International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161

Available at http://inpressco.com/category/ijcet

Research Article

Inter Cloud Scheduling with Priority using PTC Algorithm

Pankaj Singh[Á*], Sarvpal Singh[Á] and Neeraj Kumar Srivastava[Á]

[Á]Department of Computer Science & Engineering, Madan Mohan Malaviya University of Technology, Gorakhpur, (UP) India

Accepted 19 July 2014, Available online 01 Aug 2014, **Vol.4, No.4 (Aug 2014)**

Abstract

With the ever increasing number of cloud service providers, rising competition, and the growing popularity of cloud, it is extremely important for cloud providers to improve their services by improving the operational cost (response time and the processing time of requests). Since cloud computing is primarily driven by its cost effectiveness and as the scheduling of priority processes for profit, particularly in Inter-Cloud systems, has not been much focused, we propose a scheduling method using power of two choices with priority to address the service request scheduling problem with priority. The algorithm is based on power of two choices randomized load balancing in Inter-Cloud scheduling and has a priority feature. We will demonstrate that the giving priority to some processes by user for which he or she seeks better response time than average response time of the processes, using power of two choices with priority algorithm has a better response time for priority process. By providing better response time to priority processes Cloud service providers can take higher charges comparing with normal processes and thus increases their profit without much affecting the response time of the normal processes.

Keywords: *Inter-Cloud, priority, CSP, scheduling, response time, random, FCFS, I- JSQ, PTC.*

1. Introduction

Though Cloud Computing has a glorious future, many crucial problems still need to be solved for the realization of Inter-Cloud computing. One of these problem is load distribution with less complexity and better server utilization. It plays a very important role in the user request deployment in Inter-Cloud environment with minimum response time. It is used by Cloud service provider (CSP) in its own cloud computing platform to provide a high efficient solution for the user. Also, a inter CSP load distribution mechanism is needed to construct a low cost and infinite resource pool for the consumer. Load distribution in cloud computing is to distribute the local workload evenly to the whole cloud. In fact it has become indispensable for cloud computing. Thus Load distribution in cloud computing provides an organization with the ability to distribute application requests across any number of application deployments located in datacenters and through cloud computing providers.

An application operated on behalf of user in the Inter-Cloud consisting of one or more service requests is sent to the service provider specifying two main constraints, time and cost. Though the response time of a service request cannot be assumed to be accurately estimated, it is most likely that its actual processing time is longer than its original estimate due primarily to delays occurring on the provider's side. This causes the cloud computing are primarily operated by the principle of paying by time, so the service provider want to reduce the delay and improve the quality of their service. Service request scheduling is one of the most important methods to achieve those.

The random arrival of load in Inter-Cloud environment can cause some server to be heavily loaded while other server is idle or only lightly loaded. Equally load distributing improves performance by transferring load from heavily loaded server to lightly loaded. The considered characteristics have an impact on cost optimization, which can be obtained by improved response time and processing time.

2. Background

2.1 FCFS Algorithm

Equal load distribution problem can be solved by efficient scheduling algorithm. In (Maria Abu *et al*, 2011) explains the scheduling policies in virtual environment. First-Come, First-Served (FCFS) is a traditional scheduling policy that assigns one task per resource in the sequence in which the tasks have been submitted to the system. This algorithm has the biggest drawback that the processes those are coming after other processes have to wait for the processes in the queue to complete. In Inter-Cloud environment if any user want priority and is ready to pay more for immediate processing of his Vm requests then for the scenario presented above the FCFS algorithm is not a good choice.

2.2 PTC Concept

Power of two choices (PTC) concept as proposed by M. Mitzenmacher (M. Mitzenmacher, 1996) is used by (Leena

*Corresponding author: **Pankaj Singh**

V A *et al*, 2013) for random selection of cloud and server in Inter-Cloud environment and this performs far better than FCFS. To understand power of two choices concept, suppose one has a set of tasks S to distribute among a set of processors P, putting the condition that only one task can run on a processor at any time, a task must be run entirely on a single processor, tasks cannot be preempted, and processors compute at the same rate. Now the question is that if given the execution time of each task, how should one distribute them to minimize the final completion time? Answer is more interesting, even in the case where there are only two processors, finding the exact answer to this question is an NP-complete problem. Effective strategies that provide suitable performance in practice with less complexity has to be considered. In developing simple, effective algorithms, random selection often proves to be a useful tool. One possible way to distribute the tasks is to simply place each task on a random processor, that is, a processor chosen independently and uniformly at random. With this strategy, the expected load at each processor is the same, and thus, if there are enough tasks and the task sizes are not too different, then this strategy load the processors almost equally.

