

2D Scaling in Cloud Computing

Aruna Mailavaram¹, B.Padmaja Rani²

¹Madanapalle Institute of Technology & Science, A.P.

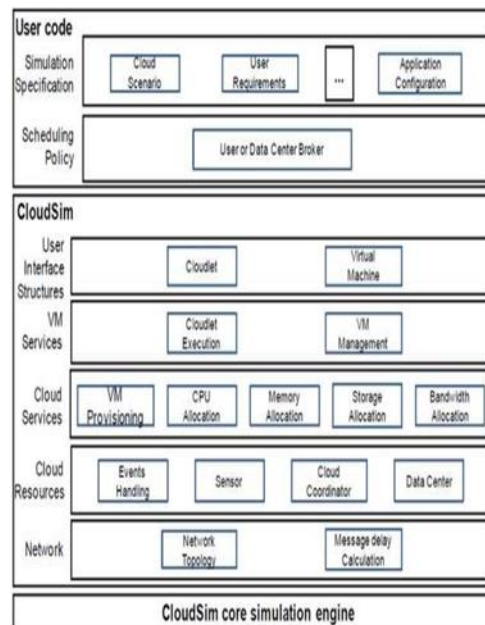
¹JNTUHCEH, Hyderabad.

*Corresponding author's E-mail: Aruna Mailavaram

Article History	Abstract
Received: 08 June 2023 Revised: 21 Sept 2023 Accepted: 10 Dec 2023	<p><i>Two-Dimensional scaling is increasing the resources of the cloud as per the requirements. In this paper we are discussing about the scaling in both vertical and horizontal as to fulfill the needs of the customers. Vertical scaling is increasing the specifications of existing infrastructure by adding or replacing CPU, HDD or other components, whereas horizontal scaling is adding new resources to the existing pool of resources. Increasing the resources in two-dimensional is known as Diagonal Scaling.</i></p>
CC License CC-BY-NC-SA 4.0	Keywords: Deadline, Scaling, Cloud Computing, Performance, Vertical Scaling And Horizontal Scaling

1. Introduction

Cloud Computing [6] is a huge source of resources where different organizations utilize them for their work. Since all the resources are globally spread through the internet, remotely many services can be accessed. Since the customers are not limited, day by day customers are increasing cloud has to be designed with no issues. But still we have few issues in cloud computing like security, scalability, performance, latency etc. To design a scalable[1] cloud system to improve performance and making all the tasks to reach within deadline time we are proposing a method, where all the tasks are reaching the deadlines.



The above figure shows the architecture of the cloud, how the cloudlets[3] are being processed using cloud sim tool, this architecture provides the environment of cloud. The scheduling policies[4] will be mounted on scheduling policy in the architecture[5]. Here in this paper we are designing horizontal and vertical scaling algorithms and compared with each other. We are trying to improve the performance of the system as well as the tasks should reach the deadline. So two parameters we are focusing on. In my next upcoming paper, we will focus on the combination of both vertical and horizontal scaling algorithms.

Design and Implementation

Here we are implementing both horizontal and vertical scaling[12] separately and compared with the previous algorithms and found better results in horizontal scaling[13]. Initially we are taking a data set of VMs, task lengths and deadlines as input and given to broker function for accessing all the tasks are arranged in ascending order and VMs in descending order. Allotting tasks to the VMs for processing if any leftover tasks are there we are implementing horizontal scaling[12].

Horizontal Scaling: Arrange all the VMs in descending order, and all tasks in ascending order associated with their deadlines according to the task length[9]. Here in this algorithm we are checking overloaded condition and under loaded condition of each VM. We are utilizing 80 percentage of each MV capacity in order to avoid overloaded condition. For example if we consider 720 MIPS capacity VM then we are utilizing 576 MIPS of its capacity. And we are seeing that tasks not reaching the deadline is less at the same time improving the performance of the system.

Since this is horizontal scaling we can add resources to the pool of existing resources[8]. In the given data set we have taken ten VMs and arranged them in descending order. Two VMs with high capacity are kept in spare for further use to handle high length tasks. So we are starting from 720 MIPS Virtual Machine, and assigning the tasks until it reaches 80 percentage of VM capacity[3], then task is being hand over to next VM, until all VMs get engaged. Check for the left over tasks, assign the tasks to the VMs which we kept in spare for further use to handle high length tasks. If further any tasks are still left over then we have to implement horizontal scaling if the percentage of unprocessed tasks[10] are less than 10 percentage then increase VMs with 10 percentage, if else the unprocessed tasks are between 10 to 25 percentage increase the VMs by 25 percentage else calculate Average Resource Utilization Ratio (ARUR).

