四种框架分别实现百万 websocket 常连接的服务器

2016-08-01 数据库开发

> 来源：鸟窝
> 链接：colobu.com/2015/05/22/implement-C1000K-servers-by-spray-netty-undertow-and-node-js/

著名的 C10K 问题提出的时候, 正是 2001 年。这篇文章可以说是高性能服务器开发的一个标志性文档，它讨论的就是单机为1万个连接提供服务这个问题，当时因为硬件和软件的限制，单机1万还是一个非常值得挑战的目标。但是时光荏苒，随着硬件和软件的飞速发展，单机1万的目标已经变成了最简单不过的事情。

现在用任何一种主流语言都能提供单机1万的并发处理的能力。所以现在目标早已提高了100倍，变成C1000k，也就是一台服务器为100万连接提供服务。在2010年,2011年已经看到一些实现C1000K的文章了，所以在2015年，实现C1000K应该不是一件困难的事情。

本文是我在实践过程中的记录，我的目标是使用spran-websocket，netty, undertow和node.js四种框架分别实现C1000K的服务器，看看这几个框架实现的难以程度，性能如何。开发语言为Scala和Javascript。

当然，谈起性能，我们还必须谈到每秒每个连接有多少个请求，也就是RPS数，还要考虑每条消息的大小。
一般来说，我们会选取一个百分比，比如每秒20%的连接会收发消息。我的需求是服务器只是push,客户端不会主动发送消息。 一般每一分钟会为这一百万群发一条消息。

所以实现的测试工具每个client建立60000个websocket连接，一共二十个client。实际不可能使用20台机器，我使用了两台AWS C3.2xlarge(8核16G)服务器作为客户端机。每台机器10个客户端。
服务器每1分钟群发一条消息。消息内容很简单，只是服务器的当天时间。

最近看到360用Go实现的消息推送系统，下面是他们的数据：

> 目前360消息推送系统服务于50+内部产品，万款开发平台App，实时长连接数亿量级，日独数十亿量级，1分钟内可以实现亿量级广播，日下发峰值百亿量级，400台物理机，3000多个实例分布在9个独立集群中，每个集群跨国内外近10个IDC。

四个服务器的代码和Client测试工具代码可以在github上下载。（其实不止四种框架了，现在包括Netty, Undertow, Jetty, Spray-websocket, Vert.x, Grizzly 和 Node.js 七种框架的实现）

测试下来可以看到每种服务器都能轻松达到同时120万的websocket活动连接，只是资源占用和事务处理时间有差别。120万只是保守数据，在这么多连接情况下服务器依然很轻松，下一步我会进行C2000K的测试。

在测试之前我们需要对服务器/客户机的一些参数进行调优。

服务器的参数调优

一般会修改两个文件，/etc/sysctl.conf和/etc/security/limits.conf， 用来配置TCP/IP参数和最大文件描述符。

TCP/IP参数配置

修改文件/etc/sysctl.conf,配置网络参数。

```
net.ipv4.tcp_wmem = 4096 87380 4161536
net.ipv4.tcp_rmem = 4096 87380 4161536
net.ipv4.tcp_mem = 786432 2097152 3145728
```

数值根据需求进行调整。更多的参数可以看以前整理的一篇文章: Linux TCP/IP 协议栈调优。

执行/sbin/sysctl -p即时生效。

最大文件描述符

Linux内核本身有文件描述符最大值的限制，你可以根据需要更改：

- 系统最大打开文件描述符数：/proc/sys/fs/file-max
  1. 临时性设置：echo 1000000 > /proc/sys/fs/file-max
  2. 永久设置：修改/etc/sysctl.conf文件，增加fs.file-max = 1000000
- 进程最大打开文件描述符数

  使用ulimit -n查看当前设置。使用ulimit -n 1000000进行临时性设置。

  要想永久生效，你可以修改/etc/security/limits.conf文件，增加下面的行：

```
* hard nofile 1000000
* soft nofile 1000000
root hard nofile 1000000
root soft nofile 1000000
```