2.3 JSQ Scheduling

Join-the-Shortest-Queue (JSQ) algorithm (Yi Lu *et al*, 2011) dispatches jobs to the server with the least number of jobs in their queues . The JSQ algorithm is considered as a greedy algorithm from the view of an incoming job, as the algorithm receives the job the immediately and dispatches to the less loaded server. All incoming requests go through the centralized load balancer and it will find the server which has less loaded or which queue length is smallest and dispatches the request to that server. The load balancer is hence aware of all arrivals of jobs to a particular server, and queue length of all the server is tracked continuously. JSQ scheduling algorithm is considered to be throughput optimal (L. Ying *et al*, 2012) , but in Inter-Cloud environment there are lots of clouds and servers distributed to many geographical locations and tracking queue length for all servers causes a very large overhead. This results JSQ scheduling is not suitable for inter-clouds.

3. Previous Work

Since JSQ scheduling is not suitable for inter-clouds a variation of JSQ algorithm for inter-cloud scheduling, called I-JSQ is given by (Leena V A *et al*, 2013).

In the I-JSQ algorithm Cloud Service Provider (CSP) has a single centralized queue where request form user for VM and job execution will arrive and queued. From this central queue each request is selected and server for that request is chosen based on I-JSQ algorithm.

In I-JSQ algorithm two clouds are selected randomly and total load in both clouds is fetched and compared. The cloud with lesser load is selected. Now within that selected cloud two servers are randomly selected and their total loads are also compared. The server with lesser load is selected and the I-JSQ algorithm verifies that the selected server has all the required resources which is needed for execution and then send that request to selected server's job queue. If the selected server cannot accommodate the request then again two servers is randomly selected in the selected cloud. All these process is periodically done for all the user requests.

The performance of I-JSQ algorithm is compared with FCFS (First Come First Serve) algorithm in inter-cloud environment by comparing the mean delay for processing of the user requests by both the algorithms. The I-JSQ scheduling algorithm gives much better response time than First Come First Serve algorithm which means that a requests will less wait for execution when I-JSQ algorithm is used.

4. Our Approach

The I-JSQ scheduling algorithm in inter-cloud environment is used to increase the profit of CSP by providing good response time to priority processes comparing with non-priority processes. The CSP will charge higher for priority processes and response time of non-priority processes also not much affected comparing the response time provided by I-JSQ algorithm.

For this we have taken two queues instead of one queue which is used in I-JSQ algorithm (Fig.1). The priority jobs will go to queue of priority jobs and non-priority jobs will go to queue of non-priority jobs and this work is done by priority checker. From these two queues jobs are selected based on our proposed algorithm Inter-Cloud Scheduling With Priority using PTC Algorithm (ICSPPTC algorithm). This algorithm ensures that priority jobs will give good response without much affecting the response time of non-priority jobs.

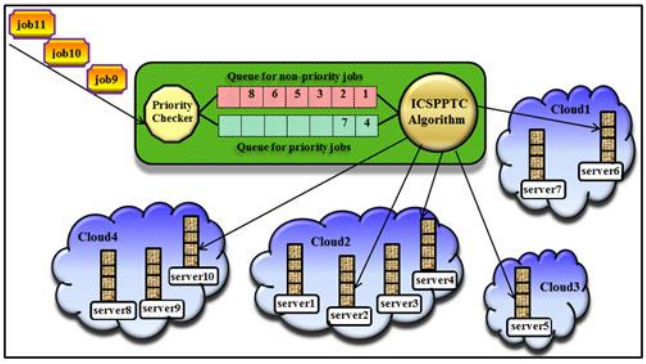

Fig.1 Inter-Cloud Scheduling With Priority using PTC Algorithm

4.1 ICSPPTC Algorithm

1. Initialize the number of clouds numCloud, number of different servers numServer in each cloud and configure the various resources in each of the server.
2. Add each priority VM request to queue q_p[ ] and each non-priority VM request to queue q_nop[ ].
3. First two selection will be from q_p[ ] and after that one selection from q_nop[ ], if jobs are available

in them. This cycle will be repeated until both queues becomes empty.

4. i = 0 and j = 0
5. while q_p[i] != NULL or q_nop[j] != NULL
6. Generate random number ranC1, ranC2 within range numCloud.
7. if (cloud_load[ranC1] < cloud_load[ranC2]) then
8. selectCloud = ranC1
9. else
10. selectCloud = ranC2
11. Generate random number ranS1, ranS2 within range numServer
12. if (server_load[ranS1] < server_load[ranS2]) then
13. selectServer = ranS1
14. else
15. selectServer = ranS2
16. Allocate VM to selectServer.
17. else repeat steps 11 to 19
18. update cloud_load and server_load.
19. end.

4.2 Simulation and results

The given algorithm is implemented in CloudSim version 3.0 cloud simulator. In order to generate different clouds and different number of server in them with different configurations, a program is written which randomly generates all the entities. We used 3 types of hosts in each server (Table 1).

