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):

s = []

priority high = 1

priority med = 2

priority low = 3

for i in range(len(list)-1):

med = len(list) / 2

l1 = list[:mid]

l2 = list[mid:]

l2.reverse()

if(i % 2 == 1):

s = s + [ zip(l1, l2) ]

else:

s = s + [ zip(l2, l1) ]

list.insert(1, list.pop())

return s

for round in create\_schedule(high):

for match in round:

print match[0] + " - " + match[1]

print

for round in create\_schedule(med):

for match in round:

print match[0] + " - " + match[1]

print

for round in create\_schedule(low):

for match in round:

print match[0] + " - " + match[1]

print

for round in create\_schedule(high+med+low):

for match in round:

print match[0] + " - " + match[1]

print

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

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)