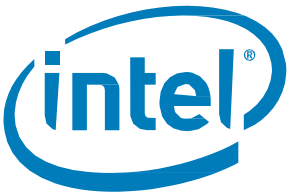


Characterizing Memcached* with Intel® Memory Drive Technology

Set-up and Configuration Guide for Benchmarking Evaluation

November 2017

Document Number: 336655-002US

Revision History

Revision Number	Description	Date
001	• Initial release	November 2017
002	• Changed classification to Public • Changed Table 1: Hardware Configuration for both Servers, page 6	November 2017

Contents

Figures

Tables

1 Introduction

Memcached* is a free and open source, high performance, distributed memory object caching system. It is suitable for large-scale deployments common on internet-facing applications and use cases where the data is mostly read oriented, and allows the retrievals to offload a more expensive database system. The memcached system is an all in-memory key-value store for typically small chunks (under 10kbytes) of arbitrary (strings, graphics objects) hydrated from results of database calls, API calls, or page rendering. A typical deployment involves offloading the most common small object reads in front of a database system, in order to improve query performance time and allow for lower cost, more elastic scalability.

§

2 Set Up Overview

The following list is an overview of the set-up steps covered in this guide:

1. Hardware and Operating System requirements
2. Configure network and test the network link
3. Understand and deploy run.sh script (benchmark and server setup tool)
4. Understand the results files and examples

§

3 Installation and Configuration

3.1 Hardware and Operating System Requirements

This document assumes Centos 7.3.1611* is pre-installed. The same steps could be applied to other operating system distributions with minor changes in the installation process.

For this test, two machines are needed, with 10GE connection between them. One will be the load generator that uses the memaslap* program, and the other machine will run memcached* as the server under test. The memcached server will compare:

1. DRAM-only – Dual Socket system with 512 GB DRAM (24 DIMMs x 32GB each) configured into the system. We utilized Operating System start-up parameters to limit the usable memory to 512GB.
2. Intel® Memory Drive Technology – the same Dual Socket system with just 128GB DRAM (32 DIMMs x 8GB each), and two 320GB Intel® Memory Drive Technology-enabled SSDs used for system memory. Keeping the same number of DIMMs ensures that when reducing the amount of memory on this system, from 512GB to 128GB, it keeps the equivalent memory bandwidth. While this setup can deliver up to 698GB of system memory, this test used Intel® Memory Drive Technology System Settings (F5) to limit the total memory to 512G – to match the DRAM-only setup above.

Table 1: Hardware Configuration for both Servers (load and server under test)

Hardware	Type
CPU	Intel® Xeon® CPU E5-2699 v4 @ 2.20 GHz
Server	Intel® Server 2600WT
Memory	memaslap server: 24 x 16GB Kingston* DDR4 @ 1600Mhz memcached (DRAM only): 24 x 32GB Micron* DDR4 @ 2667Mhz memcached (Intel® Memory Drive Technology only): 24 x 8GB Micron DDR4 @ 2667Mhz
Network	Intel Corporation I350 10GbE Gigabit Network Connection
Intel® Memory Drive Technology	2 x 320GB (one per CPU socket) Version 8.2.1455.29
Linux* OS distribution	CentOS 7.3.1611*
Kernel version	3.10.0-514.21.2.el7.x86_64
Memcached version	1.4.34
Memaslap version	1.0

System #1 is for the client load generation (memaslap) tool. System #2 is for memcached.

1. Install a recent copy of Centos* (CentOS 7.3* or 7.4*) or upgrade your systems:

```
# yum update -y
```

2. Install required software components for memcached and memaslap software stack:

```
# yum -y install gcc gcc-c++ libevent-devel


# wget https://memcached.org/files/memcached-1.4.34.tar.gz

# tar zxvf memcached-1.4.34.tar.gz

# cd memcached-1.4.34

# sed -i 's/if (enable_large_pages() == 0)/if (1 || enable_large_pages() == 0)/'
memcached.c

#./configure && make && make install && cd –


# wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-
1.0.18.tar.gz

# tar zxvf libmemcached-1.0.18.tar.gz

#cd libmemcached-1.0.18

# CXXFLAGS=-lpthread ./configure --enable-memaslap && make && make install && cd –
```

3. Identify the 10GigE IP addresses of each system, and make sure root can ssh from one machine (a) to itself and (b) to the other system without password (using keys). The below example assumes that the machines' addresses are 192.168.2.3 and 192.168.2.4.

