Reproducible and Scalable Experiments with SkyhookDM Ceph

Jayjeet Chakraborty
Carlos Maltzahn, Ivo Jimenez, Jeff LeFevre
UC Santa Cruz

CENTER FOR RESEARCH IN
OPEN SOURCE SOFTWARE

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

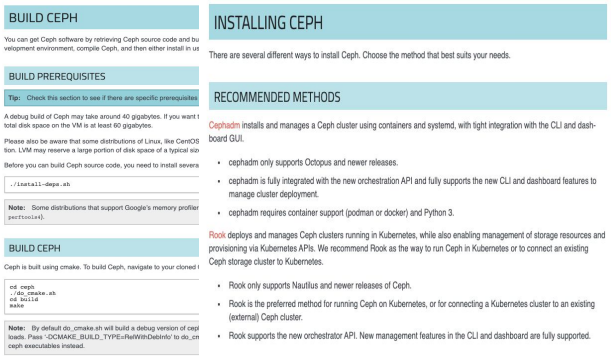
Generic Systems Experimentation Workflow

CloudLab

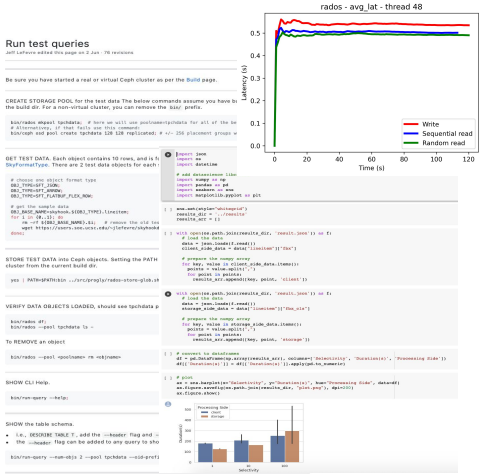
Google Cloud Platform

aws

Boot VMs or bare metal Nodes



Build, Deploy and Run experiments



Prepare plots and notebooks

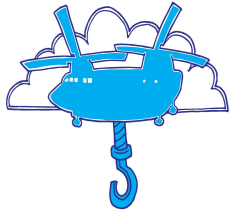
Doing manually is time consuming and error-prone !

High-Level Workflow for SkyhookDM Experiments

Spawn nodes and deploy k8s. Get the config.

Benchmark Network and Disk I/O .

Deploy Ceph and run Rados benchmarks.



Benchmark throughput by running queries on hundreds of GBs of tabular data.

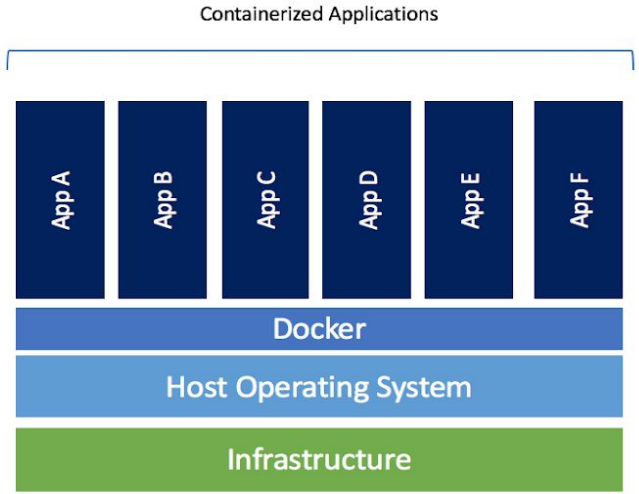
Study the Jupyter notebooks, Plots and Grafana snaps.

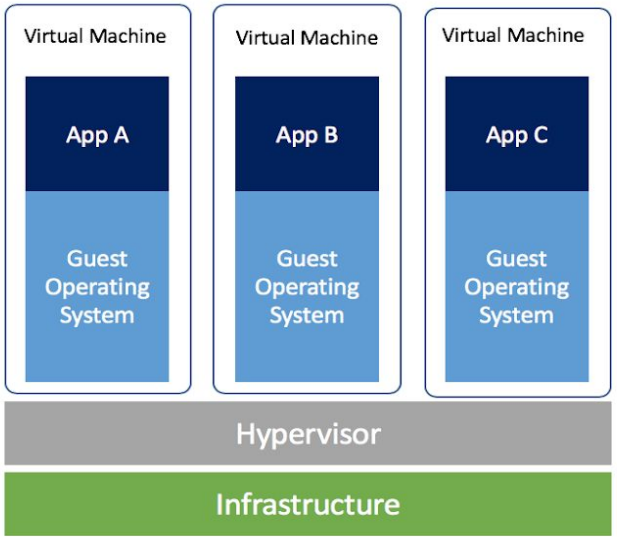
Should be platform independent and automated !

