Timeframe: 7 weeks of work

Core requirements:

* Output of the project should be both DEV and easy self-service setup for customer facing
* focus on modeling scenarios and displaying the outcome - which apps are running and which are not.
* then you can simulate monte-carlo server failures given a scenario, and aggregate scheduling results - how many apps stayed, hoped, evicted.
* a tool to build backhanded directly for topology from configure file. (especially for capacity with more dimensions than just CPU, Memory, Disk)
* Execute treadmill scheduler completely without master dependency (zookeeper, ldap or other master service in place)
* Admin command which is able to submit task/monitor/ app, add/remove server in the the scheduler
* readable CLI ascii chart/metrics to display the status of app/server, show the app placement by scheduler.

Predict scheduler behavior in situation:

1) when new dimension for capacity is introduced (e.g io ops, io throughput)

2) when a group of servers are down

3) manifest the container launch/eviction when new container in allocation is launched

4) simulate when cell size is much larger than we have at the moment.

For monte-carlo simulation, i think we should provide a tool to easily launch monte carlo test repeatedly. e.g.

1 server down, how many containers are evicted

2 lot of containers launched, how many container are in pending status, how is the fragmentation of the cell capacity.

Input:

we need to have handle input files to define the topology of the cell and application.

* Topology input

hosts.yml topology,yml

hosts.yml:

---

small\_host:

cpu: 400

memory: 8G

disk: 16G

iops: 2000

traits: xxx

large\_host:

cpu: 800

memory: 16G

iops: 4000

disk: 32g

huge\_host:

cpu: 1600

memory: 48G

iops: 4000

disk: 100g

topology.yml:

---

building:xx

rack:xx1:

- number: 50

type: { "$ref": "hosts.yml#/small\_host" }

valid\_util:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- number: 20

type: { "$ref": "hosts.yml#/large\_host" }

rack: xx2

- number: 10

type: { "$ref": "hosts.yml#/huge\_host" }

This will create following host:

* 50 hosts named small\_host-0001 - small\_host-0050 in rack xx1
* 20 hosts named large\_host-0001 - large\_host-0020 in rack xx1
* 10 hosts named huge\_host-0001 - huge\_host-0010 in rack xx2
* Application input

app.yml allocation.yml task.yml

app.yml

---

proid1.app1:

manifest:

cpu: 100

memory: 1G

disk: 1G

iops: 100

affinity: proid1

affinity\_limits: { ..}

services:

* name: bash

restart: {interval: 60, limit: 5}

command: /bin/bash

lease: 1d

lifetime:

min: 300

max: 600

proid2.app2:

manifest

cpu: 100

memory: 1G

disk: 1G

iops: 50

services: [ … ]

lifetime:

min: 0

allocation.yml

---

alloc:sub/prod:

cpu: 20

memory: 5G

disk: 5G

iops: 4000

assignments:

* pattern: proid1.app1

priority: 1

alloc2:sub2/dev

cpu: 20

memory: 5G

disk: 5G

iops: 3000

assignments:

* pattern: proid1.app1

priority: 1

task.yml

---

- action: allocation\_configure

allocation: { “$ref”: “allocation.yml#/alloc:sub/prod” }

- action: host\_up

hosts:

* large\_host-0001:0010
* small\_host-0001:0020

- action: app\_start

apps:

* app: { “$ref”: “app.yml#/proid1.app1” }

count: 20

* app: { “$ref”: “app.yml#/proid2.app2” }

count: 10

- action: host\_down

hosts:

* large\_host-0002
* large\_host-0005:0008

- action: allocation\_configure

allocation: { “$ref”: “allocation.yml#/alloc:sub/prod” }

delta:

cpu: 2000

assignments:

* pattern: proid1.app1

priority: 1

* pattern: proid2.app2

priority: 2

- action: sleep

interval: 300

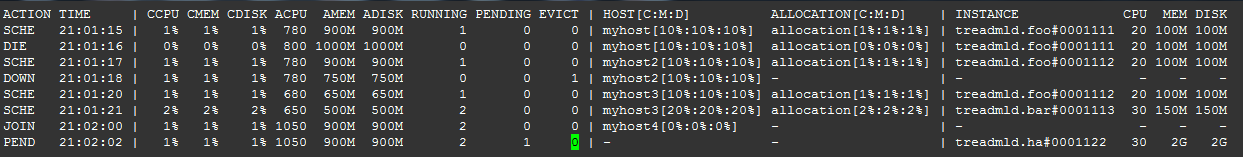
- action: host\_up

hosts:

* large\_host-0007
* large\_host-0008

Output :

Log Steps regarding scheduling process may look like this.