Problem Formulation

Capacity of VM is $CVM = p \cdot q + BW\{j,k\}$

Where p is the processing speed and q is the number of CPU and BW is the bandwidth between j and k virtual machines.

Now we find out the load at the data center i.e., $DC = \sum_{i=1}^m CVM_i$

Load information at the VM can be found by $LVM = \text{number of task} * \sum \text{task length}/p \cdot q$.

So, total load at Data Center is $TL_{VM} = \sum_{i=1}^m L_{VM}$.

Expected Execution Time at Virtual Machine is calculated as

$EET = \text{Task Length}/p \cdot q$.

Makespan Time = $\text{Max}(\sum_{i=1}^m ET)$.

Algorithm for Horizontal Scaling :

1. Generate n tasks $T_1, T_2, T_3, \dots, T_n$.
2. Sort the tasks in ascending order.
3. Generate m number of virtual machines and sort them in descending order.
4. Consider VMs from $m \cdot 0.2$ to $m-1$, higher capacity VMs are kept in UVM list in increasing order to handle high length tasks.
5. For loop for T_i from 0 to $n-1$ and for all virtual machines R_j belongs to $m-1$.
6. If $(\text{Tasks}[i] \neq \emptyset)$ then
7. For loop $\text{Tasks}[i] \rightarrow \text{VM}[j]$, $i++$ until
Total_time $TT[j] < (\text{VM}[j] \cdot 0.8)$
8. Then allot tasks to next VM i.e., $\text{VM}[j+1]$
9. Repeat step 7 and 8 until all VMs are engaged.
10. If $(\text{Tasks}[i] == \emptyset)$ then calculate makespan
11. Else If
While $(\text{UVM} \neq 0)$
 $\text{Tasks}[i+1] \rightarrow \text{VM}[j]$ until $\text{UVM} == 0$

12. Else calculate the left over tasks
 - i. If $(UT > 0 \ \&\& \ UT < m * 0.1)$ increase VMs by 10 percentage
 - ii. Else If $(UT > m * 0.1)$ Increase VMs by 20 percentage
 - iii. Else $UT == 0$ calculate ARUR.
13. Repeat step 11 and step 12 until all tasks are completed.

Vertical Scaling : In vertical scaling[14] initially we are using the same technique as we did in horizontal scaling to allot the tasks to the VMs. Once the left over tasks are available, then allot them to the VM which are in idle state, but these left over tasks has to wait until previous tasks allotted to the virtual machine is completed and then allot task to the VM so here we have to increase the VM capacity based on the deadline achievement.

Increasing the power of the system is vertical scaling unlike horizontal scaling the system power in terms of cpu, RAM[11] we can increase in vertical scaling, many public clouds can also allow to increase the capacity of the system power (example AWS). But the Virtual machine has to stop running we can increase its power and start running with the new power to handle the task to complete within the deadline.

In this scaling technique we are increasing the capacity of the VM based on the deadline, a new VM is generated to reach the task within the deadline and still increase the capacity of the VM by 25 percentage to overcome the overloaded condition and round off the capacity of the VM.

Problem Formulation

$$\text{New_VM} = \text{Task}[i] / \text{Deadline}[i]$$

To avoid overload condition we are increasing the VM capacity by 25 percentage.

$$\text{VM}_{\text{Increase}} = \text{New_VM} * 25 / 100$$

$$\text{New_VM} = \text{New_VM} + \text{VM}_{\text{Increase}}$$

Algorithm : Vertical Scaling

Input : VMs, Tasks and Deadline.

Output : All tasks are being processed.

1. Generate n tasks $T_1, T_2, T_3, \dots, T_n$.
2. Sort the tasks in ascending order.
3. Generate m number of virtual machines and sort them in descending order.
4. Consider VMs from $m * 0.2$ to $m - 1$, higher capacity VMs are kept in UVM list in increasing order to handle high length tasks.
5. For loop for T_i from 0 to $n - 1$ and for all virtual machines R_j belongs to $m - 1$.
6. If $(\text{Tasks}[i] \neq \emptyset)$ then
7. For loop $\text{Tasks}[i] \rightarrow \text{VM}[j]$, $i++$ until
 $\text{Total_time } TT[j] < (\text{VM}[j] * 0.8)$
8. Then allot tasks to next VM i.e., $\text{VM}[j+1]$
9. Repeat step 7 and 8 until all VMs are engaged.
10. If $(\text{Tasks}[i] == \emptyset)$ then calculate makespan
11. Else If
 While $(\text{UVM} \neq 0)$
 $\text{Tasks}[i+1] \rightarrow \text{VM}[j]$ until $\text{UVM} == 0$
12. Else calculate the number of left over tasks.
13. If $UT \neq \emptyset$ then
14. Generate new VM based on the deadline for each task as $\text{New_VM} = \text{tasks}[i] / \text{Deadline}[i]$
15. To avoid overload condition increase the new VM by 25 percentage.