If done manually,

```
# manually build ceph everytime and update all osd, mon, etc.
$ git clone git://github.com/ceph/ceph
$ ./install-deps.sh
$ ./do_cmake.sh
$ cd build
$ make
$ sudo make install

# benchmarking the blockdevices in every node. take care of fio version and the parameters.
$ fio --filename=/dev/sdb --rw=randwrite --direct=1 --ioengine=libaio --bs=64k --numjobs=8 --runtime=120
$ fio --filename=/dev/sdb --rw=randwrite --direct=1 --ioengine=libaio --bs=128k --numjobs=8 --runtime=120

# running rados bench multiple times with different params.
$ ceph osd pool create testpool 32 32 replicated
$ rados bench --no-hints -t 1 -p testpool 120 seq
$ rados bench --no-hints -t 8 -p testpool 120 seq
$ rados bench --no-hints -t 32 -p testpool 120 seq

# plotting results
$ python plot_fio.py
$ pytnon plot_rados.py
```

Overview of Containers





- Less resource usage than VMs
- Platform independent and portable software
- Consistent operation across environments
- Greater efficiency

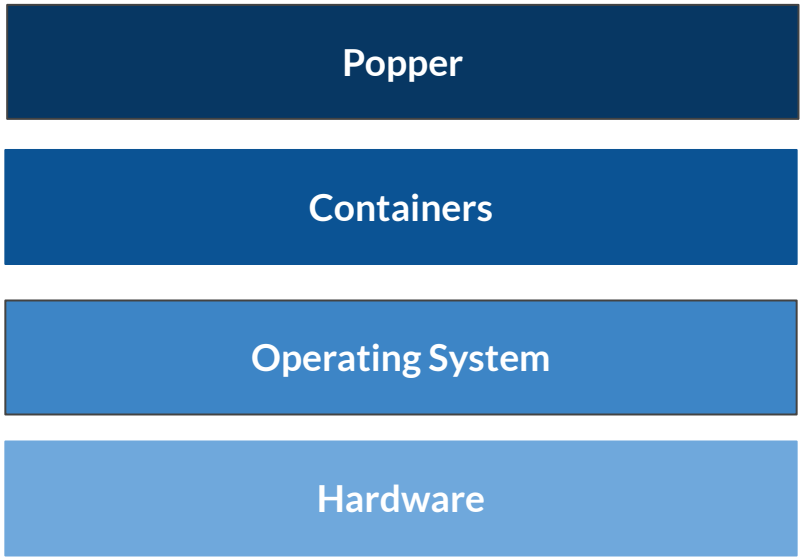
Containerizing Commands

```
$ docker run -e BLOCKDEVICE=sdb
             -e IODEPTH=32
             -v $PWD:/workspace
             --rm
             --entrypoint /bin/bash
             -w /workspace
             bitnami/kubectl:1.17.4
             ./run_benchmarks.sh
```

Solves platform dependency.

But still lacks automation !

Introducing Popper



Popper

docker

podman

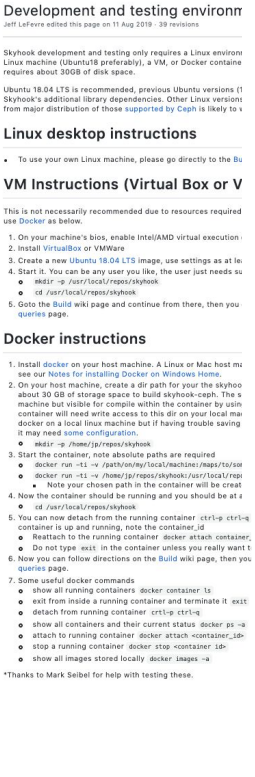
circleci


```
:~ # cd github-actions-demo/
:~/github-actions-demo # popper run
Enter the value for PHONY_SECRET:
FOO
[popper] cloning action repositories
[popper] - https://github.com/actions/bin@master
[install] docker pull node:11.6.0
[install] docker create node:11.6.0 npm install
[install] docker start
audited 195 packages in 4.857s
found 0 vulnerabilities

[test] docker pull node:11.6.0
[test] docker create node:11.6.0 npm test
[test] docker start

> github-actions-demo@1.0.0 test /Users/ivo/github-actions-demo
> mocha ./tests --recursive
```

```
steps:
- id: install lulesh
  uses: popperized/spack@master
  args: [spack, install, -j8, lulesh+mpi]

- id: delete existing jobs
  uses: popperized/bin/sh@master
  args: [rm, -fr, sweep/jobs]

- id: install sweepj2
  uses: popperized/python-actions@master
  args: [pip, install, sweepj2]

- id: generate sweep
  uses: jefftriplett/python-actions@master
  args: [
    "sweepj2",
    "--template", "./sweep/script.j2",
    "--space", "./sweep/space.yml",
    "--output", "./sweep/jobs/",
    "--make-executable"
  ]

- id: run sweep
  uses: popperized/spack@master
  args: [run-parts, ./sweep/jobs]
```

