Testing

Objectives

At the end of this module, you will be able to:

- Load test data into an Aerospike database using the Aerospike Java Benchmark tool.
- Run benchmark tests
- Testing SSDs with the ACT

These tasks are ones that you can use in your environments. Doing proper benchmarking will not only help to determine what your performance is, but also useful for finding basic connectivity issues.

Benchmarking Dos And Don'ts

Do:

- Understand what you are benchmarking. Start with the most simple use case (single server, RAM storage, small objects), to make sure you understand the upper limits of the system. Then change things one at a time to bring it closer to your production configuration.
- Be aware of what you are bottlenecked on.
- Be prepared to have more power on the clients than server.
- Know if you will be impacted by network congestion.

Don't:

- Expect high performance with VMs. You must be very careful when testing in a public cloud such as Amazon.
- Expect realistic number when the client and server are on the same host.

One of the most common problems is finding network congestion.

Often this will requires network administrators to help determine if there are network issues causing the problem.

Use standard Linux commands like top to detect your bottlenecks.

The Java Benchmark Tool

+ Multi-platform
+ Portable
+ Easy to use
+ Total throughput easily seen from database nodes

– Poor performance with objects larger than 4 KB
– Must use many processes/servers to get to high throughput

The Java benchmark tool currently generates random text using ineffective methods. So large objects are not created quickly. Be aware of this if you want to test large objects and you will quickly saturate the CPUs on the clients.

If you want to test with large objects and need performance, you may want to use the benchmarking tool built into the C client. This is specific to Linux.

Java Benchmark Syntax

```
> ./run_benchmarks [options]
```

Flag	Description
-b, --bins <arg>	The number of bins (like columns) of data.
-g, --throughput <arg>	The target total throughput. This will include all operations.
-h, --host <arg>	A seed node (any node in the cluster). The tool will learn about the other nodes from this one.
-p, --port <arg>	The port on the Aerospike node to connect to (default 3000)
-k, --keys <arg>	This is the number of keys/records to use. If you select a number too small, you will get write contention. Too large and you may run out of memory.
-s, --set <arg>	The set where the data will be loaded.
-latency <arg>	Produce a table of latencies as measured from the client. This can be compared with the latencies on the server to see where any bottleneck may lie. Recommended value "-latency 7,1"
-n, --namespace <arg>	The namespace to test against. This is similar to a "database" in a relational database.
-o, --objectSpec <arg>	The type of object to use: "I" is an integer, "S:<size>" is the size of the object. Is is recommended to start with small objects to remove the network
-w, --workload <arg>	Type of workload: I - Insert only. test stops when the final record is loaded (based on –k above) RU,<percent> - The read percentage. "90" means a 90% read rate.
-z, --threads <arg>	The number of threads used by the benchmark. For bare metal, 64 is a good value. For VMs, you should use 4-8 threads/core.

It is very important to use the appropriate parameters with the –latency command. The defaults are often too coarse.

The Java Benchmark Tool

The Aerospike benchmark tool can be found within the full Aerospike Java Client. For this class, the tool has been pre-compiled and loaded onto the Aerospike Training AMI.

To see it, go to the appropriate directory (one line)

```
> cd /home/aerotraining/packages/aerospike-client-java-3.0.26/benchmarks
```

Loading Test Data

Loading Data Example

Prior to running any tests, you should load data into the database. There is a script on the training server that will do this called `test_load.sh`. The active contents of this file are:

```
./run_benchmarks     -h 127.0.0.1 \   # Local DB server
                     -p 3000 \        # Local port
                     -n test \        # Namespace "test"
                     -k 100000 \      # Load 100,000 records
                     -s testset \     # Use the set "testset"
                     -latency "7,1" \ # Show latency numbers
                     -o S:100 \       # Use 100 byte string
                     -w I \           # Insert only
                     -z 8             # Use 8 threads
```

This will take a few minutes on your VM. When the script has loaded all the data, it will exit.

You will want to do tests on reading/writing data. However, it is important to pre-load data prior to running these tests.