   Note: If the below fails, you typically should run ssh-keygen on the local system.

```
for ip in 192.168.2.3 192.168.2.4; do ssh-copy-id root@$ip; done
```

4. Test out the network links between systems. Run on System #1.

```
# while : ; do nc -vvlnp 12345 >/dev/null; done
```

   On System #2, run:

```
# for h in 192.168.2.3; do dd if=/dev/zero bs=1M count=10K | nc $h 12345; done
```

   Execute the same in the reverse direction. Make certain the performance reached is a fully 10Gigabit connection.

3.2 Setting Up the run.sh script for the System

1. Now import the runscript that executes either memcached or memaslap, based on the machine designated to execute as one or the other role. There are two bash variables set up for the IP address of each server; otherwise, the script is generic.

2. To execute the memcached server run:

```
# ./run.sh mc
```

3. To execute the memaslap run:

```
# ./run.sh ms
```

Where the run.sh file is:

```
#!/usr/bin/bash

# memcached server address
MEMCACHED=192.168.4.2
MEMASLAPD=192.168.4.1

# value size in bytes
VAL=1024

# net BW in Gbps
BW=10

if [ "x$1" != "xmc" -a "x$1" != "xms" ]; then
  echo "Usage: $0 [mc | ms]"
  exit
fi

mc_cpus=`ssh $MEMCACHED cat /proc/cpuinfo | grep processor | wc -l`
ms_cpus=`ssh $MEMASLAPD cat /proc/cpuinfo | grep processor | wc -l`

# This fits well MC machine with 88 CPUs and MS machine with 144 CPUs
mc_inst=$((mc_cpus/2))
mc_threads=4
ms_threads=$((mc_inst))
ms_con=$((mc_cpus*2))

range="`seq 10001 $((10001+mc_inst-1))`"
log=results.`date +%F.%T`
ulimit -n 1048576

if [ "x$1" = "xmc" ]; then
  for i in ksm.service ksmtuned.service irqbalance.service tuned.service; do
    systemctl stop $i
  done
  rmmod ip_tables
  cpupower frequency-set -g performance; cpupower set -b 0
```

```
  msi_dir=`grep \`ifconfig | grep $MEMCACHED -A2 | grep ether | awk '{print $2}'\`
/sys/class/net/*/address | sed 's/[^/]*$/device\/msi_irqs/'`
  echo msi_dir: $msi_dir


  msis=`ls -1 $msi_dir | wc -l`
  echo msis: $msis


  step=$((mc_cpus/msis))
  step=$((step<1?1:step))
  echo step: $step


  j=0
  for msi in `ls $msi_dir`; do
    echo -n "$msi current: `cat /proc/irq/$msi/smp_affinity_list`"
    echo $j-$((j+step-1)) > /proc/irq/$msi/smp_affinity_list
    echo " now: `cat /proc/irq/$msi/smp_affinity_list`"
    j=$((j+step))
    j=$((j>=mc_cpus?j=0:j))
  done


  step=$((mc_cpus/mc_inst))
  step=$((step<2?2:step))
  mem=`free -m | grep Mem | awk '{print $2}'`
  size=$((mem*90/100/$mc_inst))


  echo Running $mc_inst instances each with $size MB
  j=0
  for i in $range; do
    LD_LIBRARY_PATH=/usr/local/lib taskset -c $j-$((j+step-1)) memcached -u nobody -m
$size -p $i -U $i -c 43690 -B auto -t $mc_threads -r -d -L
    j=$((j+step))
    j=$((j>=mc_cpus?j=0:j))
  done
  exit 0
fi


# we are here - $1 is "ms"
for i in tuned.service; do
  systemctl stop $i
```

```
done


cpupower frequency-set -g performance; cpupower set -b 0


BWb=$((10*1024*1024*1024/8))
MAXops=$((BWb/VAL))
cmd="uname -a; echo; free -g; echo; lscpu; echo; if [ -x /usr/local/bin/vsmpversion ];
then vsmpversion -vvv; else echo 'Running NATIVE'; fi; echo"


(echo "Max TPS possible: $MAXops"; echo) | tee -a $log
cat << EOF >> $log
mc_cpus $mc_cpus
mc_threads $mc_threads
mc_inst $mc_inst

ms_cpus $ms_cpus
ms_threads $ms_threads
ms_con $ms_con

EOF
(echo "===>> memcached system -"; ssh $MEMCACHED "$cmd") >> $log
(echo "===>> memaslap system -" ; eval $cmd) >> $log

cat << EOF > config
generated keys
key
128 128 1
value
$VAL $VAL 1
cmd
0 0.1
1 0.9
EOF


(echo "===>> config -" ; cat config; echo) >> $log


for i in $range; do
  srv="${srv}$MEMCACHED:$i,"
```

```
done
srv=${srv::-1}


params="-s $srv -F config -T $ms_threads -c $ms_con -t 864000s -S 1s -d 1 --
win_size=100k -R"
(echo "===>> params -" ; echo $params; echo) >> $log


memaslap $params >> $log &


while : ; do
  sleep 10
  ps -efa | grep memaslap | grep -v grep > /dev/null
  if [ $? -eq 1 ]; then
    exit
  fi
 (ssh $MEMCACHED free -m | grep Mem: ; tail -100 $log | grep "Total Statistics" -A3 |
grep "Total Statistics" -B5 | tail -4| head -2 | tac) | paste -d " " - - - | tr -s ' '
2>&1 >> $log.mem
done
```

