

Nginx Performance Evaluation

Steps:

1. Purpose
2. Environment
3. Setup
4. Environment Check
5. Compile Nginx
6. Tune system
7. Run ab test
8. Monitor
9. Analyze
10. Introduce jemalloc
11. Redo ab test
12. Analyze for jemalloc
13. Conclusion
14. Future works
15. DPDK

Detailed information in every step:

1. Purpose

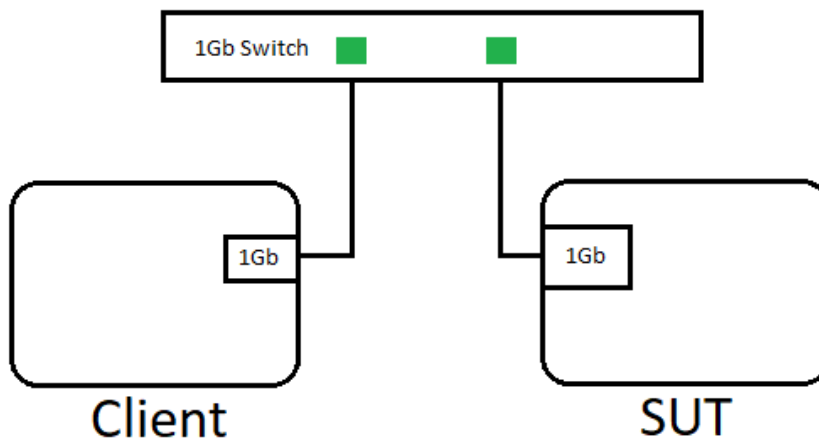
This is a performance test on latest version of plain nginx, find bottleneck, and solve it.

A simple approach is to saturate a single core in my SUT workstation, because my 1Gb NIC can only generate very limited workload

But it will show you how to easily scale-up if you have 10Gb NIC + 10-core CPU, or more easily scale-out if you have 10x servers with single 10Gb NIC + 10-core CPU.

And the CPU saturation will help you to see the bottleneck directly, not just guessing, or trying randomly.

2. Environment



SUT Hardware:

NIC: Intel Corporation Ethernet Connection (7) I219-LM (No RSS)

CPU: Intel(R) Xeon(R) E-2186G CPU @ 3.80GHz (HT=off)

Memory: 32GB x2 (2666MHz)

Software:

OS: Red Hat Enterprise Linux Server release 7.5 (Maipo)

Kernel: 3.10.0-862.11.6.el7.x86_64 (with Meltdown & Spectre patches)

nginx-1.15.5 + zlib-1.2.11 + pcre-8.42 + jemalloc-5.1.0

3. Setup

Download the latest version of Nginx from <http://nginx.org/download/nginx-1.15.5.tar.gz>

Compilation requires zlib-1.2.11 + pcre-8.42 (jemalloc-5.1.0)

4. Environment Check

NIC bandwidth:

```
[root@dr1 ~]# iperf3 -c www.example.com &  
[ ID] Interval      Transfer   Bandwidth   Retr Cwnd  
[  4]  0.00-1.00  sec  114 MBytes  958 Mb/s    0   543 Kbytes  
looks good
```

Stop SUT Services:

systemctl stop ***, after doing this step, we can check system remaining services:

```
[root@st50 www]# systemctl -a |grep running  
session-113.scope          loaded active running Session 113 of user root  
session-2.scope           loaded active running Session 2 of user root  
auditd.service            loaded active running Security Auditing Service  
dbus.service              loaded active running D-Bus System Message Bus  
getty@tty1.service         loaded active running Getty on tty1  
polkit.service             loaded active running Authorization Manager  
sshd.service               loaded active running OpenSSH server daemon  
dbus.socket                loaded active running D-Bus System Message Bus Socket
```

Enable Client irqbalance service:

```
[root@dr1 www]# systemctl start irqbalance
```

5. Compile Nginx

```
[root@st50 nginx-1.15.5]# ./configure --prefix=/home/www/nginx0 --with-  
pcre=/home/www/pcre-8.42 --with-zlib=/home/www/zlib-1.2.11 --with-debug
```