Synthetic Data :

Considering a dataset for both the algorithms.

Task ID	Task Length(MI)	Deadline of Task {ms}	VMs (MIPS)
17	98049	1050	760
18	107218	300	740
19	147281	1150	720
15	153285	800	700
7	182561	2000	680
16	205633	600	660
9	215744	1300	640
13	222784	550	620
10	253197	1250	600
14	253745	1000	580
4	325750	420	560
3	325800	410	
5	325970	700	
6	333911	660	
11	334013	450	
2	339760	920	
12	344630	400	
0	381771	400	
1	392397	745	
8	396156	300	

After implementing Horizontal and vertical scaling we are getting the result as follows:

2. Results and Discussion

Horizontal Scaling :

Task ID	Task Length	Deadline of Task	Execution Time
17	98049	1050	136
18	107218	300	284
19	147281	1150	488
15	153285	800	218
7	182561	2000	478
16	205633	600	302
9	215744	1300	326
13	222784	550	348
10	253197	1250	408
14	253745	1000	422
4	325750	420	561
3	325800	410	577
5	325970	700	440
6	333911	660	439
11	334013	450	428
2	339760	920	424
12	344630	400	420
0	381771	400	454
1	392397	745	456
8	396156	300	450

There are five tasks which are not meeting the deadline[7] Task ID 4,3,12,0,8.

Vertical Scaling :

Task ID	Task Length	Deadline of Task	Execution Time
17	98049	1050	136
18	107218	300	284
19	147281	1150	488
15	153285	800	218
7	182561	2000	478
16	205633	600	302
9	215744	1300	326
13	222784	550	348
10	253197	1250	408
14	253745	1000	422
4	325750	420	420
3	325800	410	410
5	325970	700	756
6	333911	660	978
11	334013	450	946
2	339760	920	878
12	344630	400	848
0	381771	400	863
1	392397	745	1038
8	396156	300	1038

In vertical scaling there are total 8 tasks which are not reaching the deadlines[2] they are Task ID 5,6,11,2,12,0,1,8.

ARUR (Average Resource Utilization Ratio) :

Average Resource Utilization Ration(ARUR) for Vertical Scaling

$$ARUR = (\text{Mean Time} / \text{Make Span}) * 100$$

Mean Time = \sum Total Time taken by resource VMj to finish all the job / No. of resources.

Parameters in Horizontal Scaling :

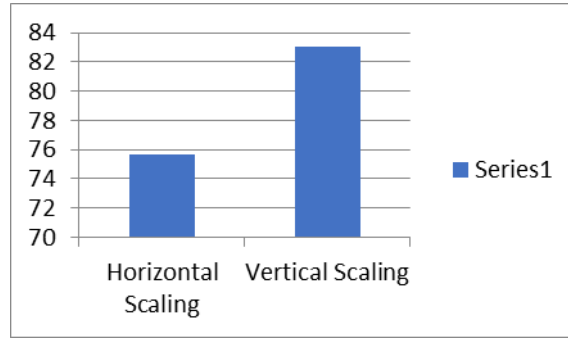
Parameters	Values
No. of Overloaded Machines	2
No. of Under loaded Machines	0
Tasks not reaching deadline	5
Makespan	577 ms
ARUR	75.65

Parameters in Vertical Scaling :

Parameters	Values
No. of Overloaded Machines	8
No. of Under loaded Machines	0
Tasks not reaching deadline	8
Make span	1038 ms
ARUR	83.01