3.3 Configuration of the Server

The servers are dual-socket machines with a 10Gbit NIC between them on the same rack.

Here is a screenshot of the top of the screen showing the basic machine configuration from the BIOS utility, showing board type, CPU, board version, and memory capacity:

```
S2600WT
Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz                    @2.20 GHz
SE5C610.86B.01.01.0019.101220160604                          262144 MB RAM
Copyright (c) 2010-2016, Intel Corporation
```

The run.sh does a number of things, including:

- Turns off ksm, ksmtuned, irqbalance, and tuned – these services are not needed for this application, and consume significant CPU time.
- Sets performance mode on the CPUs.
- The script sets irq affinity, so interrupts will be distributed evenly between the system's cores.
- It increases the number of connections to test memcached effectively.
- The workload tries to avoid missed keys, which will be wasted requests.

To ensure bandwidth, at least two Intel® Optane™ P4800X SSDs are needed. The more Intel® Optane™ Drives you use, the more memory bandwidth you will get in this system, but in this memcached benchmark, with high-end CPUs and 10GE network between the nodes, two Intel® Optane™ Drives will provide satisfactory results.

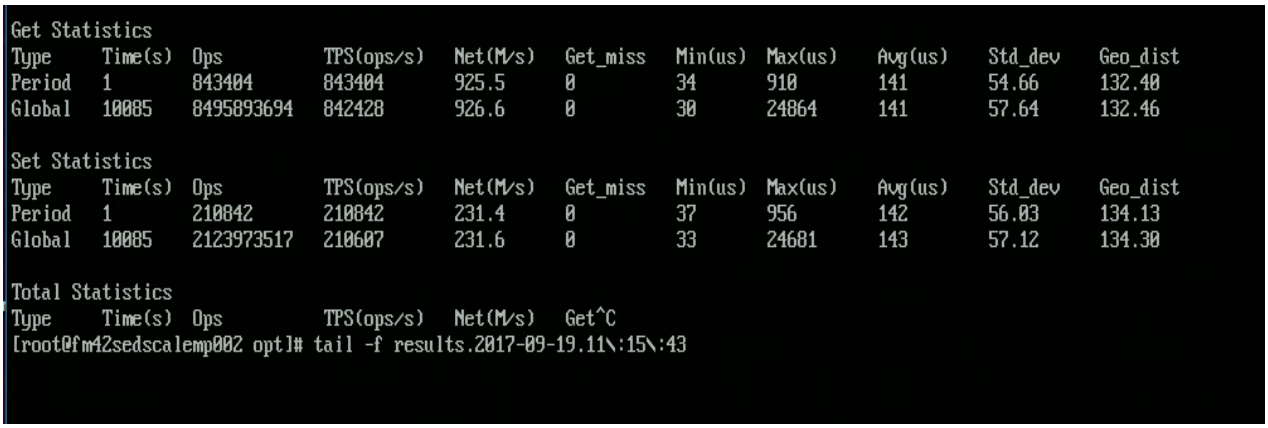
3.4 Observed Performance

The run.sh is designed to run one million 1.1kb transactions or more across a number of memcached processes, each one pinned to a core.