6. Tuning system

check irq

```
[root@st50 www]# cat /proc/interrupts |grep eno1
123: 27700      1      0      2      0      0      0      0      0      0      6      0      0 IR-
PCI-MSI-edge    eno1
```

bind irq to single logical core #2:

```
echo 4 > /proc/irq/123/smp_affinity
```

slightly enlarge the ring cache :

```
ethtool -G eno1 rx 8192
```

reduce softirq a little:

```
ethtool -C eno1 adaptive-tx off adaptive-rx off rx-usecs 400 # rx-frames 15 # in microseconds
or packets
```

Disable THP:

```
echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

```
echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

add following 2 lines into /etc/security/limits.conf

```
* hard nofile 655350
```

```
* soft nofile 655350
```

Change kernel parameters listed below:

fs.file-max=500000	#open files
kernel.sysrq = 0	#sysrq keys disabled
kernel.core_uses_pid = 1	#coredump
kernel.msgmnb = 65536	#max bytes
kernel.msgmax = 65536	#max length
kernel.shmmax = 68719476736	#single share memory segment max size
kernel.shmall = 4294967296	#pages
net.core.wmem_default = 8388608	#tx window
net.core.rmem_default = 8388608	#rx window
net.core.wmem_max = 16777216	#tx window
net.core.rmem_max = 16777216	#rx window
net.core.netdev_max_backlog = 40960	#rx queue len
net.core.somaxconn = 40960	#connection
#net.core.default_qdisc=fq	#google congestion control
#net.ipv4.tcp_congestion_control=bbr	#google congestion control
net.ipv4.ip_forward = 0	#disable ip forward
net.ipv4.conf.default.rp_filter = 1	#reverse path filter, same port io
net.ipv4.tcp_syncookies = 1	#avoid syn flood
net.ipv4.tcp_max_tw_buckets = 6000	#TIME_WAIT #
net.ipv4.tcp_sack = 1	#selective acknowledge
net.ipv4.tcp_window_scaling = 1	#64k window

```

net.ipv4.tcp_rmem = 4096      87380 4194304      #rx window: min/def/max
net.ipv4.tcp_wmem = 4096      16384 4194304      #tx window: min/def/max
net.ipv4.tcp_mem = 94500000 915000000 927000000    #sys tcp mem
net.ipv4.tcp_max_orphans = 3276800                #sockets
net.ipv4.tcp_max_syn_backlog = 40960               #syn queue
net.ipv4.tcp_timestamps = 0                        #better than resend
net.ipv4.tcp_synack_retries = 1                    #hand shake#
net.ipv4.tcp_syn_retries = 1                       #
net.ipv4.tcp_tw_recycle = 1                        #
net.ipv4.tcp_tw_reuse = 1                          # TIME-WAIT sockets reuse
net.ipv4.tcp_fin_timeout = 1                       #close timeout
net.ipv4.tcp_keepalive_time = 30                    #default 2h
net.ipv4.tcp_slow_start_after_idle=0                #
net.ipv4.ip_local_port_range = 1024 65000
vm.zone_reclaim_mode=0                             #alloc remote page when used up local
kernel.kptr_restrict=0                             #perf

```

Change nginx.conf:

```

user www;
worker_processes 10;
#worker_cpu_affinity 000000000100 000000000100;
error_log /dev/null ;

```

```

events {
    use epoll;
    worker_connections 4096;
}

```

```

http {
    include mime.types;
    default_type application/octet-stream;
    #open_file_cache      max=10 inactive=5m;
    #open_file_cache_valid 2m;
    #open_file_cache_min_uses 1;
    #access_log logs/access.log main;
    access_log off;
    server_names_hash_bucket_size 128;
    client_header_buffer_size 2k;
    large_client_header_buffers 4 4k;
    client_max_body_size 8m;

```

```

    sendfile    on;                #skip user space
    tcp_nopush  on;                #merge bundle
    tcp_nodelay on;                #disable nagle
    keepalive_timeout 60;

