import random

from oslo\_config import cfg

from nova.compute import rpcapi as compute\_rpcapi

from nova import exception

from nova.openstack.common import log as logging

from nova.openstack.common.gettextutils import \_

from nova.scheduler import driver

CONF = cfg.CONF

CONF.import\_opt('compute\_topic', 'nova.compute.rpcapi')

LOG = logging.getLogger(\_\_name\_\_)

class PriorityScheduler(driver.Scheduler):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(PriorityScheduler, self).\_\_init\_\_(\*args, \*\*kwargs)

self.compute\_rpcapi = compute\_rpcapi.ComputeAPI()

def \_filter\_hosts(self, request\_spec, hosts, filter\_properties,

hostname\_prefix):

hosts = [host for host in hosts if host.startswith(hostname\_prefix)]

return hosts

def \_schedule(self, context, topic, request\_spec, filter\_properties):

elevated = context.elevated()

hosts = self.hosts\_up(elevated, topic)

if not hosts:

msg = \_("Is the appropriate service running?")

raise exception.NoValidHost(reason=msg)

remote\_ip = context.remote\_address

if remote\_ip.startswith('10.1'):

hostname\_prefix = 'doc'

elif remote\_ip.startswith('10.2'):

hostname\_prefix = 'ops'

else:

hostname\_prefix = 'dev'

hosts = self.\_filter\_hosts(request\_spec, hosts, filter\_properties,

hostname\_prefix)

if not hosts:

msg = \_("Could not find another compute")

raise exception.NoValidHost(reason=msg)

host = random.choice(hosts)

LOG.debug("Request from %(remote\_ip)s scheduled to %(host)s" % locals())

return host

def select\_destinations(self, context, request\_spec, filter\_properties):

num\_instances = request\_spec['num\_instances'

dests = []

for i in range(num\_instances):

host = self.\_schedule(context, CONF.compute\_topic,

request\_spec, filter\_properties)

host\_state = dict(host=host, nodename=None, limits=None)

dests.append(host\_state)

if len(dests) < num\_instances:

raise exception.NoValidHost(reason='')

return dests

def schedule\_run\_instance(self, context, request\_spec,

admin\_password, injected\_files,

requested\_networks, is\_first\_time,

filter\_properties, legacy\_bdm\_in\_spec):

instance\_uuids = request\_spec.get('instance\_uuids')

for num, instance\_uuid in enumerate(instance\_uuids):

request\_spec['instance\_properties']['launch\_index'] = num

try:

host = self.\_schedule(context, CONF.compute\_topic,

request\_spec, filter\_properties)

updated\_instance = driver.instance\_update\_db(context,

instance\_uuid)

self.compute\_rpcapi.run\_instance(context,

instance=updated\_instance, host=host,

requested\_networks=requested\_networks,

injected\_files=injected\_files,

admin\_password=admin\_password,

is\_first\_time=is\_first\_time,

request\_spec=request\_spec,

filter\_properties=filter\_properties,

legacy\_bdm\_in\_spec=legacy\_bdm\_in\_spec)

except Exception as ex:

driver.handle\_schedule\_error(context, ex, instance\_uuid,

request\_spec)

package enhancement;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import base.\*;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Simulation of a containers running cloudlets

\* in a Cloud environment

\*

\* ENHANCEMENT

\*

\* @author Nathan Kong

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class enhancement {

private static int time;

private static Host host;

private static ArrayList<VM> VMs;

private static ArrayList<Container> containers;

private static ArrayList<Request> completedRequests;

public static void main(String[] args) {

time = 1;

int numOfHosts = 1;

int hostBW = 1000000; //Rounded number from CloudSim

int numOfVMs = 1;

int vmBW = 3000; //Rounded number from CloudSim

int numOfContainers = 3;

int numOfRequests = 75;

/\*\*initialize architecture\*/

//create host

host = new Host(numOfHosts, hostBW);

System.out.println("Host " + Host.getId() + " created");

//create VMs

createVMs(numOfVMs, vmBW);

//create Containers

createContainers(numOfContainers);

//Setup - Connection of Objects

setup();

/\*\*Simulate project\*/

//create requests and adds them to containers

schedule(numOfRequests);

//run simulation

run(numOfRequests);

//print

printOutput();

System.out.println("\n\*\*\*\*\* Enhancement Simulation Complete \*\*\*\*\*");

}

private static void createVMs(int num, int bw){

VMs = new ArrayList<VM>();

printBreak();

for (int i = 1; i <= num; i++){

VMs.add(new VM(i, bw));

}

for (VM m: VMs){

System.out.println("VM " + m.getId() + " created ");

}

}

private static void createContainers(int num) {

containers = new ArrayList<Container>();

printBreak();

//random generated data

int[] priority = {0, 1, 2, 3, 3 };

for (int i = 1; i <= num; i++){

containers.add( new Container(i, priority[i]) );

}

for (Container c: containers){

System.out.println("Container " + c.getId() + " has a priority of " + c.getPriority() );

}

}

private static void setup() {

//VMs look to attach to hosts

printBreak();

for (VM vm: VMs){

vm.setHost(host);

host.addVm(vm);

}

//containers look to attach to VMs

printBreak();

int v = 0;

int numOfVms = VMs.size();

for (Container con: containers){

if (v == numOfVms){ v = 0;}

VM vm = VMs.get(v++);

//System.out.println(vm.getId());

vm.addContainer(con);

con.setVm(vm);

}

}

public static void printBreak(){

System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

private static void schedule(int numOfRequests){

//create request

ArrayList<Request>requests = createRequests(numOfRequests);

for (Request r: requests){

containers.get( r.getContainerId()-1 ).addRequest(r);