还有一点要注意的就是hard limit不能大于/proc/sys/fs/nr_open，因此有时你也需要修改nr_open的值。

执行echo 2000000 > /proc/sys/fs/nr_open

查看当前系统使用的打开文件描述符数，可以使用下面的命令：

```
[root@localhost ~]# cat /proc/sys/fs/file-nr
1632 0 1513506
```

其中第一个数表示当前系统已分配使用的打开文件描述符数，第二个数为分配后已释放的（目前已不再使用），第三个数等于file-max。

总结一下：

- 所有进程打开的文件描述符数不能超过/proc/sys/fs/file-max
- 单个进程打开的文件描述符数不能超过user limit中nofile的soft limit
- nofile的soft limit不能超过其hard limit

- nofile的hard limit不能超过/proc/sys/fs/nr_open

应用运行时调优

1. Java 应用内存调优

   服务器使用12G内存，吞吐率优先的垃圾回收器：

2. V8引擎

```
node --nouse-idle-notification --expose-gc --max-new-space-size=1024 --max-new-space-size=2048 --max-old-space-size=8192 ./webserver.js
```

OutOfMemory Killer

如果服务器本身内存不大，比如8G，在不到100万连接的情况下，你的服务器进程有可能出现”Killed”的问题。 运行dmesg可以看到

```
Out of memory: Kill process 10375 (java) score 59 or sacrifice child
```

这是Linux的OOM Killer主动杀死的。 开启oom-killer的话，在/proc/pid下对每个进程都会多出3个与oom打分调节相关的文件。临时对某个进程可以忽略oom-killer可以使用下面的方式:

echo -17 > /proc/$(pidof java)/oom_adj

解决办法有多种，可以参看文章最后的参考文章,最好是换一个内存更大的机器。

客户端的参数调优

在一台系统上，连接到一个远程服务时的本地端口是有限的。根据TCP/IP协议，由于端口是16位整数，也就只能是0到 65535，而0到1023是预留端口，所以能分配的端口只是1024到65534，也就是64511个。也就是说，一台机器一个IP只能创建六万多个长连接。

要想达到更多的客户端连接，可以用更多的机器或者网卡，也可以使用虚拟IP来实现,比如下面的命令增加了19个IP地址，其中一个给服务器用，其它18个给client,这样

可以产生18 * 60000 = 1080000个连接。

```
ifconfig eth0:0 192.168.77.10 netmask 255.255.255.0 up
ifconfig eth0:1 192.168.77.11 netmask 255.255.255.0 up
ifconfig eth0:2 192.168.77.12 netmask 255.255.255.0 up
ifconfig eth0:3 192.168.77.13 netmask 255.255.255.0 up
ifconfig eth0:4 192.168.77.14 netmask 255.255.255.0 up
ifconfig eth0:5 192.168.77.15 netmask 255.255.255.0 up
ifconfig eth0:6 192.168.77.16 netmask 255.255.255.0 up
ifconfig eth0:7 192.168.77.17 netmask 255.255.255.0 up
```

```
ifconfig eth0:8 192.168.77.18 netmask 255.255.255.0 up
ifconfig eth0:9 192.168.77.19 netmask 255.255.255.0 up
ifconfig eth0:10 192.168.77.20 netmask 255.255.255.0 up
ifconfig eth0:11 192.168.77.21 netmask 255.255.255.0 up
ifconfig eth0:12 192.168.77.22 netmask 255.255.255.0 up
ifconfig eth0:13 192.168.77.23 netmask 255.255.255.0 up
ifconfig eth0:14 192.168.77.24 netmask 255.255.255.0 up
ifconfig eth0:15 192.168.77.25 netmask 255.255.255.0 up
ifconfig eth0:16 192.168.77.26 netmask 255.255.255.0 up
ifconfig eth0:17 192.168.77.27 netmask 255.255.255.0 up
ifconfig eth0:18 192.168.77.28 netmask 255.255.255.0 up
```