```

```

gzip on;
#gzip_static on;
gzip_proxied expired no-cache no-store private auth;
gzip_min_length 1k;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_comp_level 4;
gzip_types text/plain application/x-javascript text/css application/xml image/svg+xml;
gzip_vary on;

server {
    listen 80;
    server_name www.example.com;
    #access_log off;
    location / {
        root html;
        index index.html index.htm;
    }
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root html;
    }
}

```

7. Run test

start nginx on the core #2:

```
numactl -C 2 --localalloc nginx/sbin/nginx
```

Validate logo.svg

```
wget http://www.example.com/logo.svg
```

run ab test:

```

ab -n 800000 -c 100 http://www.example.com/logo.svg
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

```

Benchmarking www.example.com (be patient)

```

Completed 80000 requests
Completed 160000 requests
Completed 240000 requests
Completed 320000 requests
Completed 400000 requests

```

Completed 480000 requests
Completed 560000 requests
Completed 640000 requests
Completed 720000 requests
Completed 800000 requests
Finished 800000 requests

Server Software: nginx/1.15.5
Server Hostname: www.example.com
Server Port: 80

Document Path: /logo.svg
Document Length: 2649 bytes

Concurrency Level: 100
Time taken for tests: 23.820 seconds
Complete requests: 800000
Failed requests: 0
Write errors: 0
Total transferred: 2328000000 bytes
HTML transferred: 2119200000 bytes
Requests per second: 33585.56 [#/sec] (mean)
Time per request: 2.977 [ms] (mean)
Time per request: 0.030 [ms] (mean, across all concurrent requests)
Transfer rate: 95443.35 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	1 0.3	1	6
Processing:	0	2 0.5	2	22
Waiting:	0	2 0.5	1	22
Total:	1	3 0.7	3	25

Percentage of the requests served within a certain time (ms)

50%	3
66%	3
75%	3
80%	3
90%	4
95%	4
98%	5
99%	5
100%	25 (longest request)

8. Monitor

Network: sar -n DEV 2 4

CPU: mpstat -P ALL 2 4

Memory: numactl -H

Cache/TLB: perf stat --cpu=2 -dd

Syscall utility:

perf probe 'tcp_recvmsg'

perf probe -x /lib64/libc.so.6 malloc

perf record -e probe:tcp_recvmsg -e probe_libc:malloc -a

Hotspot capture: perf record ; perf top ; perf stat

Memory access utility: perf mem -D record

9. Analyze

CPU usage: (from mpstat)

Average:	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
----------	-----	------	-------	------	---------	------	-------	--------	--------	--------	-------

Average:	2	24.28	0.00	45.06	0.00	0.00	30.66	0.00	0.00	0.00	0.00
----------	---	-------	------	-------	------	------	-------	------	------	------	------

Network usage: (from sar -n DEV)

02:51:19 PM	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s	%ifutil
-------------	-------	---------	---------	--------	--------	---------	---------	----------	---------

02:55:25 PM	eth0	188525.00	196267.50	15738.66	106685.75	0.00	0.00	2.00	87.40
-------------	------	-----------	-----------	----------	-----------	------	------	------	-------

Syscall usage in 8 seconds: (from perf record -e probe_libc:malloc -e probe:tcp_recvmsg)

1,027,994	probe_libc:malloc
-----------	-------------------

256,699	probe:tcp_recvmsg
---------	-------------------

TLB usage: (from perf stat -dd)

6,543,139,385	dTLB-loads	#	817.829 M/sec	(71.49%)
---------------	------------	---	---------------	----------

6,057,670	dTLB-load-misses	#	0.09% of all dTLB cache hits	(57.20%)
-----------	------------------	---	------------------------------	----------

Memory usage: (from numactl -H)

node 0 size: 65371 MB

node 0 free: 63229 MB

Memory access samples in 8 seconds: (perf mem -D record)

total: 134024 samples

36K cpu/mem-loads,ldlat=30/P

97K cpu/mem-stores/P

the bottleneck is **memory allocator (can be misleading on RHEL 7.5 with kernel 3.10)**