Table 1 Different types of Hosts in a Server and their configurations

Parameters	Host Type1	Host Type2	Host Type3
MIPS	1000	10000	100000
Ram	1024	2048	3072
Storage	10000	100000	1000000
Bandwidth	100	1000	10000

Table 3 Different types of Cloudlets and their configurations

Parameters	Type1	Type2	Type3
Length	568923	7873291	1188365
FileSize	300	600	900
OutputSize	300	600	900
UtilizationModel	Full	Full	Full

The program is executed several times with each time generating random configuration of clouds, servers, users, VMs, cloudlets and their types. The results of all the executions are studied and comparison of response times is shown in Fig.2.

Aim of the proposed algorithm is that, we want lesser response time for priority processes without much affecting the response time non-priority processes. From Fig.3 consider red-line (I-JSQ algorithm) and violet-line (ICSPPTC non-priority processes), it is clear that average response time of non-priority processes is not much affected as compared with I-JSQ algorithm average response time. On the other hand comparing red-line and green-line (ICSPPTC priority process), we observe that priority process give good response time comparing with I-JSQ algorithm in different combination of clouds and users.

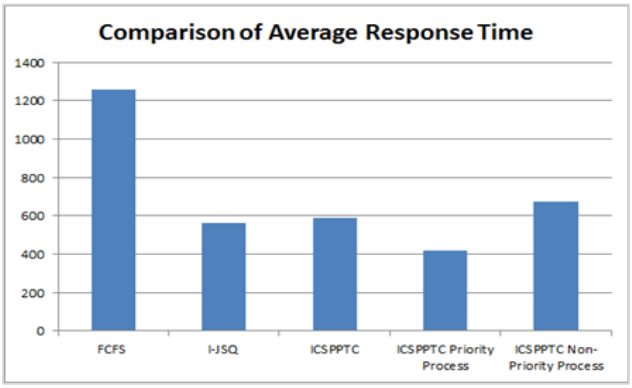

Fig.2 Comparison of average response time

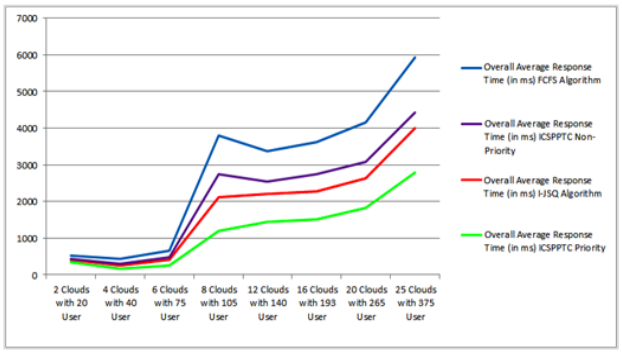

Fig.3 Comparison of outputs of different simulation scenarios through line chart

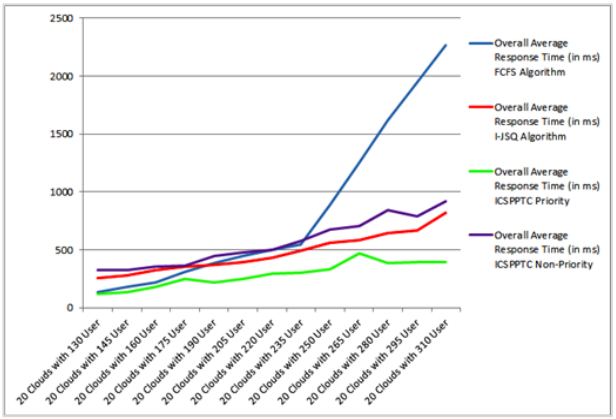

Fig.4 Performance comparison when servers are less loaded and heavily loaded

Above Fig.4 shows performance comparison of algorithm FCFS, I-JSQ and ICSPPTC priority and non-priority processes by varying the load on the system. We kept number of clouds fixed at 20 and vary the load on whole system. We see that when the system is less loaded then FCFS performs better than other algorithms, but when load increases more and more on system FCFS gives higher response time and I-JSQ algorithm performs better. We have extended I-JSQ algorithm by implementing

priority concept and proposed ICSPPTC algorithm, the average response time of priority process and non-priority processes by ICSPPTC algorithm are also comparable with I-JSQ and FCFS algorithm.

4.3 Limitations

1) The algorithm has been tested and analysed on the simulator, CloudSim. The delays and cost may differ from a real time cloud environment depending on different cloud infrastructures. The simulation results only serve as a basis for comparison.
2) The priority processes considered fewer than non-priority processes. If priority processes increases than non-priority processes, the average response time of non-priority processes will be more and more affected.
3) The work does not deal with any specific network topology.

