Auto-Scaling and Priority Based Scheduling for Improving Efficiency of Cloud System

V.Bala Suthan[1],
M.E Scholar (M.E Communication and Networking), Cape Institute of Technology
[1] Email: balasuthan@live.com

***Abstract*:** Cloud Computing become most popular nowadays because of its reliability, and available on demand with pay as you go manner. Maintaining Demand Satisfaction Ratio among various user without violating the SLA (Service Level Agreement) is very big challenge for cloud service providers. In this paper we present a sort of algorithms that will allocate resource along the user more efficiently depend up on their priority or class, to maintain the SLA. This help the cloud services provider to make their services available for both free and in paid manner by giving more priority to the paid user. Without violating the SLA of paid user the free user can access the cloud resources. This algorithm increases overall demand satisfaction ratio of cloud service by using Auto scaling algorithm. This method also support green computing by using the available resource more efficiently by placing the task adaptively among the server and make ideal server under sleep mode using wake on the LAN technology.

***Keywords**: Cloud Computing, Auto Scaling, Green Computing, Virtualization.*

I. INTRODUCTION

Cloud computing is a way of delivering computing as a service through network or Internet to the user. Cloud service provider provide various computing resource such as storage, Infrastructure, application etc. to the users. Cloud computing allow user to perform virtually any compute or data storage operations by scaling and provisioning necessary resource on demand on a pay as you go basis this made cloud computing most popular nowadays. Development of smartphone devices and high speed mobile network allow various mobile user to use cloud to execute their complex and heavy weight computing task on cloud with their hand held mobile device or Tablet Pcs this is refer as MCC(Mobile Cloud Computing).

Many cloud service provider also provide their service for free of cost. This efficiency of cloud and availability of various services make number of person using cloud to grow rapidly day by day. Normally cloud present an illusion that it has infinite capacity but in reality datacentres are having limited resources only. When large number of application attain its peak at the same time even cloud cannot satisfy some demand. So to serve the huge population of requests efficiently without violating their SLA (Service Level Agreement) is a very big challenge. To face this challenge cloud service provider should optimize their resource allocation to every user as well as reduce the usage cost and energy. To obtain this, this paper proposed a method that uses a sort of algorithm for scaling a scheduling the cloud application dynamically depend up on priority of users. This method optimize the resource available for priority or paid user while providing Best Effort service for the free users. If overall demand goes high this method can maintain the Demand satisfaction ratio of priority customers as high and only provide best effort service to the free users so it will maintain the SLA of paid user even in high demand. When in the low demand time it gives good service to both the free users as well as priority customers.

Proposed method is a combination of two algorithm one of that is used to schedule the user request or task depend up on its priority without violating the SLA. Another one will automatically scale the cloud resource depend up on the request dynamically and more efficiently to get high demand

satisfaction ratio. This is the ratio between the number of request or task that execute without violating the SLA to the total number of request or task submitted by users. This algorithm supports green computing by adaptively place the application instance along the running server so as to reduce the number of server in use and put the ideal sever in to sleep mode to reduce the energy. We construct this scaling problem as the Class Constrained Bin Packing problem as mention in [1].Here each class represent as the task or user request and Bin represent as the Server. Here the solution is to filling the bin with the class but only after filling a current bin completely we can move to another. Here the problem is solved by using the proposed modified Semi online algorithm proposed in this paper. This method encapsulate each application inside a separate VM (Virtual Machine) to get fault tolerance it also follow fast restart techniques based on virtual machine suspend and resume to reduce the application start up time.

Rest of the paper is organized as follows section 2 present the Related Work. System Architecture is presented in section 3, section 4 consist of Algorithm. Section 5 and 6 consist of Simulation Result and Conclusion.

II. RELATED WORKS

Many previous works have been done by many researchers for providing efficient cloud computing services most of them are concentrate particularly on single parameter like to reduce energy usage or optimum scheduling algorithm for increasing performance or priority based scheduling. In [1] authors presented the method that concentrate on both demand satisfaction and Energy efficiency and it is used for internet application we take some concepts from this paper and optimize or modified the algorithms in this paper to provide very high demand satisfaction ratio. This paper concentrate on most important three parameters for cloud performance such as priority based Scheduling, Dynamic scaling to increase demand satisfaction ratio, and also concentrate on energy saving.