Loading Data Example Output

```
[aerotraining@ip-10-231-34-97 benchmarks]$ ./test_load.sh
Benchmark: 127.0.0.1:3000, namespace: test, set: testset, threads: 8, workload: INITIALIZE
keys: 100000, start key: 1, key length: 10, bins: 1, throughput: unlimited, debug: false
write policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500
bin type 1: string[100]
2014-08-27 22:12:52.091 INFO Thread 1 Add node BB9ED14150B0022 127.0.0.1:3000
2014-08-27 22:12:52.137 write(count=0 tps=0 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write    0%   0%   0%   0%   0%    0%    0%
2014-08-27 22:12:53.141 write(count=2432 tps=2432 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   90%  10%   9%   8%   7%    5%    1%
2014-08-27 22:12:54.170 write(count=6492 tps=4060 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   92%   8%   7%   6%   5%    3%    1%
2014-08-27 22:12:55.194 write(count=12705 tps=6213 timeouts=0 errors=0)
...
```

While this is running, check on the AMC.

Notice here that there are latencies given for writes.

The numbers represent the that exceed the timemarks at the top.

Running Benchmarks

Running Balanced Workload

The first test to run is on a balanced workload. This will test database operations that are 50% read/50% write. This is in a file called `test_balanced.sh`. The active contents of this file are:

```
./run_benchmarks    -h 127.0.0.1 \   # Local DB server
                    -p 3000 \        # Local port
                    -n test \        # Namespace "test"
                    -k 100000 \      # Load 100,000 records
                    -s testset \     # Use the set "testset"
                    -latency "7,1" \ # Show latency numbers
                    -o S:100 \       # Use 100 byte string
                    -w RU,50 \       # Read at 50%
                    -z 8             # Use 8 threads
```

This script will continue running until it is stopped.

You can run an intensive test using a balanced workload. These numbers show reading 50% of the time and updating 50% of the time.

When interpreting these remember the following:

- An update on this test is actually a read and a write. This is due to the need to update existing data. While you can replace the data rather than update, the Java benchmark tool currently only does an update.
- Keep in mind that when writing data, you will be writing a number of copies based on the replication factor. For a single node, this will be always be 1.
- You may need to run multiple copies of these scripts to maximize performance.

Running Balanced Load Example Output

```
aerotraining@ip-10-231-34-97 benchmarks]$ ./test_balanced.sh
Benchmark: 127.0.0.1:3000, namespace: test, set: testset, threads: 8, workload: READ_UPDATE
read: 50% (all bins: 100%, single bin: 0%), write: 50% (all bins: 100%, single bin: 0%)
keys: 100000, start key: 0, key length: 10, bins: 1, throughput: unlimited, debug: false
read policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500, reportNotFound: false
write policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500
bin type 1: string[100]
2014-08-27 22:14:56.269 INFO Thread 1 Add node BB9ED14150B0022 127.0.0.1:3000
2014-08-27 22:14:56.329 write(tps=2 timeouts=0 errors=0) read(tps=0 timeouts=0 errors=0)
total(tps=2 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   89%  11%  11%  11%  11%    5%    2%
read    86%  14%  13%  13%  13%    9%    2%
2014-08-27 22:14:57.577 write(tps=1458 timeouts=0 errors=0) read(tps=1402 timeouts=0
errors=0) total(tps=2860 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   86%  14%  13%  12%  11%    6%    1%
read    89%  11%  10%   9%   8%    5%    2%
2014-08-27 22:14:58.585 write(tps=1994 timeouts=0 errors=0) read(tps=1934 timeouts=0
errors=0) total(tps=3928 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   93%   7%   7%   6%   5%    3%    1%
read    94%   6%   5%   5%   4%    2%    1%
...
```

While this is running, check on the AMC.

Running High-read Workload

You can then run a workload based on high read rates. This will test database operations that are 95% read/5% write. This is in a file called `test_highread.sh`. The active contents of this file are:

```
./run_benchmarks    -h 127.0.0.1 \   # Local DB server
                    -p 3000 \        # Local port
                    -n test \        # Namespace "test"
                    -k 100000 \      # Load 100,000 records
                    -s testset \     # Use the set "testset"
                    -latency "7,1" \ # Show latency numbers
                    -o S:100 \       # Use 100 byte string
                    -w RU,95 \       # Read at 95%
                    -z 8             # Use 8 threads
```

This script will continue running until it is stopped.