Slide borrowed from Ivo Jimenez

"*Popperizing*" the SkyhookDM Experimentation Workflow


Development and testing environm

Jeff LeFevre edited this page on 11 Aug 2019 · 39 revisions

Skyhook development and testing only requires a Linux environ
Linux machine (Ubuntu18 preferably), a VM, or Docker containe
requires about 30GB of disk space.

Ubuntu 18.04 LTS is recommended, previous Ubuntu versions (1
Skyhook's additional library dependencies. Other Linux versions
from major distribution of those supported by Ceph is likely to

Linux desktop instructions

- To use your own Linux machine, please go directly to the Bu

VM Instructions (Virtual Box or V

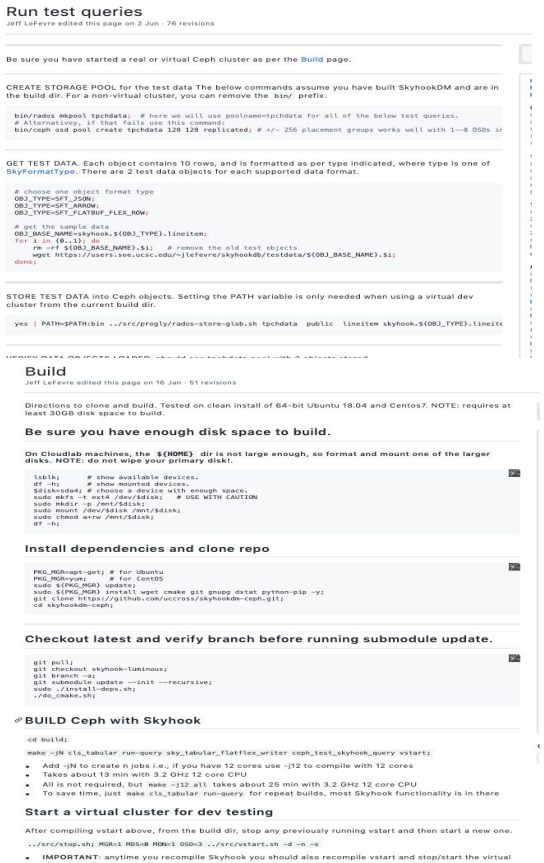
This is not necessarily recommended due to resources required
use Docker as below.

1. On your machine's bios, enable Intel/AMD virtual execution
2. Install VirtualBox or VMWare
3. Create a new Ubuntu 18.04 LTS image, use settings as at le
4. Start it. You can be any user you like, the user just needs su
   - `mkdir -p /usr/local/repos/skyhook`
   - `cd /usr/local/repos/skyhook`
5. Goto the Build wiki page and continue from there, then you
   queries page.

Docker instructions

1. Install docker on your host machine. A Linux or Mac host ma
   see our Notes for installing Docker on Windows Home.
2. On your host machine, create a dir path for your the skyhoo
   about 30 GB of storage space to build skyhook-ceph. The s
   machine but visible for compile within the container by usin
   container will need write access to this dir on your local ma
   docker on a local linux machine but if having trouble saving
   it may need some configuration.
   - `mkdir -p /home/jp/repos/skyhook`
3. Start the container, note absolute paths are required
   - `docker run -ti -v /path/on/my/local/machine:/maps/to/so`
   - `docker run -ti -v /home/jp/repos/skyhook:/usr/local/repo`
     - Note your chosen path in the container will be creat
4. Now the container should be running and you should be at a
   - `cd /usr/local/repos/skyhook`
5. You can now detach from the running container `ctrl-p ctrl-q`
   container is up and running, note the container_id
   - Reattach to the running container `docker attach container_`
   - Do not type `exit` in the container unless you really want t
6. Now you can follow directions on the Build wiki page, then you
   queries page.
7. Some useful docker commands
   - show all running containers `docker container ls`
   - exit from inside a running container and terminate it `exit`
   - detach from running container `ctrl-p ctrl-q`
   - show all containers and their current status `docker ps -a`
   - attach to running container `docker attach <container_id>`
   - stop a running container `docker stop <container id>`
   - show all images stored locally `docker images -a`

*Thanks to Mark Seibel for help with testing these.


Run test queries

Jeff LeFevre edited this page on 2 Jun · 76 revisions

Be sure you have started a real or virtual Ceph cluster as per the Build page.

