Profiling Storage Without OpenStack

... or even with Openstack, just using any Linux VM with SSH enabled.

This is a quick overview of how to profile any system, be it bare metal, a VM or a container.

Prerequisites

1. Docker and docker-compose are installed
2. StorPerf is up and running
3. SSH access to target system(s) to be profiled

Set up

Assumption: system that will run Docker to host StorPerf is Ubuntu.

Install Docker and Docker-Compose

This step may be skipped if docker is already installed.

```
    sudo apt -y install docker docker-compose
sudo usermod -aG docker $USER
```

Log out and back in so your user is now part of the docker group.

Set Up StorPerf

StorPerf still has some code that expects to find OpenStack variables, even if OpenStack will not be used.

```
    mkdir storperf
cd storperf
cat << EOF > admin.rc
OS_AUTH_URL=dummy
OS_PROJECT_NAME=dummy
OS_USERNAME=dummy
OS_PASSWORD=dummy
EOF
```

Next, make a directory for storing metrics and ensure it has writable permissions for the Carbon process.

```
    mkdir carbon
sudo chown 33:33 carbon
```

Pull and start the StorPerf container.

```
    TAG=x86_64-latest ENV_FILE=./admin.rc CARBON_DIR=./carbon/ docker-compose up
```

StorPerf Swaager UI is now available at the IP address of this server, at port 5000.

Initialize Volumes to Test

This is an important step, especially as we do not know the implementation of the underlying storage device. For example, Ceph uses lazy block allocation, so reading from a Ceph volume that has no data yields incredible performance as no disk I/O will occur.

We need to decide: are we profiling a device or a filesystem? Initialization of files in an existing filesystem with a directory called "/storperf" looks as follows:

```
    {
  "target": "/storperf",
  "filesize": "2G",
  "nrfiles": 10,
  "numjobs": 10,
  "username": "user-with-sudo-access",
  "password": "password",
  "stack_name": null,
  "slave_addresses": [
    "10.0.0.69"
  ]
}
```

Submitting this to the ReST API will cause a new job to start which will create 100 files of 2GB each in the /storperf directory.

Profile the Disk

We use the same parameters as the initialize, but now we add the operations to perform. StorPerf can iterate over a matrix of block sizes, queue depths and IO patterns. Post the following to the /api/v2.0/jobs API:

```
    {
  "target": "/storperf",
  "filesize": "2G",
  "nrfiles": 10,
  "numjobs": 10,
  "username": "user-with-sudo-access",
  "password": "password",
  "stack_name": null,
  "slave_addresses": [
    "10.0.0.69"
  ],
  "block_sizes": "4k",
  "queue_depths": "8",
  "workloads": {
    "readwritemix": {
      "rw": "rw",
      "filesize": "2G",
      "nrfiles": "10",
      "numjobs": "10"
    }
  }
}
```

This will return a job id, which can be queried from the jobs API or viewed via the built-in graphite browser.

View the Results

First query the status of the job, using the id returned from the jobs API and type=status. This shows the status of a job that is still in progress.

blocked URL

This is a screenshot of Graphite showing the metrics for read and write latency.

blocked URL

Use the following graph targets to see average read and write latency for any job in Graphite:

```
    averageSeries(*.*.queue-depth.*.block-size.*.*--storperf.jobs.*.write.lat_ns.mean)
averageSeries(*.*.queue-depth.*.block-size.*.*--storperf.jobs.*.read.lat_ns.mean)
```

Once the job has finished, the status API will return the following:

```
    {
  "Status": "Completed"
}
```

From there we can get the summary metrics by specifying status=metrics. Note: bandwidth is reported in kilobytes (KB) per second. Bandwidth is the product of IOPS times block size.

```
    {
  "readwritemix.queue-depth.8.block-size.4k.read.bw": 450757.45454545453,
  "readwritemix.queue-depth.8.block-size.4k.read.iops": 112690.95571309091,
  "readwritemix.queue-depth.8.block-size.4k.read.lat_ns.mean": 599585.3004094727,
  "readwritemix.queue-depth.8.block-size.4k.write.bw": 451163.2727272727,
  "readwritemix.queue-depth.8.block-size.4k.write.iops": 112792.39080145456,
  "readwritemix.queue-depth.8.block-size.4k.write.lat_ns.mean": 109216.81024536365
}
```

Or, if desired, the full report can be fetched using the status=metadata tag. This includes the "criteria" tag, which will be PASS or FAIL. PASS means all tests were able to achieve steady state and report valid statistics, and FAIL is reported when at least one test run failed to achieve steady state,

The metadata also includes a report_data tag which provides the raw metrics for all runs. The series data, max, min and slope are all provided in this raw report for plotting or investigation. Each run will also report steady_state: true or false to indicate if the metrics are based on steady state and therefore can be counted as valid.

```
          "report_data": {
        "readwritemix.queue-depth.8.block-size.4k": {
          "bw": {
            "read": {
              "average": 450757.45454545453,
              "range": 18947,
              "series": [
                [
                  1,
                  464350
                ],
...
              "series_max": [
                [
                  1,
                  540908.9454545454
                ],
...
              "series_min": [
                [
                  1,
                  360605.9636363636
                ],
...
              "series_slope": [
                [
                  1,
                  457960.3181818182
                ],
...
              "slope": -1440.5727272727272,
              "steady_state": true
```