.

1 Introduction

This document provides installation and run guidelines for the different benchmarking applications we used. Most of the benchmarking applications are contained in larger **Benchmark Suites** and a few come as standalone applications. The benchmarks were run on a linux PV instance and on a native linux system. The installation guidelines are valid for both systems (ie virtual machine and host machine).

- **Benchmark Suites**
  - SPEC CPU 2006 : Compute single-threaded workloads.
  - PARSEC3.0 : Compute multi-threaded workloads.
- **Standalone benchmarks**
  - Redis : Open-source in-memory database
  - MongoDB : Document oriented database program
  - Graph500 : Breadth-first search in a large undirected graph

2 Installation guidelines

2.1 SPEC CPU 2006

SPEC CPU 2006 is a benchmark suite containing 31 different benchmarking applications. We ran our tests using the following benchmarks:

- bzip2 : Compression
- omnetpp : Discrete Event Simulation
- astar : Path-finding Algorithms
- mcf : Combinatorial Optimization
- gobmk : AI
- hmmer : Search gene-sequence
- libquantum : Quantum computing
- xalancbmk : XML Processing

2.1.1 Installing SPEC CPU

To install SPEC CPU 2006 benchmark suite, start by downloading its ISO image. Create a folder on your system and mount the image there. In our case, we mount the image and get into the folder as follows:

```
$ mount cpu2006.iso /media/spec2006
$ cd /media/spec2006
```

Invoke the install script **install.sh** and enter the destination directory when prompted

```
$ ./install.sh

SPEC CPU2006 Installation

Top of the CPU2006 tree is '/media/SPEC_CPU2006'
Enter the directory you wish to install to (e.g./usr/cpu2006)
/cpu2006

Installing FROM /media/SPEC_CPU2006
Installing TO /cpu2006

Is this correct? (Please enter 'yes' or 'no')
yes

The following toolset is expected to work on your platform. ..
...more logs...
```

After successful installation of the benchmark suite, we are ready to test some benchmarking applications. Change your current working directory to the location of the new SPEC CPU tree.

```
$ cd /cpu2006
```

Set up environment variables and paths for SPEC.

```
$. ./shrc
```

Create a config file corresponding to your test system information.

```
$ cp Example-linux64-amd64-gcc43+.cfg mytest.cfg
```

Build one benchmark. We build bzip2 in our case

```
$ runspec --config=mytest.cfg --action=build --tune=base bzip2
....log...messages
Build successes: 401.bzip2(bas)  <<-- what we want to see

Build Complete
```

After a successful build, we are ready to run the benchmark. To run bzip2 on a ref workload, we launch the following command:

```
runspec --config=mytest.cfg --size=ref --noreportable
--tune=base --iterations=1 bzip2
```

All logs corresponding to benchmark results are found in the results directory. To run multiple benchmarks you could create a script to build and run your desired benchmarks. More information could be found on the official website : `spec cpu installation guidelines`.

2.2 PARSEC3.0

The **Princeton Application Repository for Shared-Memory Computers** (PARSEC) is a benchmark suite composed of multithreaded programs. The benchmark suite is composed of 13 different workloads and all its applications and input sets are available as open source free of charge from the `PARSEC website`. We used the following benchmarks from the PARSEC suite:

- Canneal : Simulated cache-aware annealing to optimize routing cost of a chip design.
- Dedup : Next-generation compression with data deduplication.
- Streamcluster : Online clustering of an input stream.

2.2.1 Installing PARSEC suite

Download PARSEC 3.0

```
$ wget http://parsec.cs.princeton.edu/download/
  3.0/parsec-3.0-core.tar.gz
```

Unpack PARSEC 3.0 package

```
$ tar -xzf parsec-3.0.tar.gz
$ cd parsec-3.0
$ ls
 bin CHANGELOG  config CONTRIBUTORS  env.sh
 ext FAQ  LICENSE  man  pkgs  README  version
```

Setup environment variables before running PARSEC

```
$ source env.sh
```

Build/run/uninstall benchmarks. Here we build one benchmark : streamcluster. All other benchmarks can be built and run the same way.

```
$ parsecmgmt -a build -p streamcluster
  ...
[PARSEC] Copying source code of package parsec.streamcluster.
[PARSEC] Running 'env version=pthreads /usr/bin/make':
```

Run streamcluster on the default input.

```
$ parsecmgmt -a run -p streamcluster
  ...
[PARSEC] [---------- Beginning of output ----------]
PARSEC Benchmark Suite Version 3.0
read 10 points

real    0m0.016s
user    0m0.002s
sys     0m0.001s
```

To run our benchmarks with other inputs, we start by downloading the inputs file.

```
$ wget http://parsec.cs.princeton.edu/download/3.0/
  parsec-3.0-input-sim.tar.gz

...
```

Move the input file to the same level of the PARSEC 3.0 root directory for unpacking.

```
$ mv parsec-3.0-input-sim.tar.gz ~/
$ cd ~/
$ ls
parsec-3.0   parsec-3.0-input-sim.tar.gz
$ tar -xzf parsec-3.0-input-sim.tar.gz
```

Run the benchmark using the new inputs.

```
$ parsecmgmt -a run -p streamcluster -i simsmall
  ...

$ parsecmgmt -a run -p streamcluster -i simlarge -n 2
  ...
```

More information on PARSEC benchmarking could be found on the official website : `PARSEC Benchmarking`.