CREATE STORAGE POOL for the test data The below commands assume you have built SkyhookDM and are in the build dir. For a non-virtual cluster, you can remove the `bin/` prefix.

```
bin/rados mkpool tpchdata;  # here we will use poolname=tpchdata for all of the below test queries.
# Alternatively, if that fails use this command:
bin/ceph osd pool create tpchdata 128 128 replicated; # +/- 256 placement groups works well with 1--8 OSDs ir
```

GET TEST DATA. Each object contains 10 rows, and is formatted as per type indicated, where type is one of SkyFormatType. There are 2 test data objects for each supported data format.

```
# choose one object format type
OBJ_TYPE=SFT_JSON;
OBJ_TYPE=SFT_ARROW;
OBJ_TYPE=SFT_FLATBUF_FLEX_ROW;

# get the sample data
OBJ_BASE_NAME=skyhook.${OBJ_TYPE}.lineitem;
for i in {0..1}; do
    rm -rf ${OBJ_BASE_NAME}.$i;   # remove the old test objects
    wget https://users.soe.ucsc.edu/~jlefevre/skyhookdb/testdata/${OBJ_BASE_NAME}.$i;
done;
```

STORE TEST DATA into Ceph objects. Setting the PATH variable is only needed when using a virtual dev cluster from the current build dir.

```
yes | PATH=$PATH:bin ../src/progly/rados-store-glob.sh tpchdata  public  lineitem skyhook.${OBJ_TYPE}.lineit
```

Build

Jeff LeFevre edited this page on 16 Jan · 61 revisions

Directions to clone and build. Tested on clean install of 64-bit Ubuntu 18.04 and Centos7. NOTE: requires at least 30GB disk space to build.

Be sure you have enough disk space to build.

On Cloudlab machines, the `${HOME}` dir is not large enough, so format and mount one of the larger disks. NOTE: do not wipe your primary disk!.

```
lsblk;      # show available devices.
df -h;      # show mounted devices.
$disk=sda4; # choose a device with enough space.
sudo mkfs -t ext4 /dev/$disk;   # USE WITH CAUTION
sudo mkdir -p /mnt/$disk;
sudo mount /dev/$disk /mnt/$disk;
sudo chmod a+rw /mnt/$disk;
df -h;
```

Install dependencies and clone repo

```
PKG_MGR=apt-get; # for Ubuntu
PKG_MGR=yum;     # for CentOS
sudo ${PKG_MGR} update;
sudo ${PKG_MGR} install wget cmake git gnupg dstat python-pip -y;
git clone https://github.com/uccross/skyhookdm-ceph.git;
cd skyhookdm-ceph;
```

Checkout latest and verify branch before running submodule update.

```
git pull;
git checkout skyhook-luminous;
git branch -a;
git submodule update --init --recursive;
sudo ./install-deps.sh;
./do_cmake.sh;
```

BUILD Ceph with Skyhook

```
cd build;
make -jN cls_tabular run-query sky_tabular_flatflex_writer ceph_test_skyhook_query vstart;
```

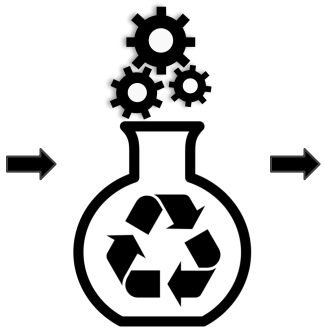
- Add -jN to create n jobs i.e., if you have 12 cores use -j12 to compile with 12 cores
- Takes about 13 min with 3.2 GHz 12 core CPU
- All is not required, but `make -j12 all` takes about 25 min with 3.2 GHz 12 core CPU
- To save time, just `make cls_tabular run-query` for repeat builds, most Skyhook functionality is in there