修改/etc/sysctl.conf文件：

```
net.ipv4.ip_local_port_range = 1024 65535
```

执行/sbin/sysctl -p即时生效。

服务器测试

实际测试中我使用一台AWS C3.4xlarge (16 cores, 32G memory)作为应用服务器，两台AWS C3.2xlarge (8 cores, 16G memory)服务器作为客户端。

这两台机器作为测试客户端绰绰有余，每台客户端机器创建了十个内网虚拟IP, 每个IP创建60000个websocket连接。

客户端配置如下：

/etc/sysctl.conf配置

```
fs.file-max = 2000000
fs.nr_open = 2000000
net.ipv4.ip_local_port_range = 1024 65535
```

/etc/security/limits.conf配置

```
* soft nofile 2000000
* hard nofile 2000000

* soft nproc 2000000
* hard nproc 2000000
```

服务端配置如下：

/etc/sysctl.conf配置

```
fs.file-max = 2000000
fs.nr_open = 2000000
```

```
net.ipv4.ip_local_port_range = 1024 65535
```

/etc/security/limits.conf配置

```
* soft nofile 2000000
* hard nofile 2000000

* soft nproc 2000000
* hard nproc 2000000
```

Netty服务器

- 建立120万个连接，不发送消息，轻轻松松达到。内存还剩14G未用。

```
[roocolobu ~]# ss -s; free -m
Total: 1200231 (kernel 1200245)
TCP: 1200006 (estab 1200002, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 4

Transport Total IP IPv6
* 1200245 - -
RAW 0 0 0
UDP 1 1 0
TCP 1200006 1200006 0
INET 1200007 1200007 0
FRAG 0 0 0

total used free shared buffers cached
Mem: 30074 15432 14641 0 9 254
-/+ buffers/cache: 15167 14906
Swap: 815 0 815
```

每分钟给所有的120万个websocket发送一条消息，消息内容为当前的服务器的时间。这里发送显示是单线程发送，服务器发送完120万个总用时15秒左右。

```
02:15:43.307 [pool-1-thread-1] INFO com.colobu.webtest.netty.WebServer$ - send msg to channels for c4453a26-bca6-42b6-b29b-43653767f9fc
02:15:57.190 [pool-1-thread-1] INFO com.colobu.webtest.netty.WebServer$ - sent 1200000 channels for c4453a26-bca6-42b6-b29b-43653767f9fc
```

发送时CPU使用率并不高，网络带宽占用基本在10M左右。

```
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read writ| recv send| in out | int csw
0 0 100 0 0 0| 0 0 | 60B 540B| 0 0 | 224 440
0 0 100 0 0 0| 0 0 | 60B 870B| 0 0 | 192 382
0 0 100 0 0 0| 0 0 | 59k 74k| 0 0 |2306 2166
2 7 87 0 0 4| 0 0 |4998k 6134k| 0 0 | 169k 140k
1 7 87 0 0 5| 0 0 |4996k 6132k| 0 0 | 174k 140k
1 7 87 0 0 5| 0 0 |4972k 6102k| 0 0 | 176k 140k
1 7 87 0 0 5| 0 0 |5095k 6253k| 0 0 | 178k 142k
```

```
2 7 87 0 0 5| 0 0 |5238k 6428k| 0 0 | 179k 144k
1 7 87 0 0 5| 0 24k|4611k 5660k| 0 0 | 166k 129k
1 7 87 0 0 5| 0 0 |5083k 6238k| 0 0 | 175k 142k
1 7 87 0 0 5| 0 0 |5277k 6477k| 0 0 | 179k 146k
1 7 87 0 0 5| 0 0 |5297k 6500k| 0 0 | 179k 146k
1 7 87 0 0 5| 0 0 |5383k 6607k| 0 0 | 180k 148k
1 7 87 0 0 5| 0 0 |5504k 6756k| 0 0 | 184k 152k
1 7 87 0 0 5| 0 48k|5584k 6854k| 0 0 | 183k 152k
1 7 87 0 0 5| 0 0 |5585k 6855k| 0 0 | 183k 153k
1 7 87 0 0 5| 0 0 |5589k 6859k| 0 0 | 184k 153k
1 5 91 0 0 3| 0 0 |4073k 4999k| 0 0 | 135k 110k
0 0 100 0 0 0| 0 32k| 60B 390B| 0 0 |4822 424
```

