Priority based scheduling for Lease management in cloud computing

Pragnesh K. Patel[1] Prof. H. B. Jethva[2]
[1] Research Scholar, ITNS
[2] Associate Professor, Computer Engineering Department
[2]L. D. College of Engineering, Ahmedabad, Gujarat, India

Abstract – Cloud Computing is the next generation Architecture of the most IT enterprise. It made possible, the application to run without the burden of local hardware and software. So, resource allocation is key issue in cloud computing.For better resource allocation cloud service providers use lease managers. Lease managers depend on their default scheduling algorithms and their efficiency of granting and revoking leases is dependent on efficiency of scheduler they have. Haizea, a lease manager written in python, provides three types of leases: Immediate, Best Effort (BE) and Advance Reservation (AR). AR leases are most privileged leases with "AR-preempts everything" policy, since they can preempt & suspend other BE leases when demanded. But BE or Immediate can't preempt other BE or Immediate. BE lease have to wait in queue. This work proposes priority based resource allocation algorithm to solve these problem. Experimental results of the proposed algorithm successfully demonstrate that we can get less waiting time for BE leases than default algorithm.

Key words: cloud computing, lease management, resource allocation, scheduling, Haizea.

I. INTRODUCTION

Clouds can be used to provide on-demand capacity as a utility [1].Since a cloud typically comprises a large amount of virtual and physical servers, in the order of hundreds or thousands, efficiently managing this virtual infrastructure becomes a major concern[2].

Several solutions, such as VMWare Virtual Center, Platform Orchestrator, or Enomalism, have emerged to manage virtual infrastructures, providing a centralized control platform for the automatic deployment and monitoring of virtual machines (VMs) in resource pools[6].

However, these solutions provide simple VM placement and load balancing policies.In particular, existing clouds use an immediate provisioning model, where virtualized resources are allocated at the time they are requested, without the possibility of requesting resources at a specific future time and, at most, being placed in a simple first-come-first-serve queue when no resources are available[6].

However, service provisioning clouds have requirements that cannot be supported within this model, such as [6]:

- Resource requests that are subject to non-trivial policies
- Capacity reservations at specific times to meet peak capacity requirements
- Variable resource usage throughout a VM's lifetime
- Dynamic renegotiation of resources allocated to VMs

Additionally, smaller clouds with limited resources, where not all requests may be satisfiable immediately for lack of resources, could benefit from more complex VM placement strategies supporting queues, priorities, and advance reservations. So, we need capacity provisioning using *resource leases.*

A lease is "a negotiated and renegotiable agreement between a resource provider and a resource consumer, where the former agrees to make a set of resources available to the latter, based on a set of lease terms presented by the resource consumer" [6]. For this kind of capacity provisioning we need lease manager.

II. HAIZEA

Haizea is an open source lease management architecture developed by Borja Sotomayor that OpenNebula can use as a scheduling backend. Haizea uses leases as a fundamental resource provisioning abstraction, and implements those leases as virtual machines, taking into account the overhead of using virtual machines when scheduling leases. Using OpenNebula with Haizea allows resource providers to lease their resources, using potentially complex lease terms, instead of only allowing users to request VMs that must start immediately [4].

In Haizea, the lease terms include the hardware resources, software environments, and the availability period during which the hardware and software resources must be available. Haizea maps leases to VMs, scheduling the deployment overhead of starting those VMs separately and leveraging backfilling algorithms and the suspend/resume/migrate capability of VMs to schedule the leases more efficiently[3].

Haizea can be used in three modes: OpenNebula mode, unattended simulation mode, and interactive simulation mode.

In unattended simulation mode, Haizea takes a list of lease requests (specified in a trace file) and a configuration file specifying simulation and scheduling options (such as the characteristics of the hardware to simulate), and processes them in "simulated time".

In interactive simulation mode, enactment actions are simulated, but Haizea runs in "real time". This means that, instead of having to provide a list of lease requests beforehand, you can use Haizea's command-line interface to request leases interactively and query the status of Haizea's schedule.

To use Haizea as an OpenNebula scheduling backend, a resource provider simply has to run Haizea instead of OpenNebula's default Capacity Manager[2]. To request a lease, instead of a VM that must start immediately, users

have to specify an HAIZEA option in the OpenNebula request. Haizea currently works with OpenNebula by polling it for new requests, and sending OpenNebula enactment commands and polling for any changes in state. Haizea supports the following types of leases:

- Leases requiring a single VM or groups of VMs that must run in parallel.
- Best-effort leases, which will wait in a queue until resources become available.
- Advance reservation leases, which must start at a specific time.
- Immediate leases, which must start right now, or not at all.

So, Haizea aims to support resource leasing along these three dimensions. Best-effort lease Resources are provisioned as soon as they are available.

Advance reservation-style leases (or "AR leases") Resources are provisioned during a strictlydefined time period (e.g., from 2pm to 4pm).Immediate leases Resources must be provisioned right now, or not at all. Haizea uses several scheduling algorithms internally to determine what resources to allocate to a lease. Backfilling algorithm to use for scheduling in Haizea with option like aggressive (at most 1 reservation in the Future), conservative (unlimited reservations in the future, intermediate N reservations in the future).