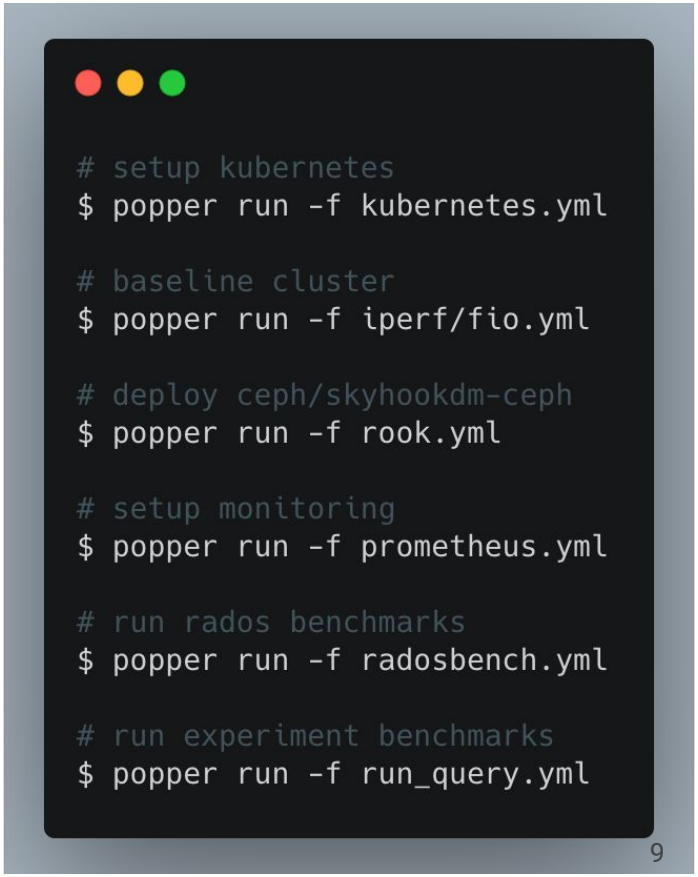
Start a virtual cluster for dev testing

After compiling vstart above, from the build dir, stop any previously running vstart and then start a new one.

```
../src/stop.sh; MGR=1 MDS=0 MON=1 OSD=3 ../src/vstart.sh -d -n -x
```

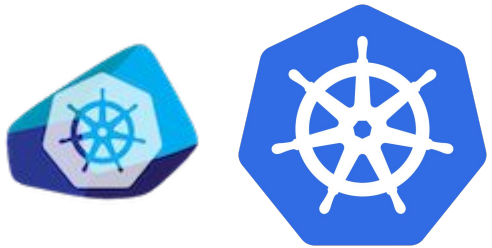
- **IMPORTANT**: anytime you recompile Skyhook you should also recompile vstart and stop/start the virtual cluster again.





Implementation Highlights

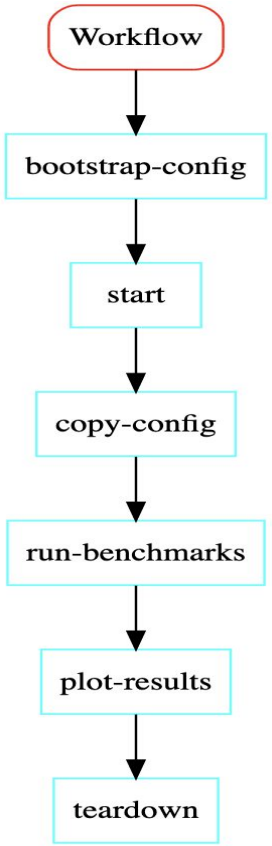
- Highly configurable and scalable workflows with parameter sweeps.
- Monitoring infrastructure with Prometheus + Grafana.
- Everything in Kubernetes. Rook.io, kube-prometheus, kubespray, kubestone, etc.