In this example, one memcached process per core, with four threads per process is used.

You will be able to monitor the performance of the GETs and SETs.

Figure 1: GETs and SETs



Intel® Memory Drive Technology will deliver similar performance to DRAM; the performance should be evaluated after RESIDENT memory stabilizes, and not during memory ramp up. For example, in this test, we state that each memcached process will consumer roughly 11GB of resident memory. During memcached ramp up, the application bandwidth may drop for a while as Intel® Memory Drive Technology determines the nature of the memory blocks and what can be cached efficiently. Resident memories can be tracked using many different utilities or monitors, such as top* or htop*. The process of filling memory and stabilizing the load will take many minutes, up to 1 hour, based on your configuration.

It is recommended to set the read/write ratios of the load server to match your real application, so that the benchmark result will be as close as possible to what you should expect from your in-memory key-value store application. For the purpose of the test's published results shown below, we used a 90/10 READ/WRITE ratio, and a 'value' size of 1K. See run.sh above for all settings to memaslap. The active connection count into the server was 176, per the parameters in the run.sh script.

Table 2: 512GB All DRAM System (11.2GB assigned per memcached process)

Metric	Value
CPU average utilization	11%
Memory assigned usage (Large Page support set)	494 GB assigned to 44 memcached processes
Network average usage	9344 Mbit / sec
GET QPS (queries per second)	945016
SET QPS (queries per second)	106001
GET request average latency (us)	153
GET request standard deviation	51.04
SET request average latency (us)	156
SET request standard deviation	53.72

Table source: Results file produced by the run.sh ms (memaslap load tier). End of file summary results. Filename: results.2017-09-27.15:06:09

Table 3: 128GB DRAM + Intel® Memory Drive Technology System (10.544 GB assigned per memcached process)

Metric	Value
CPU average utilization	3%
Memory assigned usage (Large Page support set)	464 GB assigned to 44 memcached processes
Network average usage	9420 Mbit / sec
GET QPS (queries per second)	951811
SET QPS (queries per second)	105757
GET request average latency (us)	154
GET request standard deviation	63.59
SET request average latency (us)	208
SET request standard deviation	92.39

Table source: Results file produced by the run.sh ms (memaslap load tier). End of file summary results. Results.2017-10-19.10:50:35