Figure 1,2 &3. Shows pseudo code for Greedy Mapping without Preemption, Best effort Algorithm and Advance Reservation Algorithm used in Haizea Lease manager.

Algorithm: Greedy Mapping without Preemption

Input: *A lease l, a time t , duration d*
Output: A mapping, if one can be found, nodes[*l*] P, starting at time t and ending at t + d
Begin
map? empty dictionary
P'? sort(**P**)
{Sorted according to host select ion policy}
Cur_node? First node in nodes[*l*]
for all p P'(while there are still nodes in nodes[*l*] left t o map) do
p_done? false
while not p_done do
if cur_node fits in p from t to t + d
then
map cur_node? p
cur_node? Next node in nodes[*l*]
else
p_done? False
end if
end while
end for
If (all nodes in nodes[*l*] have been mapped)
then
return map
end if
return *l*
end if
end

Fig-1. Greedy Mapping without Preemption

Algorithm: Best effort Algorithm

Input: A lease *l, a Boolean allow_follow*
Output: A lease *l* **Begin**
m? map(*l*,now,duration[*l*])
If (m!=□) **then**
vmrr? new r eservation
start[*vmrr*] ? n ow, end[*vmrr*]
? now+duration [*l*]
res[*vmrr*] ? …
Add *vmrr* to r eservations[*l*] and to slot table.
state[*l*] ? Schedule
else if m= □and n ot allow_future **then**
state[*l*] ? Queued
else
changepoints ? Time in the future in the which there is ch ange in the slot table
for all cp □changepoints **do**
m? map(l,cp,duration[*l*])
if m!= □□**then** break **end if end for**
if m=□□**then** state[*l*] ? Queued
else
vmrr? new reservation
start[*vmrr*] ? cp, end[*vmrr*] ? cp+duration[*l*]
res[*vmrr*] ? …, Add *vmrr* to reservations[*l*] and to slot table.
state[*l*] ? Schedule
end if
end if
return *l*
end

Fig-2. Best effort Algorithm

Algorithm : Advance Reservation

Input: A lease *l*
Output: A lease *l*
Begin
m? map(*l*,start[*l*],duration[*l*])
If (m=□□) **then**
state[*l*]? Rejected
else
vmrr? new reservation
start[*vmrr*] ? start[*l*]
end[*vmrr*] ? start[*l*]+duration[*l*]
r es[*vmrr*] ? …
Add *vmrr* to reservations[*l*] and to slot table.
state[*l*] ? Schedule
end if
end

Fig-3. Advance Reservation.

If AR lease don't get free slot on reserved time, it will be rejected. If we don't want this kind of rejections, we can use AR-To-BE conversion algorithm proposed by Vivek Shrivastava and D.S. Bhilare [5]. AR-to-BE Conversion algorithm prevents AR lease rejection by converting AR

lease to a BE lease, and thus queues lease to be rejected to run in future time frames [5]. But immediate leases will be rejected if they don't get resources at time. Further more if any BE lease has more priority than current running BE, than resources should be given to this BE lease by pre-empting current BE lease. This same should be done with immediate lease rather than rejecting it if it has more priority than current lease. So, to solve these issues we proposing following priority based algorithm.

Algorithm: Priority Based resource allocation

Input: A lease *l, priority of lease*
Output: A lease *l*
Begin
if scheduling lease type== AR **then:**
if no AR lease is running on required time slot & resources are free **then:**
allocate resources to AR
else
reject AR
else
if scheduling lease type == Immediate lease **then:**
if resources required are available at the time **then:**
allocate resources to Immediate lease
else
if priority of Immediate lease >priority of current lease **then:**
preempt current lease and allocate resources to Immediate lease
else
reject Immediate lease and print message
else
if scheduling lease type == BE lease **then:**
if priority of BE lease > priority of current lease **then:**
preempt current lease and allocate resources to BE lease
else
queue BE lease and set state of lease to queue
else
print message Invalid lease type
end

Fig-4. Priority based resource allocation algorithm

III. IMPLEMENTATION OF PROPOSED ALGORITHM.

We can customize scheduling policy in haizea by writing own policies by creating python modules.

The following decisions are supported by pluggable module [3]:

A. *Lease admission*

It takes into account that this decision takes place before Haizea determines if the request is even feasible. However, being accepted doesn't guarantee the lease will get resources.

B. *Lease preemptability*

Not all leases are created equal and, if the scheduler determines that a lease request can only be satisfied by preempting other leases, it may have to determine what leases are better candidates for preemption.

C. *Host selection*

When the scheduler has a choice of several physical hosts on which to deploy VMs, some might be preferable than others.

We have written a custom module to implement my proposed priority based resource allocation algorithm which uses 2nd kind of (i.e. lease preemptabilty) decision making.

IV. EXPERIMENTS AND RESULTS

The haizea simulator will read trace requests from a trace file. The location of this trace file is specified in the configuration file, in the [tracefile] section.