Running High-read workload Example Output

```
[aerotraining@ip-10-231-34-97 benchmarks]$ ./test_highread.sh
Benchmark: 127.0.0.1:3000, namespace: test, set: testset, threads: 8, workload: READ_UPDATE
read: 95% (all bins: 100%, single bin: 0%), write: 5% (all bins: 100%, single bin: 0%)
keys: 100000, start key: 0, key length: 10, bins: 1, throughput: unlimited, debug: false
read policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500, reportNotFound: false
write policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500
bin type 1: string[100]
2014-08-27 22:18:08.031 INFO Thread 1 Add node BB9ED14150B0022 127.0.0.1:3000
2014-08-27 22:18:08.121 write(tps=7 timeouts=0 errors=0) read(tps=68 timeouts=0 errors=0)
total(tps=75 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   57%  43%  43%  29%  14%    0%    0%
read    91%   9%   9%   9%   7%    3%    0%
2014-08-27 22:18:09.130 write(tps=107 timeouts=0 errors=0) read(tps=2650 timeouts=0
errors=0) total(tps=2757 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   85%  15%  14%  11%  11%    4%    2%
read    90%  10%   9%   8%   7%    4%    1%
2014-08-27 22:18:10.179 write(tps=263 timeouts=0 errors=0) read(tps=5012 timeouts=0
errors=0) total(tps=5275 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   93%   7%   5%   5%   5%    3%    2%
read    94%   6%   5%   5%   4%    2%    1%
...
```

While this is running, check on the AMC.

Testing SSDs with the ACT

SSD Testing

Aerospike has specialized the database to make the most out of SSDs. But not all SSDs have the same performance. Getting the most means being able to test.

Aerospike has open sourced the Aerospike Certification Tool (ACT), which is available at Github (https://github.com/aerospike/act). A copy has been put into the AWS instance at:

/home/aerotraining/packages/act-3.0

What the ACT Does

The ACT is a low level tool used to test the performance of SSDs as Aerospike uses them. The default settings for the tests are:

- Reads are of 1.5 KB objects
- Writes are in large blocks of 128 KB
- Reads and writes are simultaneous
- Simulates Aerospike garbage collection
- Does not use network
- Published results are based on a single drive. Aerospike can use these in parallel, but linearity depends on several factors in the hardware.
- Tests will work on a factor of "x". "1x" was the performance of a good SSD in 2011. This is the equivalent of 2,000 reads/second with 1,000 simultaneous writes/second. "2x" is twice that, "3x" is 3 times, etc. A good SATA drive in 2015 will support 3x speeds.

What Affects ACT Tests

- SSDs are not all alike. Higher model numbers are not always better.
- Changing the object size, read/write ratio, etc will change the results.
- RAID controller – How the drives are connected is extremely important.
  - Low cost controllers will always add a few ms of latency.
  - Even high cost RAID controllers may have issues. If you are using one of these, please see our note at: http://www.aerospike.com/docs/operations/plan/ssd/lsi_megacli.html
- Performance can depend a lot on overprovisioning (OP). Total OP should be between 22%-29%.
- Firmware can also have a big impact. Generally, newer firmware is better.

Running ACT Tests - Preparation

- You will be destroying any data on the drive/partition. Make sure you are not using the drive with the OS or any important data.
- Make sure you know the device IDs of the device(s)/partitions(s) (e.g. /dev/sdb, /dev/xvdb, etc).
- Check the Aerospike Web site for devices that Aerospike have tested, this will form a good basis for your testing. http://www.aerospike.com/docs/operations/plan/ssd/ssd_certification.html
- Download and compile the ACT (https://github.com/aerospike/act).
- Prior to running any tests, run the program `actprep` on each device/ partition you are going to test. This will place random data on the device.
- If necessary OP the drives (instructions are at: http://www.aerospike.com/docs/operations/plan/ssd/ssd_setup.html).

The Importance of Overprovisioning (OP)

Overprovisioning is space on an SSD that is reserved for use by the SSD controller (not the RAID controller).