Figure 2: Intel® Memory Drive Technology: Memcached Performance Throughput

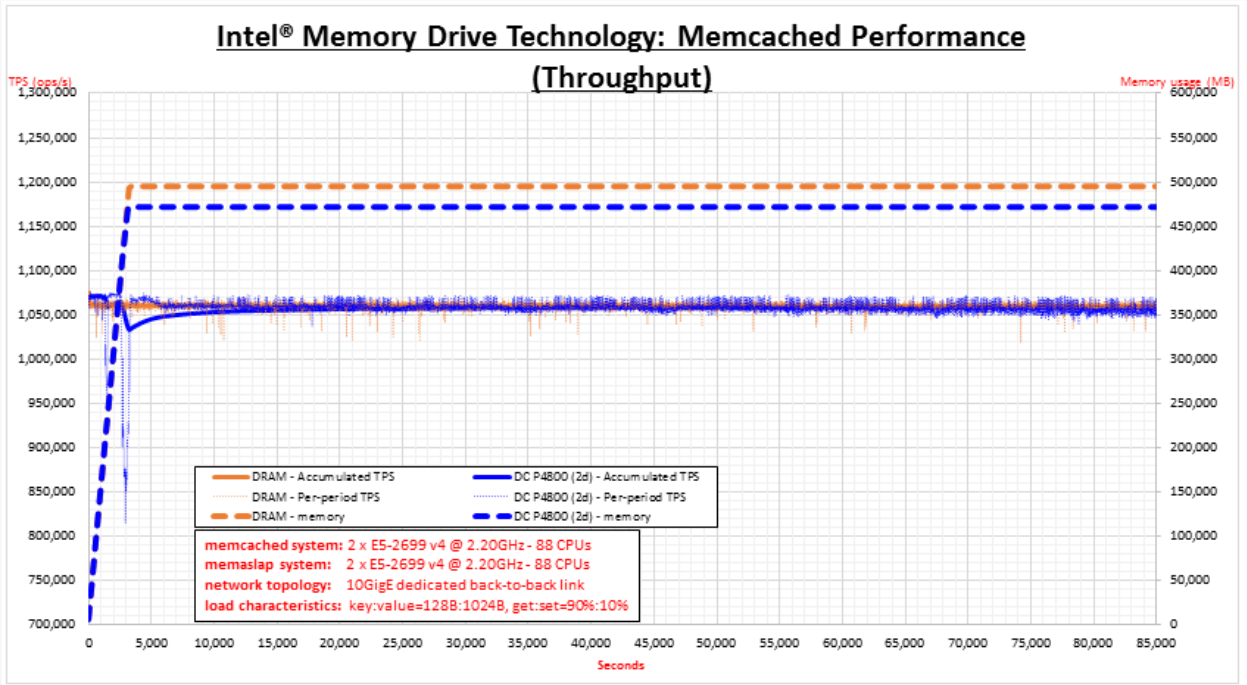
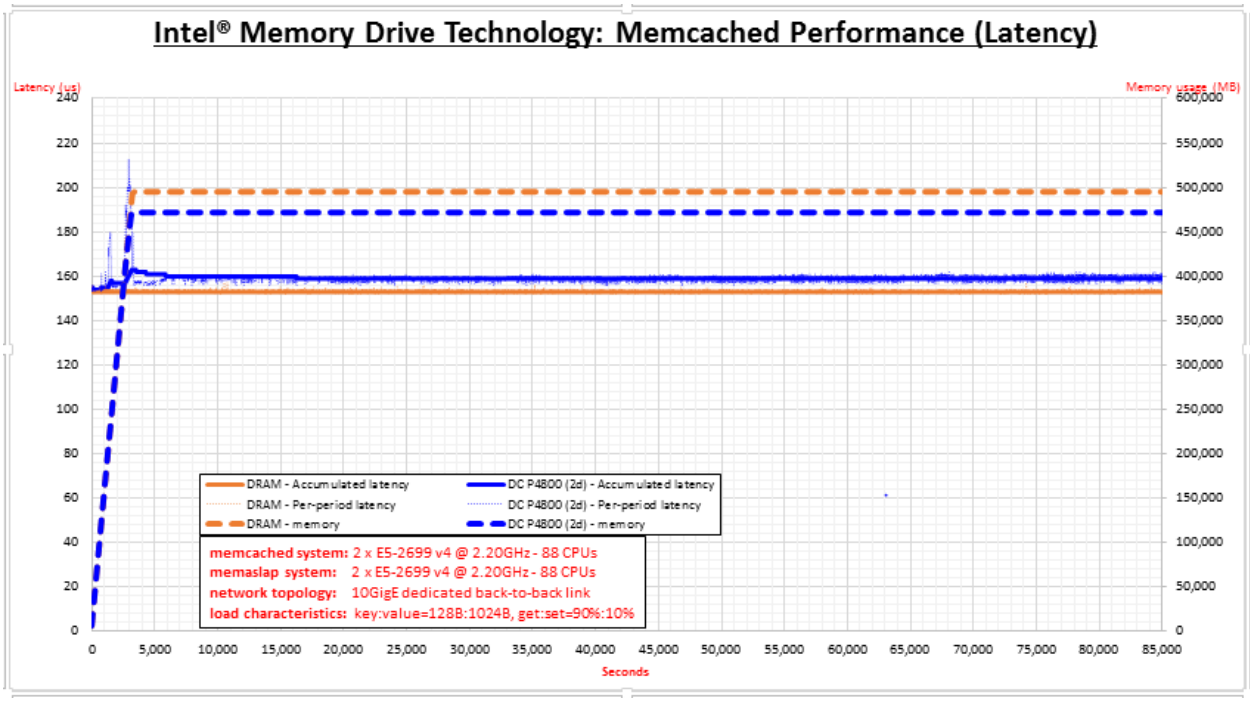


Figure 3: Intel® Memory Drive Technology: Memcached Performance Latency



§