The format of this file is LWF (Lease Workload Format), an XML format which is particular to Haizea.

sample data set with 3 leases:

```
<?xml version="1.0" encoding="utf-8"?>
<lease-workload name="sample">
<description>
A simple trace where an BE lease preempts a
BE lease that is already running.
</description>
<lease-requests>
<!-- The lease requests are initially commented out -->
<!-- First lease request -->
<lease-request arrival="00:00:00">
<lease preemptible="true">
<nodes>
<node-set numnodes="1">
<res type="CPU" amount="100"/>
<res type="Memory" amount="1024"/>
</node-set>
</nodes>
<start></start>
<duration time="01:00:00"/>
<software>
<disk-image id="foobar.img" size="1024"/>
</software>
</lease>
</lease-request>
<!-- Second lease request -->
<lease-request arrival="00:15:00">
<lease preemptible="true">
<nodes>
<node-set numnodes="4">
<res type="CPU" amount="100"/>
<res type="Memory" amount="1024"/>
</node-set>
</nodes>
<start>
</start>
<duration time="00:30:00"/>
<software>
<disk-image id="foobar.img" size="1024"/>
</software>
</lease>
</lease-request>
```

```
<!-- Third lease request -->
<lease-request arrival="00:50:00">
<lease preemptible="true">
<nodes>
<node-set numnodes="2">
<res type="CPU" amount="100"/>
<res type="Memory" amount="1024"/>
</node-set>
</nodes>
<start>
</start>
<duration time="00:30:00"/>
<software>
<disk-image id="foobar.img" size="1024"/>
</software>
</lease>
</lease-request>
</lease-requests>
</lease-workload>
```

Fig -5. Sample trace file.

Haizea is software written in python. It has set of variables which take values from trace file. So, it is very difficult to add new tab for priority in trace file. It requires many changes in many modules of haizea. So, we used number of nodes as priority to test our algorithm. If number of nodes is high,lease has high priority. We have used same data set, i.e. set of 7 trace files containing all BE leases to compare haizea's default BE algorithm with our proposed algorithm. Results are shown in following table**:**

Total Leases	No. of nodes	Total BR comp-leted	Total waiting time without priority (all-best-effort)	Total waiting time with priority (all-best-effort)
3	21	3	6664	6300
6	42	6	13864	13500
9	63	9	21064	20700
12	84	12	28264	27900
15	105	15	35464	35100
18	126	18	42664	42300
21	147	21	49864	49500

In table waiting time is taken from all-best-effort probe from results.dat file. The all-best-effort probe collects the waiting time of each best-effort lease.

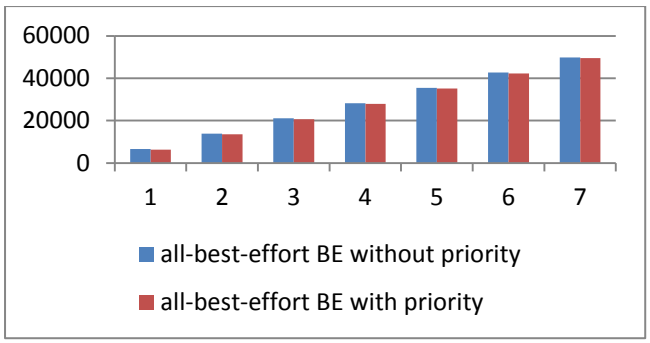

Fig-6. Graph Showing Waiting Time of Priority Based Algorithm over Existing Algorithm.

V. CONCLUSION

To reduce request waiting time and immediate resource allocation depending on priority, and increase resource utilization on cloud provider side, priority based resource allocation is necessary. This work also show how priority based resource algorithm can be implemented as pluggable module in haizea. This work also saves immediate leases from being rejected if they have more priority than current preemptible lease.

REFERENCES

[1] Peter Mell ,Timothy Grance, The NIST Definition of Cloud Computing, National Institute of Standards and Technology Special Publication, 800-145

[2] Neha Tyagi, Rajesh Pathak, Negotiation for Resource Allocation for Multiple processes Infrastructure as a Service Cloud, (IJAER) 2011, Vol. No. 2, Issue No. I, July

[3] Sotomayor B. - Haizea: http://haizea.cs.uchicago.edu/whatis.html, Computation Institute, University of Chicago.

[4] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, Ian Foster, Virtual Infrastructure Management in Private and Hybrid Clouds, IEEE Internet Computing, vol. 13, no. 5, pp. 14-22, Sep./Oct. 2009.

[5] Vivek Shrivastava, D.S. Bhilare, Algorithms to Improve Resource Utilization and Request Acceptance Rate in IaaS Cloud Scheduling, Int. J. Advanced Networking and Applications ,Volume: 03, Issue: 05, Pages: 1367-1374 (2012)

[6] Sotomayor B. et. al. , Resource Leasing and the Art of Suspending Virtual Machines, HPCC'09), 2009, pp. 59--68.

[7] Borja Sotomayor., Provisioning Computational Resources Using Virtual Machines and Leases., University of chicago, Dept. of Computer Science. Defended July 7, 2010