Move to SLES-15 with new kernel 4.12, lots of perf improvements

perf record -a -g --all-kernel (all net relative)

perf record -a -g --all-user (yeah)

Samples: 29K of event 'cycles:ppp', Event count (approx.): 4339942980

Children	Self	Command	Shared Object	Symbol
- 20.39%	0.00%	nginx	[unknown]	[.] 0000000000000000
- 0				
+ 4.56%		0x1		
3.41%		int_malloc		
1.95%		_int_free		
1.18%		0		
1.00%		ngx_http_header_filter		
0.85%		ngx_http_headers_filter		
+ 0.75%		0x1e2e650		
0.53%		syscall_return_via_sysret		
4.81%	0.00%	nginx	[unknown]	[.] 0x08478b48f58948fb
4.70%	0.00%	nginx	[unknown]	[.] 0x0000000000000001
4.63%	4.63%	nginx	nginx	[.] ngx_vslprintf
4.36%	0.00%	nginx	[unknown]	[.] 0x0000000001e3f878
3.68%	0.00%	nginx	[unknown]	[.] 0x000000000000000a
3.67%	3.67%	nginx	libc-2.26.so	[.] _int_malloc
2.54%	2.54%	nginx	nginx	[.] ngx_http_finalize_request

And I got malloc usage data for this scenario:

```
st250:/home/www # perf record -e probe_libc:malloc -e probe:tcp_recvmmsg -aR -g --
output=/tmp/perf-probes.data -- sleep 8
st250:/home/www # perf script -i /tmp/perf-probes.data 2>/dev/null | grep malloc | awk
'{a[$1]++;}END{for (i in a)print i, a[i];}' | sort -rnk2 > libc.malloc.sys
st250:/home/www # cat libc.malloc.sys
nginx 998487
sleep 832
systemd 345
dbus-daemon 300
systemd-journal 254
sadc 212
systemd-logind 182
sshd 171
mpstat 62
sar 50
perf 2
```

10. Introduce jemalloc

Compile jemalloc:

```
[root@www jemalloc-5.1.0]# ./autogen.sh
```

```
[root@www jemalloc-5.1.0]# make & make install
```

Compile nginx with jemalloc:

```
[root@st50 nginx-1.15.5]# ./configure --prefix=/home/www/nginx0 --with-
pcre=/home/www/pcre-8.42 --with-zlib=/home/www/zlib-1.2.11 --with-debug --with-ld-opt="-
ljemalloc"
```

Restart Nginx:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```



```
numactl -C 2 --localalloc nginx/sbin/nginx
```

#keep all other settings same

11. Redo ab test

```
[root@dr1 www]# ab -n 800000 -c 300 http://www.example.com/logo.svg
```

This is ApacheBench, Version 2.3 <\$Revision: 1430300 \$>

Copyright 1996 Adam Twiss, Zeus Technology Ltd, <http://www.zeustech.net/>

Licensed to The Apache Software Foundation, <http://www.apache.org/>

Benchmarking www.example.com (be patient)

Completed 80000 requests

Completed 160000 requests

Completed 240000 requests

Completed 320000 requests

Completed 400000 requests

Completed 480000 requests

Completed 560000 requests

Completed 640000 requests

Completed 720000 requests

Completed 800000 requests

Finished 800000 requests

Server Software: nginx/1.15.5

Server Hostname: www.example.com

Server Port: 80

Document Path: /logo.svg

Document Length: 2649 bytes

Concurrency Level: 300

Time taken for tests: 22.148 seconds

Complete requests: 800000

Failed requests: 0

Write errors: 0

Total transferred: 2328000000 bytes

HTML transferred: 2119200000 bytes

Requests per second: 36120.88 [#/sec] (mean)

Time per request: 8.305 [ms] (mean)

Time per request: 0.028 [ms] (mean, across all concurrent requests)

Transfer rate: 102648.20 [Kbytes/sec] received

Connection Times (ms)

min mean[+/-sd] median max

Connect: 0 4 24.6 3 1007

```
Processing: 1 4 6.0 4 225
Waiting:    0 4 5.6 4 209
Total:      1 8 25.3 7 1015
```