}

}

private static ArrayList<Request> createRequests(int num) {

ArrayList<Request> requests = new ArrayList<Request>();

printBreak();

//random generated data

int[] bw = {1343, 2426, 2207, 947, 2201, 1222, 1085, 1547, 1098,

1112, 2187, 1391, 1260, 1687, 1233, 560, 1334, 1329, 2007,

1947, 1394, 1737, 1761, 762, 655, 1509, 1380, 1432, 1035,

921, 1595, 1456, 1265, 1174, 1922, 1470, 1418, 1957, 1409,

2067, 1116, 2275, 727, 1511, 2120, 1266, 832, 1860, 1175,

1635, 1139, 1507, 1191, 1408, 835, 1782, 741, 2083, 1943,

2378, 1836, 1798, 2262, 2418, 908, 770, 2192, 1384, 1297,

1020, 2108, 1524, 2229, 1462, 2301, 2240, 2206, 1098, 1470,

1837, 930, 2031, 2210, 1738, 1096, 920, 1666, 2194, 1208, 869,

1138, 1024, 2401, 1052, 700, 2405, 1864, 501, 1906, 686, 1513};

/\*int[] requestTime = {29, 10, 57, 34, 30, 23, 12, 33, 56, 60, 43, 32,

27, 30, 11, 57, 20, 16, 30, 47, 49, 9, 48, 54, 38, 23, 56,

52, 4, 23, 37, 35, 32, 39, 20, 2, 46, 19, 31, 32, 39, 57, 51,

12, 11, 23, 30, 8, 3, 37, 49, 17, 32, 24, 54, 46, 7, 9, 23,

28, 20, 29, 59, 19, 7, 50, 2, 22, 52, 42, 2, 53, 34, 38, 25,

10, 54, 42, 40, 45, 27, 56, 1, 51, 40, 22, 41, 50, 37, 48, 59,

11, 36, 11, 19, 42, 49, 53, 51, 50, 39};

\*/

int[] containerID = {0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,

2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,

3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,

4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4};

//create container

for (int i = 1; i <= num; i++){

requests.add( new Request(i, bw[i], 4, containerID[i]) );

}

for (Request r: requests){

System.out.println("Requests: " + r.getId() + " requires " + r.getBw() + " bw and will take " + r.getTime() + " seconds on container " + r.getContainerId() );

}

return requests;

}

private static void run(int numOfRequests){

printBreak();

boolean run = true;

ArrayList<Request> queue = new ArrayList<Request>();

//ArrayList<Container> running = new ArrayList<Container>();

//add to list

queue = scheduleList();

completedRequests = new ArrayList<Request>();

do {

//run based off of the list

runContainers(queue);

run = checkRequests(numOfRequests);

increment();

}while(run);

}

private static ArrayList<Request> scheduleList() {

ArrayList<Request> queue = new ArrayList<Request>();

Container c1 = containers.get(0);

Container c2 = containers.get(1);

Container c3 = containers.get(2);

int s1 = c1.getRequestSize();

int s2 = c2.getRequestSize();

int s3 = c3.getRequestSize();

while (s1 + s2 + s3 != 0) {

if (s1 > 0){

queue.add(c1.RemoveFirstRequest());

s1 = c1.getRequestSize();

}

if (s1 > 0){

queue.add(c1.RemoveFirstRequest());

s1 = c1.getRequestSize();

}

if (s2 > 0){

queue.add(c2.RemoveFirstRequest());

s2 = c2.getRequestSize();

}

if (s1 > 0){

queue.add(c1.RemoveFirstRequest());

s1 = c1.getRequestSize();

}

if (s2 > 0){

queue.add(c2.RemoveFirstRequest());

s2 = c2.getRequestSize();

}

if (s3 > 0){

queue.add(c3.RemoveFirstRequest());

s3 = c3.getRequestSize();

}

}

return queue;

}

private static void runContainers(ArrayList<Request> req){

Boolean c1 = true;

Boolean c2 = true;

Boolean c3 = true;

//System.out.println(completedRequests.size());

int con = 0;

int cnt = 0;

for (Request r: req){

System.out.println(cnt++);

if (r.getContainerId() == 1 && c1 && r.getStatus() != Status.completed){

con = processRequest(r);

}else if (r.getContainerId() == 2 && c2 && r.getStatus() != Status.completed){

con = processRequest(r);

}else if (r.getContainerId() == 3 && c3 && r.getStatus() != Status.completed){

con = processRequest(r);

}

if(con == 1){c1=false;}

if(con == 2){c2=false;}

if(con == 3){c3=false;}

if(!c1 && !c2 && !c3){

break;

}

}

}

private static int processRequest(Request r){

VM vm = VMs.get(0);

if (vm.getBW() >= r.getBw() && r.getStatus() != Status.completed){

//subtract time from VM bandwidth

vm.subBW(r.getBw());

//if the request has not been started

if(r.getStatus() == Status.waiting){

r.setStartTime(time);

r.setStatus(Status.running);

}

if(r.getTime() == 0 && r.getStatus() != Status.completed){

r.setFinishTime(time);

completedRequests.add( r );

//req.remove(r);

r.setStatus(Status.completed);

}else{

r.subTime(1);

}

}

return r.getContainerId();

}

//checks to make sure all requests have been completed

private static boolean checkRequests(int numOfRequests){

int size = completedRequests.size();

if(size == numOfRequests){

return false;

}

/\*else{

System.out.println(size);

}

\*/

return true;

}

private static void increment(){

//increment global time

time++;

//reset the bandwidth for each VM

for (VM vm: VMs){

vm.resetBW();

}

}

private static void printOutput(){

System.out.println("Request ID\tContainer ID\tStart Time\tFinish Time\tBandwidth");

for (Request r: completedRequests){

r.printResults();

}

}

}