Total OP (manufacturer + user) for Aerospike use should be at least 29%.

- Consumer drives – generally have 6%-8%
- Enterprise SATA – vary a lot, but many have up to 30%
- PCIe/NVMe – 22%+

This space is not user accessible and user OP will decrease the size of the SSD.

Running ACT Tests – Executing Tests

- Create a configuration file for the test by executing:

```
[act-3.0]$ python act_config_helper.py
Enter the number of devices you want to create config for: 1
Enter either raw device if over-provisioned using hdparm or partition if over-provisioned using
fdisk:
Enter device name #1 (e.g. /dev/sdb or /dev/sdb1): /dev/sdb
Change test duration default of 24 hours? (y/N): y
Enter the test duration in hours: 3
Use non-standard configuration? (y/N): n
"1x" load is  2000 reads per second and 1000 writes per second.
Enter the load factor (e.g. enter 1 for 1x test): 3
Do you want to save this config to a file? (y/N): y
Config file actconfig_3x_1d.txt successfully created.
```

- Select an appropriate starting point for your drive
  - Consumer SATA/Cloud – 3x
  - Enterprise SATA – 6x
  - PCIe/NVMe – 24x
- Run the ACT test and redirect output to a file:

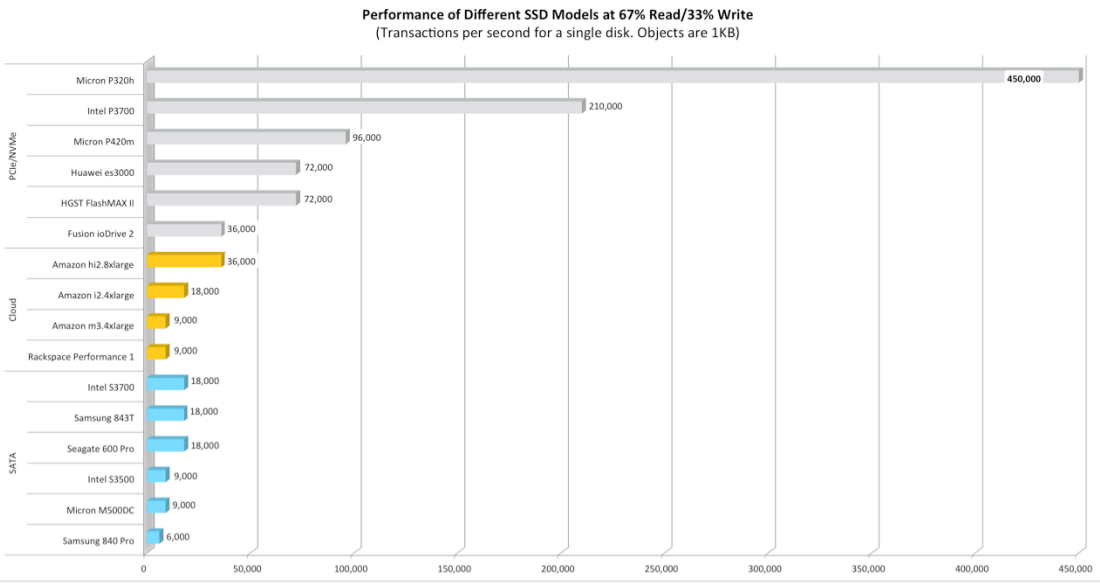
```
[act-3.0]$ act actconfig_3x_1d.txt > actconfig_3x_1d.log &
```

- Change the x factor and retest. The final test should be for 24 hours.

Analyzing the results:

- Passing means:
  - < 5% of transactions should take longer than 1 ms
  - < 1% of transactions should take longer than 8 ms
  - < 0.1% of transactions should take longer than 64 ms

SSD Certification Tests



Aerospike maintains a list of SSD performance numbers at:
http://www.aerospike.com/docs/operations/plan/ssd/ssd_certification.html

Summary

What we have covered

- Load test data into an Aerospike database using the Aerospike Java Benchmark tool.
- Run benchmark tests
- Testing SSDs with the ACT

You can use the benchmark tool by simply copying the files and running them from different servers.