客户端(一共20个，这里选取其中一个查看它的指标)。每个客户端保持6万个连接。每个消息从服务器发送到客户端接收到总用时平均633毫秒，而且标准差很小，每个连接用时差不多。

```
Active WebSockets for eb810c24-8565-43ea-bc27-9a0b2c910ca4
count = 60000
WebSocket Errors for eb810c24-8565-43ea-bc27-9a0b2c910ca4
count = 0

-- Histograms ------------------------------------------------------------------
Message latency for eb810c24-8565-43ea-bc27-9a0b2c910ca4
count = 693831
min = 627
max = 735
mean = 633.06
stddev = 9.61
median = 631.00
75% 633.00
95% 640.00
98% 651.00
99% 670.00
99.9% 735.00

-- Meters ----------------------------------------------------------------------
Message Rate for eb810c24-8565-43ea-bc27-9a0b2c910ca4
count = 693832
mean rate = 32991.37 events/minute
1-minute rate = 60309.26 events/minute
5-minute rate = 53523.45 events/minute
15-minute rate = 31926.26 events/minute
```

平均每个client的RPS = 1000, 总的RPS大约为 20000 requests /seconds.
latency平均值为633 ms，最长735 ms，最短627ms。

Spray服务器

- 建立120万个连接，不发送消息，轻轻松松达到。它的内存相对较高，内存还剩7G。

```
# ss -s; free -m
Total: 1200234 (kernel 1200251)
TCP: 1200006 (estab 1200002, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 4

Transport Total IP IPv6
* 1200251 - -
RAW 0 0 0
UDP 1 1 0
TCP 1200006 1200006 0
INET 1200007 1200007 0
FRAG 0 0 0

             total       used       free     shared    buffers     cached
Mem:         30074      22371       7703          0         10        259
-/+ buffers/cache:      22100       7973
Swap:          815          0        815
```

每分钟给所有的120万个websocket发送一条消息，消息内容为当前的服务器的时间。

CPU使用较高，发送很快，带宽可以达到46M。群发完一次大约需要8秒左右。

```
05/22 04:42:57.569 INFO [ool-2-worker-15] c.c.w.s.WebServer - send msg to workers 。for 8454e7d8-b8ca-4881-912b-6cdf3e6787bf
05/22 04:43:05.279 INFO [ool-2-worker-15] c.c.w.s.WebServer - sent msg to workers for 8454e7d8-b8ca-4881-912b-6cdf3e6787bf. current workers: 1200000
```

```
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
74   9  14   0   0   3|   0   24k|6330k   20M|   0     0 |  20k 1696
70  23   0   0   0   6|   0   64k|  11M   58M|   0     0 |  18k 2526
75  11   6   0   0   7|   0     0 |9362k   66M|   0     0 |  24k   11k
82   4   8   0   0   6|   0     0 |  11M   35M|   0     0 |  24k   10k
85   0  14   0   0   1|   0     0 |8334k   12M|   0     0 |  44k  415
84   0  15   0   0   1|   0     0 |9109k   16M|   0     0 |  36k  425
81   0  19   0   0   0|   0   24k| 919k  858k|   0     0 |  23k  629
76   0  23   0   0   0|   0     0 | 151k  185k|   0     0 |  18k 1075
```