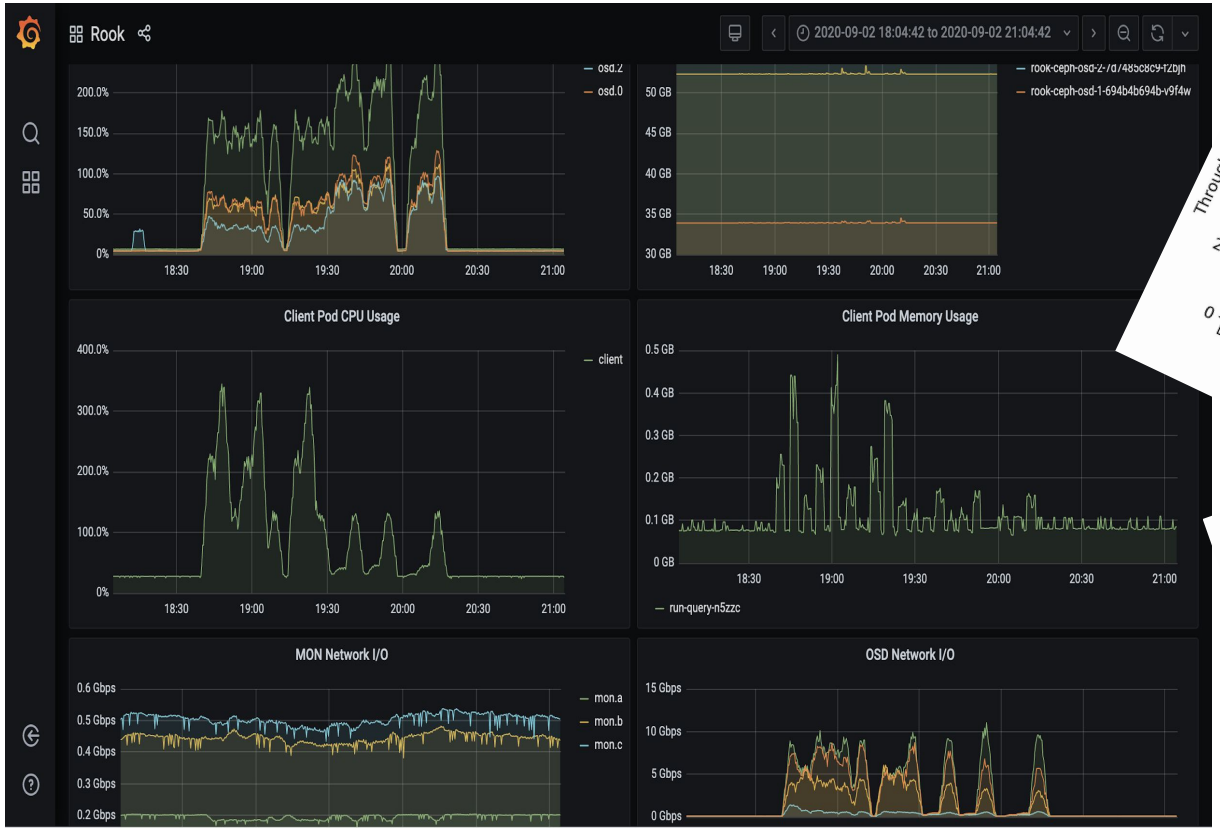
Workflows for Every High-level Step

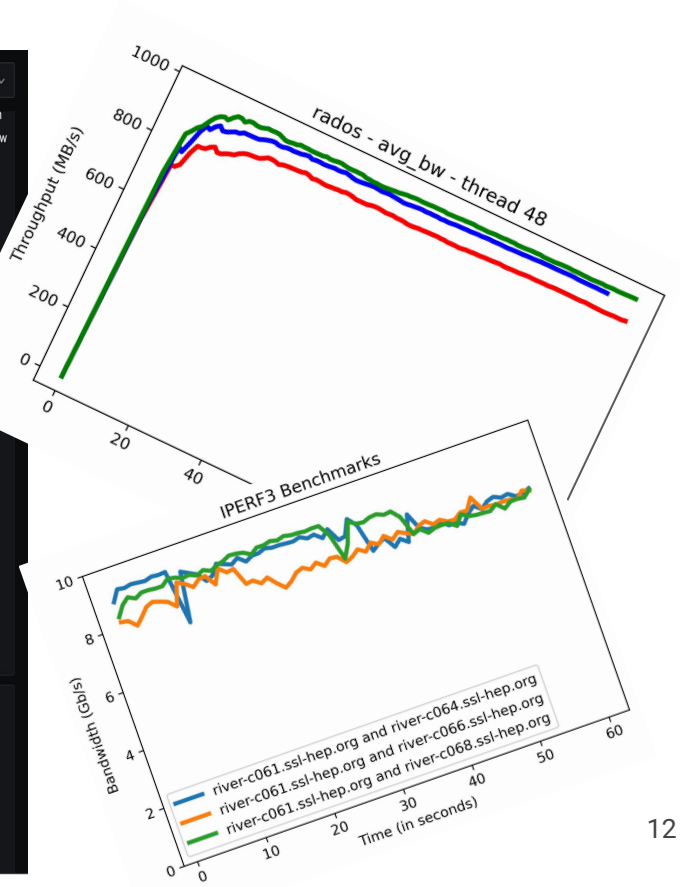
```yaml
options:
  env:
    KUBECONFIG: ./kubeconfig/config
    BLOCKDEVICES: $_BLOCKDEVICES
    BLOCKSIZE: '4k 32k 128k 1m 4m'
    PV_SIZE: 4Gi
    IO_DEPTH: '32'
    DURATION: '120'
    IO_ENGINE: libaio
    NUM_JOBS: '8'
    MODES: 'read write randread randwrite'
    HOSTNAME: $_HOSTNAME

steps:
- id: bootstrap-config
  uses: docker://biowdl/pyyaml:3.13-py37-slim
  runs: [python]
  args: [./kubestone_fio/scripts/bootstrap.py]

- id: start
  uses: docker://bitnami/kubectl:1.17.4
  runs: [bash, -euc]
  args:
  - |
    kubectl apply -n kubestone -f ./kubestone_fio/pv.yaml
    kubectl apply -n kubestone -f ./kubestone_fio/pvc.yaml
    kubectl apply -n kubestone -f ./kubestone_fio/job.yaml

- id: run-benchmarks
  uses: docker://bitnami/kubectl:1.17.4
  runs: [./kubestone_fio/scripts/run_benchmarks.sh]

- id: download-results
  uses: docker://bitnami/kubectl:1.17.4
  runs: [./kubestone_fio/scripts/download_results.sh]

- id: plot-results
  uses: docker://jupyter/datascience-notebook:python-3.8.5
  runs: [jupyter]
  args: ["nbconvert", "--execute", "--to=notebook", "./kubestone_fio/notebook/plot.ipyr
  options:
    ports:
      8888/tcp: 8888

- id: teardown
  uses: docker://bitnami/kubectl:1.17.4
  runs: [bash, -euc]
  args:
  - |
    kubectl delete -n kubestone --ignore-not-found -f ./kubestone_fio/job.yaml
    kubectl delete -n kubestone --ignore-not-found -f ./kubestone_fio/pvc.yaml
    kubectl delete -n kubestone --ignore-not-found -f ./kubestone_fio/pv.yaml
```

```yaml
options:
  env:
    KUBECONFIG: ./kubeconfig/config
    NAMESPACE: kubestone
    WRITE_DURATION: '120'
    READ_DURATION: '120'
    THREADS: '1 8 32'
    OBJECT_SIZE: '10M'
    PG_SIZE: '128'
    POOL_NAME: 'testbench'
    POOL_TYPE: 'replicated'
    REPLICATION_DISABLED: '1'
    CLIENT: $_CLIENT

steps:
- id: bootstrap-config
  uses: docker://biowdl/pyyaml:3.13-py37-slim
  runs: [python]
  args: [./radosbench/scripts/bootstrap.py]

- id: start
  uses: docker://bitnami/kubectl:1.17.4
  runs: [bash, -euc]
  args:
  - |
    kubectl apply -n "$NAMESPACE" -f ./radosbench/deployment.yaml

- id: copy-config
  uses: docker://bitnami/kubectl:1.17.4
  runs: [./radosbench/scripts/copy_config.sh]

- id: run-benchmarks
  uses: docker://bitnami/kubectl:1.17.4
  runs: [./radosbench/scripts/run_benchmarks.sh]

- id: plot-results
  uses: docker://jupyter/datascience-notebook:python-3.8.5
  runs: [jupyter]
  args: ["nbconvert", "--execute", "--to=notebook", "./radosbench/notebook/plot.ipynb"]
  options:
    ports:
      8888/tcp: 8888

- id: teardown
  uses: docker://bitnami/kubectl:1.17.4
  runs: [bash, -euc]
  args:
  - |
    kubectl delete -n "$NAMESPACE" --ignore-not-found -f ./radosbench/deployment.yaml
```