Percentage of the requests served within a certain time (ms)

```
50%    7
66%    8
75%    8
80%    8
90%    8
95%    9
98%   11
99%   12
100% 1015 (longest request)
```

12. Analyze for jemalloc

CPU usage: (from mpstat)

```
Average: CPU  %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average:   2 19.54 0.00 35.12 0.00 0.00 32.18 0.00 0.00 0.00 13.15
```

Network usage: (from sar -n DEV)

```
02:51:19 PM IFACE rxpck/s txpck/s rxkB/s txkB/s rxcmp/s txcmp/s rxcmt/s %ifutil
02:51:21 PM eth0 211446.00 222039.50 17673.09 120630.16 0.00 0.00 0.00 98.82
```

Syscall usage in 8 seconds: (from perf record -e probe:tcp_recvmsg -e probe_libc:malloc)

```
840 probe_libc:malloc
289,287 probe:tcp_recvmsg
```

TLB usage: (from perf record)

```
6,861,897,872 dTLB-loads # 857.666 M/sec (76.95%)
18,741,425 dTLB-load-misses # 0.27% of all dTLB cache hits (61.60%)
```

Memory usage: (from numactl -H)

```
node 0 size: 65371 MB
node 0 free: 63200 MB
```

Memory access samples in 8 seconds: (perf mem -D record)

```
total: 113608 samples
30K cpu/mem-loads,ldlat=30/P
82K cpu/mem-stores/P
```

And malloc usage data now:

```
st250:/home/www # perf record -e probe_libc:malloc -e probe:tcp_recvmsg -aR -g --
output=/tmp/perf-probes.data -- sleep 8
st250:/home/www # perf script -i /tmp/perf-probes.data 2>/dev/null | grep malloc | awk
'{a[$1]++;}END{for (i in a){print i, a[i];}}' | sort -rnk2 > libc.malloc.sys.jemalloc
st250:/home/www # cat libc.malloc.sys.jemalloc
```

nginx 6883
systemd 901
dbus-daemon 635
systemd-journal 462
sleep 416
systemd-logind 208
sshd 165
perf 2

perf top shows NO memory allocator any more ~~ (actually CPU is not bottleneck)

```
6.29% nginx    [kernel.vmlinux] [k] system_call
5.05% nginx    [kernel.vmlinux] [k] sysret_check
1.14% swapper   [kernel.vmlinux] [k] memcpy
1.14% nginx    [kernel.vmlinux] [k] _raw_spin_lock
1.02% nginx    [e1000e]         [k] e1000_xmit_frame
0.97% nginx    libc-2.17.so      [.] __memcpy_ssse3_back
```

13. Conclusion

Memory allocation latency is critical to nginx, glibc malloc is blamed for years, and TCMalloc and jemalloc is developed to resolve this performance issue.

As a simple comparison before and after introducing jemalloc,

RPS is increased from 33585.56 to 36087.54 (+8%)
CPU %idle is increased from 0% to 13.15% (13%) while workload is +8% (total jemalloc > 20%)
CPU %sys is reduced from 45.50% to 35.12% (-10%) while workload is +8% heavier
Network util: increased from 87.40% to 98.82% (+11%) Saturated
TLB-loads is increased from 819.362 to 857.666 M/sec (+5%) while workload is +8% heavier
Syscall probe_libc:malloc is reduced from 926,719 to 840 (99%?) Deviation ?
Syscall tcp_recvmmsg increased from 231K to 289K, (+25%), why >8%? Deviation ?
99% latency has been reduced from 12ms to 12ms (0%) while workload is +8% heavier

14. Future works

This is a very simple test to resolve malloc performance bottleneck by introducing jemalloc. It demonstrates the iteration of simplifying the problem, finding performance bottleneck, fixing problem, and verifying it. And then continue this iteration into a higher performance state.

Will do more research on DPDK user-space TCP stack, because the later profile shows that bottleneck is in tcp stack of Linux kernel. I know it could be something complicated, but it's worthy to try, there is no free lunch.