客户端(一共20个，这里选取其中一个查看它的指标)。每个客户端保持6万个连接。每个消息从服务器发送到客户端接收到总用时平均1412毫秒，而且标准差较大，每个连接用时差别较大。

```
Active WebSockets for 6674c9d8-24c6-4e77-9fc0-58afabe7436f
count = 60000
WebSocket Errors for 6674c9d8-24c6-4e77-9fc0-58afabe7436f
count = 0

-- Histograms ------------------------------------------------------------
Message latency for 6674c9d8-24c6-4e77-9fc0-58afabe7436f
count = 454157
```

```
min = 716
max = 9297
mean = 1412.77
stddev = 1102.64
median = 991.00
75% 1449.00
95% 4136.00
98% 4951.00
99% 5308.00
99.9% 8854.00

-- Meters ----------------------------------------------------------------------
Message Rate for 6674c9d8-24c6-4e77-9fc0-58afabe7436f
count = 454244
mean rate = 18821.51 events/minute
1-minute rate = 67705.18 events/minute
5-minute rate = 49917.79 events/minute
15-minute rate = 24355.57 events/minute
```

Undertow

- 建立120万个连接，不发送消息，轻轻松松达到。内存占用较少，还剩余11G内存。

```
# ss -s; free -m
Total: 1200234 (kernel 1200240)
TCP: 1200006 (estab 1200002, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 4

Transport Total IP IPv6
* 1200240 - -
RAW 0 0 0
UDP 1 1 0
TCP 1200006 1200006 0
INET 1200007 1200007 0
FRAG 0 0 0

total used free shared buffers cached
Mem: 30074 18497 11576 0 10 286
-/+ buffers/cache: 18200 11873
Swap: 815 0 815
```

每分钟给所有的120万个websocket发送一条消息，消息内容为当前的服务器的时间。

群发玩一次大约需要15秒。

```
03:19:31.154 [pool-1-thread-1] INFO c.colobu.webtest.undertow.WebServer$ - send msg to channels for d9b450da-2631-42bc-a802-44285f63a62d
03:19:46.755 [pool-1-thread-1] INFO c.colobu.webtest.undertow.WebServer$ - sent 1200000 channels for d9b450da-2631-42bc-a802-44285f63a62d
```

客户端(一共20个，这里选取其中一个查看它的指标)。每个客户端保持6万个连接。每个消息从服务器发送到客户端

接收到总用时平均672毫秒，而且标准差较小，每个连接用时差别不大。

```
Active WebSockets for b2e95e8d-b17a-4cfa-94d5-e70832034d4d
count = 60000
WebSocket Errors for b2e95e8d-b17a-4cfa-94d5-e70832034d4d
count = 0

-- Histograms ------------------------------------------------------------
Message latency for b2e95e8d-b17a-4cfa-94d5-e70832034d4d
count = 460800
min = 667
max = 781
mean = 672.12
stddev = 5.90
median = 671.00
75%
95%
98%
99%
99.9%

-- Meters ----------------------------------------------------------------------
Message Rate for b2e95e8d-b17a-4cfa-94d5-e70832034d4d
count = 460813
mean rate = 27065.85 events/minute
1-minute rate = 69271.67 events/minute
5-minute rate = 48641.78 events/minute
15-minute rate = 24128.67 events/minute
Setup Rate for b2e95e8d-b17a-4cfa-94d5-e70832034d4d
```

node.js

node.js不是我要考虑的框架，列在这里只是作为参考。性能也不错。

```
Active WebSockets for 537c7f0d-e58b-4996-b29e-098fe2682dcf
count = 60000
WebSocket Errors for 537c7f0d-e58b-4996-b29e-098fe2682dcf
count = 0

-- Histograms ------------------------------------------------------------
Message latency for 537c7f0d-e58b-4996-b29e-098fe2682dcf
count = 180000
min = 808
max = 847
mean = 812.10
stddev = 1.95
median = 812.00
75% 812.00
```

```
95% 813.00
98% 814.00
99% 815.00
99.9% 847.00

-- Meters ----------------------------------------------------------------------
Message Rate for 537c7f0d-e58b-4996-b29e-098fe2682dcf
count = 180000
mean rate = 7191.98 events/minute
1-minute rate = 10372.33 events/minute
5-minute rate = 16425.78 events/minute
15-minute rate = 9080.53 events/minute
```