Monitoring with Prometheus and Grafana

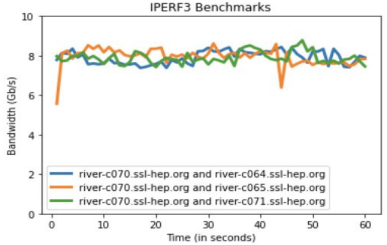




Jupyter Notebooks for further Exploration

In [1]:
```
import os
import json
import matplotlib.pyplot as plt
```

In [2]:
```
results_dir = '../results'
files = os.listdir(results_dir)

# iterate over each result file and plot the results
for file in files:
    if not file.endswith('.json'): continue
    with open(os.path.join(results_dir, file)) as f:
        results = json.load(f)

    seconds = []
    bandwidth = []

    for run in results["intervals"]:
        seconds.append(float(run["sum"]["end"]))
        bandwidth.append(float(run["sum"]["bits_per_second"])/(10

    plt.plot(seconds, bandwidth, markersize=10, linewidth=3.0, la


# pin the range between 0 and 10
plt.ylim(0.0, 10.0)
plt.xlabel('Time (in seconds)')
plt.ylabel('Bandwidth (Gb/s)')
plt.title('IPERF3 Benchmarks')
plt.legend()
plt.savefig(os.path.join(results_dir, './iperf-benchmarks.png'),
plt.show()
```



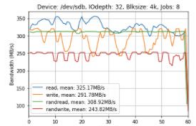
In [34]:
```
import os
import json
import numpy as np
import matplotlib.pyplot as plt

from collections import defaultdict
from statistics import mean
```

In [35]:
```
results_dir = '../results'
blockdevices = os.environ["BLOCKDEVICES"].split(" ")
modes = os.environ["MODES"].split(" ")
iodepths = os.environ["IO_DEPTH"].split(" ")
jobsizes = os.environ["NUM_JOBS"].split(" ")
blocksizes = os.environ["BLOCKSIZE"].split(" ")
duration = os.environ["DURATION"]
```

In [36]:
```
def get_mean(lst):
    return "%.2f" % mean(lst)

def get_moving_avg(x, N):
    return np.convolve(x, np.ones((N,))/N)[(N-1):]
```