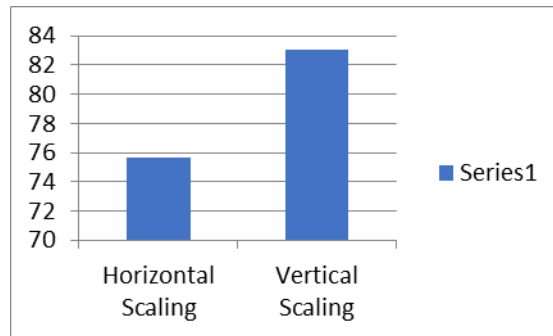
Comparison(ms) :

Horizontal Scaling	577
Vertical Scaling	1038



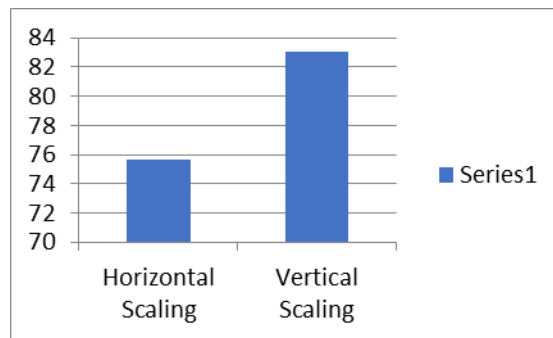
Number of Overloaded machines :

Horizontal Scaling	2
Vertical Scaling	8



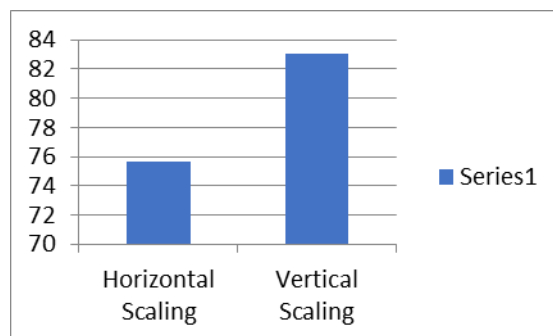
Number of tasks not reaching deadline :

Horizontal Scaling	5
Vertical Scaling	8



ARUR :

Horizontal Scaling	75.65
Vertical Scaling	83.01



3. Conclusion

We have discussed about both vertical and horizontal scaling in cloud computing separately, scale out is horizontal scaling on x-axis and scale up vertical scaling on y-axis. In my next coming papers we will combine horizontal and vertical scaling. But according to this paper we concluded that in horizontal scaling we were able to achieve more number of tasks meeting the deadline when compared with vertical scaling.

References:

1. Dynamic load balancing algorithm for balancing the workload among virtual machine in cloud computing Mohit Kumara,*,S.C.Sharma. <https://doi.org/10.1016/j.procs.2017.09.141>.
2. Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment. <https://doi.org/10.1016/j.compeleceng.2017.11.018>.
3. Elastic and flexible deadline constraint load Balancing algorithm for Cloud Computing. <https://doi.org/10.1016/j.procs.2017.12.092>
4. Dynamic load balancing algorithm to minimize the make span time and utilize the resources effectively in cloud environment. <https://doi.org/10.1080/1206212X.2017.1404823>.
5. Load balancing algorithm to minimize the make span time in cloud environment. ISSN1746-7233,England,UK World Journal of Modelling and Simulation. Vol.14(2018)No.4,pp.276-288.
6. A comprehensive survey for scheduling techniques in cloud computing. <https://doi.org/10.1016/j.jnca.2019.06.006>.
7. Dynamic Auto-scaling and Scheduling of Deadline Constrained Service Workloads on IaaS Clouds. 10.1016/j.jss.2016.05.011
8. Optimized Task Scheduling and Resource Allocation on Cloud Computing Environment Using Improved Differential Evolution Algorithm. <http://dx.doi.org/10.1016/j.cor.2013.06.012>
9. Heuristic-based load-balancing algorithm for IaaS cloud. <https://doi.org/10.1016/j.future.2017.10.035>.
10. Load balancing in cloud computing: A big picture. <https://doi.org/10.1016/j.jksuci.2018.01.003>
11. A Priority Based Job Scheduling Algorithm Using IBA and EASY Algorithm for Cloud Metascheduler <https://www.researchgate.net/publication/312814322>
12. Horizontal and Vertical Scaling of Container-Based Applications Using Reinforcement Learning. <https://doi.org/10.1109/CLOUD.2019.00061>
13. Vertical Scaling of Virtual Machines In Cloud Environment. <https://doi.org/10.1109/RTEICT52294.2021.9573715>
14. Vertical/Horizontal Resource Scaling Mechanism for Federated Clouds. <https://doi.org/10.1109/ICISA.2014.6847479>.