K, used 112357K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4958K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:38] [WARN] Frequency = 24909.037 messages/second
[2013-06-17, 22:48:43] [WARN] Frequency = 25143.771 messages/second
[2013-06-17, 22:48:48] [WARN] Frequency = 25012.195 messages/second
```

root@t14:~/mobile/migratorydata-publisher-benchmark2
File Edit View Terminal Tabs Help

```
2013-06-17T22:48:26.814+0300: 922.477: [GC 922.477: [ParNew
Desired survivor size 393216000 bytes, new threshold 15 (max 15)
- age   1:       8832 bytes,       8832 total
: 4643533K->26345K(5376000K), 0.0147470 secs] 4755892K->138704K(7116800K), 0.0148960 secs] [Times: user=0.05 sys=0.00, real=0.01 secs]
Heap after GC invocations=32 (full 0):
 par new generation   total 5376000K, used 26345K [0x0000000619a00000, 0x0000000790a00000, 0x0000000790a00000)
  eden space 4608000K,   0% used [0x0000000619a00000, 0x0000000619a00000, 0x0000000732e00000)
  from space 768000K,   3% used [0x0000000732e00000, 0x00000007347ba5e0, 0x0000000761c00000)
  to   space 768000K,   0% used [0x0000000761c00000, 0x0000000761c00000, 0x0000000790a00000)
 concurrent mark-sweep generation total 1740800K, used 112359K [0x0000000790a00000, 0x00000007fae00000, 0x00000007fae00000)
 concurrent-mark-sweep perm gen total 21248K, used 4956K [0x00000007fae00000, 0x00000007fc2c0000, 0x0000000800000000)
}
[2013-06-17, 22:48:29] [WARN] Frequency = 24655.068 messages/second
[2013-06-17, 22:48:34] [WARN] Frequency = 25196.72 messages/second
[2013-06-17, 22:48:39] [WARN] Frequency = 24943.412 messages/second
[2013-06-17, 22:48:44] [WARN] Frequency = 25054.777 messages/second
[2013-06-17, 22:48:49] [WARN] Frequency = 25029.988 messages/second
```





Conclusion

In this post we presented the obstacles faced when scaling to 12 million concurrent connections in terms of kernel tuning, memory tuning, tools, and benchmarking methodology. More insights into how MigratoryData WebSocket Server itself is architectured to achieve this massive vertical scalability might be the subject of a new post.

This entry was posted in Benchmarks, C10M, High Scalability, migratory-data, Push Notifications, WebSockets on October 10, 2013 [https://mro-taru.wordpress.com/2013/10/10/scaling-to-12-million-concurrent-connec-tions-how-migratorydata-did-it/] .

---

8 thoughts on "Scaling to 12 Million Concurrent Connections: How

Follow

MigratoryData Did It"

sailoren
December 28, 2013 at 12:31 pm

great job,,could you provide benchmark client for us to download?and show our more detail about gc config?

Mihai Rotaru Post author
December 28, 2013 at 1:53 pm

Thanks for your interest for the MigratoryData solution.

The GC details are available in the last screenshot above (see "VM arguments" line).

The evaluation package of the MigratoryData solution is freely available at :

http://migratorydata.com/migratorydata-downloads.html

In order to download it, please register using your corporate information and accept the trial agreement.

Then, please send your request from your corporate/professional email (to support [at] migatorydata [dot] com) for MigratoryData Benchmark Tools and an evaluation licence for your benchmark.

Alejandro Varela
June 3, 2015 at 12:11 pm

this is awesome!

Follow

could you recommend any book on the subject? thank!

Suresh Khemka
June 18, 2015 at 8:45 am

Mihai,
So did you only tuned rmem and wmem buffer on linux side or tuned any other parameters also to enable linux to scale to so many connections? Didn't single TCP server socket became a bottleneck for you?

Ben Chiu
July 8, 2015 at 10:42 am

Dear Sir,

I get problem when using mutiple IP address for clients to create more than 64K connections. The number of outbound connections stucks at about 67500. I post this problem at stackoverflow (http://stackoverflow.com/questions/31290255/increasing-max-outbound-connections-on-centos).
Could you help on this? Thanks.

Mihai Rotaru Post author
July 8, 2015 at 1:22 pm

Hi Ben,

I just provided an answer to your stackoverflow question available at:

http://stackoverflow.com/q/31290255

Follow

Thanks,
Mihai

Follow

pradyumna
July 23, 2015 at 5:55 pm

Hi Mihai,

I am trying to get 85000 concurrent http connections. It would be great if you could help me out, I posted a question on stack overflow:

http://stackoverflow.com/questions/31576282/need-to-increase-the-number-of-concurrent-http-connections-to-85000/31576448#31576448

I have done all the tuning required on my client and server but the number won't increase. Could you share the script you used to curl or wget the connections?

Thanks,
Pradyumna

Pingback: C10K no more, C10M already here! | 笑点低

Follow