Bandwidth Plots

In [37]:
```
for blkdev in blockdevices:
    for iodepth in iodepths:
        for blksize in blocksizes:
            for jobsize in jobsizes:
                base_dir = os.path.join(f"../results/{blkdev}/{blksize}")

                for mode in modes:
                    data = defaultdict(dict)
                    for jobnum in range(1, int(jobsize) + 1):

                        time = []
                        value = []

                        with open(os.path.join(base_dir, f'{mode}-iodepth-{iodepth}-numjobs-{jobsize}_bw.{jobnum}.log'), "r")
as f:

                            row = f.readlines()
                            for datapoint in row:
                                time.append(int(datapoint.split(",")[0])/1000)
                                value.append(int(datapoint.split(",")[1])/1000)

                        data[jobnum]['time'] = time
                        data[jobnum]['value'] = value

                    min_time_length = 1000000000
                    min_val_length = 1000000000

                    for jobnum in range(1, int(jobsize) + 1):
                        min_time_length = min(min_time_length, len(data[jobnum]['time']))
                        min_val_length = min(min_val_length, len(data[jobnum]['value']))

                    cum_time = []
                    cum_val = []

                    for idx in range(0, min_time_length):

                        sum_time = 0
                        for jobnum in range(1, int(jobsize) + 1):
                            sum_time += data[jobnum]['time'][idx]

                        sum_time /= int(jobsize)
                        cum_time.append(sum_time)

                        sum_val = 0
                        for jobnum in range(1, int(jobsize) + 1):
                            sum_val += data[jobnum]['value'][idx]

                        cum_val.append(sum_val)

                    moving_avg_val = list(get_moving_avg(cum_val, 3))
                    mean_val = get_mean(moving_avg_val)
                    plt.xlim(0, int(duration))
                    plt.plot(cum_time, moving_avg_val, markersize=10, linewidth=1.5, label=f"{mode}, mean: {mean_val}MB/s")

                plt.xlabel("Time (s)")
                plt.ylabel("Bandwidth (MB/s)")
                plt.title(f"Device: /dev/{blkdev}, IOdepth: {iodepth}, Blksize: {blksize}, Jobs: {jobsize}")
                plt.grid()
                plt.legend()
                plt.savefig(os.path.join(results_dir, f"./fio-bw-{blkdev}-{iodepth}-{blksize}-{jobsize}.png"), dpi=300, bbox_
inches='tight')
                plt.show()
```




```
import json
import os
import datetime

# add datascience libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

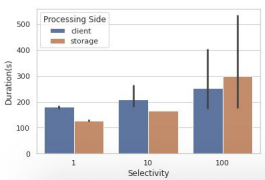
```
sns.set(style="whitegrid")
results_dir = '../results'
results_arr = []
```

```
with open(os.path.join(results_dir, 'result.json')) as f:
  # load the data
  data = json.loads(f.read())
  client_side_data = data["lineitem"]["fbx"]

  # prepare the numpy array
  for key, value in client_side_data.items():
    points = value.split(",")
    for point in points:
      results_arr.append([key, point, 'client'])
```

```
with open(os.path.join(results_dir, 'result.json')) as f:
  # load the data
  data = json.loads(f.read())
  storage_side_data = data["lineitem"]["fbx_cls"]

  # prepare the numpy array
  for key, value in storage_side_data.items():
    points = value.split(",")
    for point in points:
      results_arr.append([key, point, 'storage'])
```

```
# convert to dataframes
df = pd.DataFrame(np.array(results_arr), columns=['Selectivity', 'Duration(s)', 'Processing Side'])
df[['Duration(s)']] = df[['Duration(s)']].apply(pd.to_numeric)
```

```
# plot
ax = sns.barplot(x="Selectivity", y="Duration(s)", hue="Processing Side", data=df)
ax.figure.savefig(os.path.join(results_dir, "plot.png"), dpi=200)
ax.figure.show()
```



Case Study: Benchmarking SkyhookDM on River SSL

- Baselined the River SSL Kubernetes cluster and performed performance benchmarks for SkyhookDM Ceph.
- Discovered bottleneck in Network I/O for 10GbE links.
- Discovered unbalanced CPU usage in OSDs due to unbalanced PGs.

New avenues for further investigation !

Future Work

- Try capturing and creating workflows for other categories of Ceph benchmarks like CephFS and Ceph RBD benchmarks.

- Create experiment/benchmark workflows for other popular systems e.g. Key-value stores like RocksDB, databases like PostgreSQL, etc.

Thank you !

Visit https://github.com/uccross/skyhookdm-workflows/

Questions ?

jchakra1@ucsc.edu
https://twitter.com/heyjc25