5. Conclusion

1) The proposed algorithm decreases the response time of the some important jobs which user mark as priority process, without much affecting the response time of other processes.
2) From the result obtained in Fig.3 and Fig.4 , it could be deduced comparatively that response time of ICSPPTC non-priority processes are not much increased comparing with I-JSQ algorithm and ICSPPTC priority processes gives much better response time.
3) By providing lesser response time to priority processes a CSP can charge higher as compared with the normal processes. If a user has an urgent job for which he wants early execution than normal jobs, he set priority 1 to that job otherwise default priority is 0. Finding priority of the job set to 1, CSP charges higher to user for this job and executes this job from queue of priority processes from which 2 processes are selected per 1 selection of processes from queue of non-priority processes by proposed algorithm. This provides lesser response time for priority processes and thus increases the profit of the CSP with its existing resources.

6. Future work

In the proposed algorithm we are focused on decreasing the response time of the priority jobs which is urgent for user and user set priority for it. This work can be further extended by introducing multiple priority concept.

In this scheduler will be consists of several queues, the number of which depends on the how many priorities of the task units have. Before the task units are sent into the scheduler, they have their own initial priorities which are set by the system. The priorities depend on the task's source/ geographical location and the system design.

To reduce the waiting time of low priority processes with multiple priority, every task comes into the scheduler and will be divided into jobs, which are pushed into their corresponding queues with the cycle time T_{in}. The scheduler puts a new job in the end of its corresponding queue, every schedule unit does the schedule process with the cycle time T_{out}. To avoid a job with low priority wait so long, the scheduler sets every queue a threshold which means the limited time a jobs can wait in a queue of the schedule unit. If a jobs has waited for threshold time, the scheduler will move the jobs to a high priority queue. If the jobs in the highest priority queue has waited till threshold time, the scheduler will send this jobs to ICSPPTC algorithm for further cloud and server selection .

References

M. Mitzenmacher (1996), The power of two choices in randomized load balancing, Ph.D. dissertation, *University of California at Berke-ley*.

Stelios Sotiriadis (2013), The Inter-Cloud Meta-Scheduling Framework, *Ph.D. dissertation, University of Derby*.

M.N. Bennani and D.A. Menasc´e (2005), Resource Allocation for Autonomic Data Centers Using Analytic Performance Models, *Proc. 2005 IEEE International Conference on Autonomic Computing, Seattle, WA,*June 13-16.

Aditya Marphatia, Aditi Muhnot, Tanveer Sachdeva, Esha Shukla and Prof. Lakshmi Kurup (2010), Optimization of FCFS Based Resource Provisioning Algorithm for Cloud Computing, *IOSR Journal of Computer Engineering (IOSR-JCE).*

D.A. Menasc´e and M.N. Bennaniv (2006), Dynamic Server Allocation for Autonomic Service Centers in the Presence of Failures, in *Autonomic Computing: Concepts, Infrastructure, and Applications, eds. S.Hariri and M. Parashar, CRC Press*.

A. Beloglazov and R. Buyya (2010), Energy efficient allocation of virtual machines in cloud data centers, in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp.577578.

Leena V A, Ajeena Beegom A S and Rajasree M S (2013), Inter-Cloud Scheduling Technique Using Power Of Two Choices, *IEEE International Conference on Computational Intelligence and Computing Research*.

D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow (2009), Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability, *Fourth International Conference on Internet and Web Applications and Services*.

T. Dillon, C. Wu, and E. Chang (2010), Cloud Computing: Issues and Challenges, *24th IEEE International Conference on Advanced Information Networking and Applications*.

Tahereh Nodehi, Sudeep Ghimire, Ricardo Jardim-Goncalves and Antonio Grilo (2013), On MDA-SOA based Intercloud Interoperability framework, in *CMSS* - VOL. I, Issue 1

S. Zhou (1988). A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14:1327-1341.

D. Bernstein (2009), The Intercloud : Cloud Interoperability at Internet Scale, *Sixth IFIP International Conference on Network and Parallel Computing*.

N. Grozev and R. Buyya (2012), InterCloud architectures and application brokering: taxonomy and survey, *Software: Practice and Experience*.

Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R. Larus and Albert Greenberg (2011), Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services, *Journal Performance Evaluation* Volume 68 Issue 11, Pages 1056-1071.

L. Ying, S.T. Maguluri and R. Srikant (2012), Stochastic models of load balancing and scheduling in cloud computing clusters, in *Proc IEEE INFOCOM*.

Maria Abu, Aminu Mohammed Sani Danjuma and Saleh Abdullahi (2011). A Critical Simulation of CPU Scheduling Algorithm using Exponential Distribution, *IJCSI International Journal of Computer Science Issues*, Vol. 8, Issue 6, No 2.