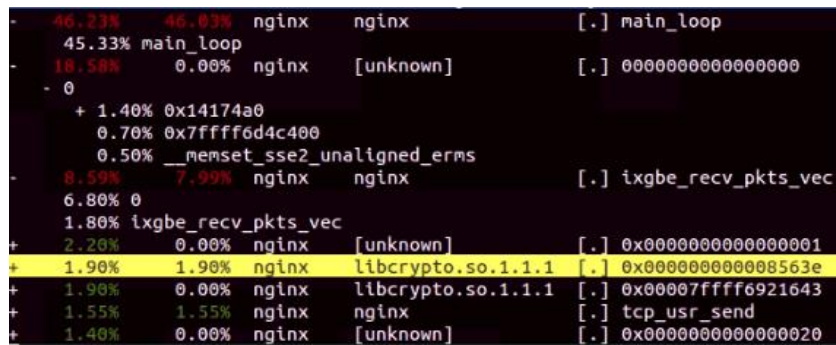
15. DPDK

Finally got a chance to have a look at DPDK on my server.

I chose the F-Stack implementation: <https://github.com/f-stack/f-stack>

The first glance of the readiness state is 99% CPU on my cores, it surprised me a bit. A minute later, I realize that it is in User Space Mode: means it should be a busy loop, and then I was released.

And the profile looks like this:



-	46.23%	46.03%	nginx	nginx	[.]	main_loop
-	45.33%		main_loop			
-	10.58%	0.00%	nginx	[unknown]	[.]	0000000000000000
-	0					
+	1.40%		0x14174a0			
	0.70%		0x7ffff6d4c400			
	0.50%		__memset_sse2_unaligned_erms			
-	8.59%	7.99%	nginx	nginx	[.]	ixgbe_rcv_pkts_vec
	6.80%		0			
	1.80%		ixgbe_rcv_pkts_vec			
+	2.20%	0.00%	nginx	[unknown]	[.]	0x0000000000000001
+	1.90%	1.90%	nginx	libcrypto.so.1.1.1	[.]	0x0000000000008563e
+	1.90%	0.00%	nginx	libcrypto.so.1.1.1	[.]	0x00007ffff6921643
+	1.55%	1.55%	nginx	nginx	[.]	tcp_usr_send
+	1.40%	0.00%	nginx	[unknown]	[.]	0x0000000000000020

It looks like not saturated, but my driver side still using interrupt, and usage of CPU0 is 100%, I need to start multiple ab processes, let's say 10, and we can add the throughput altogether.

And I got this:



3.19%	libcrypto.so.1.1.1	[.]	0x0000000000008563e
2.69%	nginx	[.]	tcp_output
2.21%	nginx	[.]	ff_dpd_k_if_send
2.05%	nginx	[.]	tcp_do_segment
1.79%	nginx	[.]	ixgbe_xmit_pkts
1.75%	libc-2.28.so	[.]	__memmove_avx_unaligned_erms
1.68%	libc-2.28.so	[.]	_int_free
1.64%	nginx	[.]	ip_output
1.63%	nginx	[.]	in_pcblookup_hash_locked.isra.1
1.60%	nginx	[.]	ngx_http_parse_header_line
1.55%	libcrypto.so.1.1.1	[.]	EVP_EncryptUpdate
1.55%	libc-2.28.so	[.]	malloc
1.51%	nginx	[.]	ngx_sprintf_num
1.42%	nginx	[.]	tcp_input

The total throughput is 91,595. Then I tried more ab, 20, 30, the throughput stays at 91k. I know the throughput of nginx server on a **single logical core** has reached it ceiling.