Monte carlo simulation result:

Chart 1: Application -- check pending/evicted

proid1:app1:

pending: 10

evicted: 5

proid2:app2:

pending: 5

evicted: 3

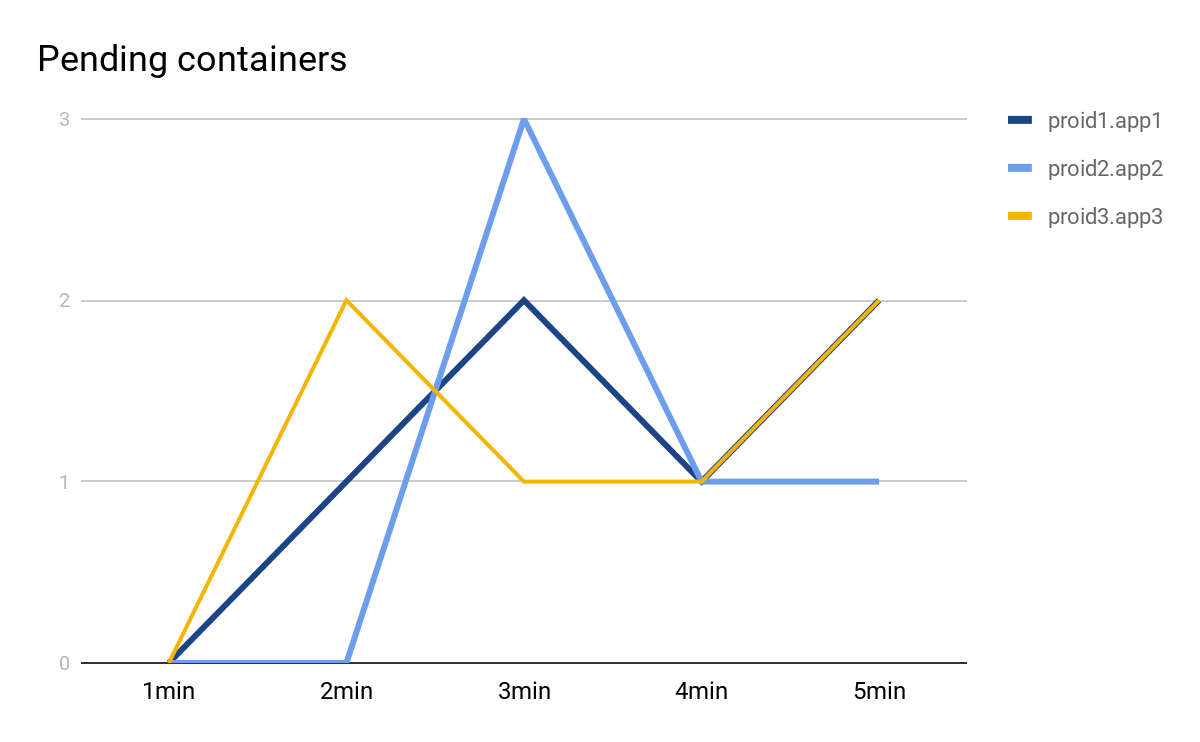


Chart 2: Cell -- see fragmentation

free cpu: xxx

free memory: xxx

free disk: xxx

Chart 3: Allocation -- check applications out of allocation

Allocation: alloc:sub

proid1.app1: 10

proid2.app2: 10

Project Components

1. scheduler to use fs-backend (already have)
2. script to setup topology based on yaml file definition
3. script to submit tasks to scheduler
4. a daemon to manage container lifecycle [optional]
5. a daemon to collect fs-backend data based on the time
6. a script to generate chart based on data from #5 in monte carlo test