2.3 Redis

Redis is an open-source in-memory database project implementing a distributed, in-memory key-value store with optional durability.

2.3.1 Installing Redis

Download the latest Redis tar ball from the redis.io web site, decompress and compile

```
wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
```

```
cd redis-stable
make
```

After the compilation the src directory inside the Redis distribution is populated with the different executables that are part of Redis:

- **redis-server** is the Redis Server itself.
- **redis-sentinel** is the Redis Sentinel executable (monitoring and failover).
- **redis-cli** is the command line interface utility to talk with Redis.
- **redis-benchmark** is used to check Redis performances.
- **redis-check-aof** and **redis-check-dump** are useful in the rare event of corrupted data files.

We are concerned only with benchmarking. We start by checking if Redis is working; cd into the src directory.

```
$ redis-cli ping
PONG
```

The above result means you are good to go. All executables are in the src directory so this should be your main working directory. Open a terminal in the test system and run redis-server.

```
$ ./redis-server
[28550] 01 Aug 19:29:28 # Warning: no config file specified..
[28550] 01 Aug 19:29:28 * Server started, Redis version 2.2..
[28550] 01 Aug 19:29:28 * Server ready to accept connections
... more logs ...
```

After starting redis-server, to run benchmark tests, open a different terminal and run your test commands. For example:

```
$ redis-benchmark -t set,lpush -n 100000 -q
SET: 74239.05 requests per second
LPUSH: 79239.30 requests per second
```

NB: When running a PV instance, you can ssh into your virtual machine from the host so you can access multiple terminals.
More information on redis benchmarking could be found on the official website : `Redis Benchmarking`.

2.4 Memcached

Here we assume that you already have memcached installed on your machine. We would use mcperf as our benchmarking tool for memcached. Start by downloading the `mcperf tarball`. Uncompress and install it with the following commands

```
$ tar xzf mcperf-0.1.0.tar.gz
$ cd mcperf-0.1.0
$ ./configure
$ make
$ sudo make install
```

To benchmark memcached with mcperf, start by launching the memcached daemon.

```
$ memcached -d -m 1024 -u root -l 127.0.0.1 -p 11211 on shell
```

Memcached is now running on localhost port 11211. The following example creates 1000 connections to a memcached server running on localhost:11211. The connections are created at the rate of 1000 conns/sec and on every connection it sends 10 'set' requests at the rate of 1000 reqs/sec with the item sizes derived from a uniform distribution in the interval of [1,16) bytes.

```
$ mcperf --linger=0 --timeout=5 --conn-rate=1000 \
  --call-rate=1000 --num-calls=10 --num-conns=1000 \
  --sizes=u1,16

 Total: connections 1000 requests 10000 responses 10000

Connection rate: 991.1 conn/s ..
Connection time [ms]: avg 10.3 min 10.1 max 14.1 stddev 0.1
Connect time [ms]: avg 0.2 min 0.1 max 0.8 stddev 0.0

Request rate: 9910.5 req/s (0.1 ms/req)
Request size [B]: avg 35.9 min 28.0 max 44.0 stddev 4.8

Response rate: 9910.5 rsp/s (0.1 ms/rsp)
Response size [B]: avg 8.0 min 8.0 max 8.0 stddev 0.0
Response time [ms]: avg 0.2 min 0.1 max 13.4 stddev 0.00
Response time [ms]: p25 1.0 p50 1.0 p75 1.0
.................Other Logs.........................
```

2.5 Graph500

For our graph500 benchmark, we would be using NETALX. NETALX is a customized Breadth First Search (BFS) implementation for the Graph500 list. It can conduct BFS to large graphs whose sizes exceed the capacity of DRAM on a machine. The basic idea of the NETALX implementation is described in the following paper: Keita Iwabuchi, Hitoshi Sato, Yuichiro Yasui, Katsuki Fujisawa, Satoshi Matsuoka, "NVM-based Hybrid BFS with memory efficient data structure", 2014 IEEE International Conference on Big Data, 529-538, Oct 2014.

To install NETALX, clone the github repository and install as follows:

```
$ git clone https://github.com/htsst/netalx.git
$ cd src
$ make
```

The different execution options can be found on the github page here. As an example, to execute BFS on a graph with 2^{26} vertices (SCALE26) using only DRAM with alpha = 128 and beta = 8, we run the following command.

```
$ ./graph500 -s 26 -k 128:8
```

For more information, visit the github page: `netalx`.