I then increase the server **logical cores to 2**, the throughput goes up to 183,180, well the scaling is not bad, and most importantly: there is nearly no sys & irq:

Average:	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
Average:	0	98.25	0.00	1.42	0.00	0.33	0.00	0.00	0.00	0.00	0.00
Average:	1	98.42	0.00	1.25	0.00	0.33	0.00	0.00	0.00	0.00	0.00
Average:	2	0.00	0.00	0.00	0.00	0.00	1.08	0.00	0.00	0.00	98.92
Average:	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00
Average:	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00

The %usr shown here is very noticeable, since there is a `idle_sleep(ms)` in “while (1)” of `main_loop` function

If set `idle_sleep=0`, the \$usr will be constantly equal to 97-99, and set `idle_sleep=1` \$usr will cost about 3% when idle, but the throughput will decrease from 183,180 -> 181,563 (-1%), I will keep this anyway, since it will show me the real workload.

Enabling jemalloc increased RPS from 181,563 to 184,589 (+1.7%)

3.22%	libcrypto.so.1.1.1	[.]	0x0000000000008563e
2.24%	nginx	[.]	tcp_output
2.10%	nginx	[.]	tcp_do_segment
2.07%	nginx	[.]	ff_dpd_kif_send
1.77%	nginx	[.]	in_pcblookup_hash_locked.isra.1
1.71%	nginx	[.]	ixgbe_xmit_pkts
1.56%	libc-2.28.so	[.]	__memset_avx2_erms
1.54%	nginx	[.]	rn_match
1.53%	nginx	[.]	ip_output
1.49%	libjemalloc.so.2	[.]	free
1.48%	libc-2.28.so	[.]	__memmove_avx_unaligned_erms
1.47%	libcrypto.so.1.1.1	[.]	EVP_EncryptUpdate
1.44%	nginx	[.]	ngx_sprintf_num
1.41%	nginx	[.]	ngx_http_parse_header_line
1.39%	nginx	[.]	tcp_input
1.28%	nginx	[.]	ngx_http_log_escape.part.0
1.18%	libcrypto.so.1.1.1	[.]	OPENSSL_cleanse
1.11%	nginx	[.]	uma_zalloc_arg
1.09%	nginx	[.]	main_loop
1.09%	libcrypto.so.1.1.1	[.]	0x0000000000008564c
1.03%	nginx	[.]	callout_reset_tick_on
1.01%	nginx	[.]	ngx_pcalloc
0.96%	libc-2.28.so	[.]	__memset_sse2_unaligned_erms
0.96%	nginx	[.]	ff_veth_input.isra.3
0.86%	nginx	[.]	ngx_hash_find
0.84%	nginx	[.]	ip_input
0.83%	nginx	[.]	ngx_http_process_request_headers
0.83%	libcrypto.so.1.1.1	[.]	EVP_CipherInit_ex
0.81%	libcrypto.so.1.1.1	[.]	EVP_CIPHER_CTX_reset
0.80%	nginx	[.]	ngx_http_header_filter

Then, I turn on `gzip` in `nginx.conf`, even though the transfer rate reduced from 13 to 3 (MB/s). it is not surprising, small packets didn't bring any performance benefit, since the network bandwidth is not current bottleneck.

When I extend to 4 lcore, each CPU avg usage is only about 40%, I checked the hugepages were used up.

```
echo 2048 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

Doubling large page memory will give the CPU usage up to 80%, continue to add

```
echo 40960 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

Let's add to 40960, which I assume it may support about 40 lcores, and add ab clients to 40.

And now, with 4 lcores, RPS goes up to 368,615, scaling factor = 2, not bad

Meanwhile the in/out %ifutil is about 15%

Average:	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s	%ifutil
Average:	enp0s20u1u5	1.06	1.06	0.08	0.10	0.00	0.00	0.00	0.00
Average:	eno2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Average:	ens1f1	0.06	1812744.69	0.02	150540.41	0.00	0.00	0.00	12.33
Average:	eno3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Average:	eno4	70.19	62.94	7.15	9.95	0.00	0.00	4.19	0.01
Average:	eno1	5.62	0.00	1.61	0.00	0.00	0.00	4.31	0.00
Average:	ens1f0	1806853.75	0.75	192683.14	0.05	0.00	0.00	0.00	15.78

Noticeably, if I enabled the **Hyper Threading** in ab side, the latency on the extra lcores is much higher than the pcores only: 99% latency increases from 8ms to 200ms,

I feel super tired today, will try more lcores & other options later