In [2] the authors presented a model that can monitor the QoS oriented cloud computing resource availability on cloud which help us to estimate the current situation or resource availability to full fill particular QoS. In [3] authors developed a method that schedule the task depend up on its priority using modified waiting queen model. In this paper the scaling problem is modelled as the Class Constrained Bin packing Problem (CCBP) as presented in [1],[4][6]. Class Constrained Multiple Knapsack problems (CCMK) is used in [7],[8] it aims to maximize the total number of packed item but un like Class Constrained Bin Packing (CCBP) it will not reduce the number of knapsack in used that is it does not minimize the energy use. To solve this Class Constrained Bin Packing problem various methods or algorithms have been proposed.

In [5] authors proposed a tight bound algorithm but it can't rearrange the packed item in the class. Many offline algorithm are available but it does not consider the item departure when colour set become empty due to item departure and if it is not repacked then it definitely leads to performance degradation. Also many strict online algorithm are available but it can't move the task once it was packed. It has been shown in [9] that existing colour set algorithm have poor performance when there is frequent item departure this cannot be used in cloud computing. The application placement in enterprise environment is shown in [9],[10]. In [11],[12],[13] authors concentrate on resource provisioning on web farm some allocate task in granularity of full server it leads to poor resource utilization. [13] This paper do not consider the practical limit on the number of application a server can run simultaneously. In [14] authors proposed two greedy algorithms, to generate the static allocation: the cloud list scheduling (CLS) and the cloud min-min scheduling (CMMS). In [15] author presented a view of various open source cloud system, It provide the way, how to use open source cloud systems. Various scheduling algorithms and method depend on various metrics such as Latency, time and cost for cloud and web services are

proposed in this papers [17], [21], [22], [23]. In [18] authors proposed a Greedy Resource allocation algorithm this will reduce the number of server in use so as to reduce the energy usage.

III. SYSTEM ARCHITECTURE

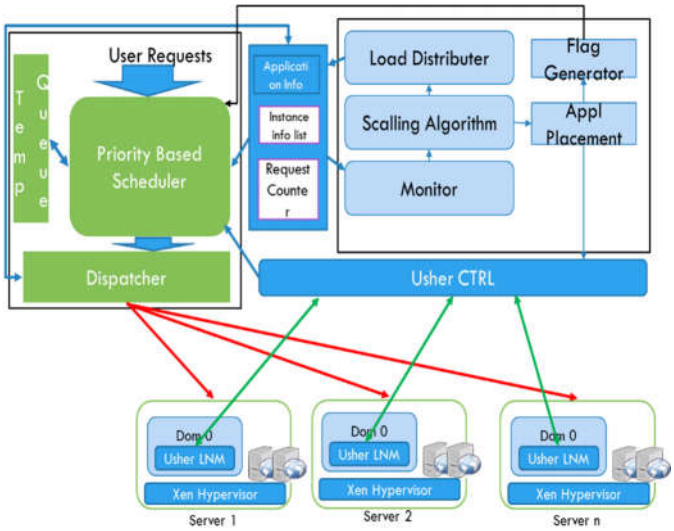

Figure 1. System Architecture

Figure 1. Shows the architecture of the proposed system. The system architecture shows the three segments, Application Scheduler Plugin. Priority Based Scheduler Plugin and Infrastructure consist of servers. This priority based scheduler is the modified version of layer 7 switch which consist of extra priority based scheduler block. Infrastructure consist of n number of servers depend on the size of the cloud data centre. Request from the client is directly send to the Priority based scheduler that runs the priority scheduling algorithm that schedule or arrange the tasks depend on the priority information available on it. It also send the number of application request to the request counter in the application info. Then the application scheduler plugin is invoked.

This application scheduler plugin consist of Usher CTRL [20] that dynamically maintain the running instance, also monitor the resource availability of each server and its current running application. The monitor plugin get all the require information from the Usher CTRL and Current request rate from the application info and send it to the colour set adjustment plugin. This colour set adjustment plugin run the proposed modified colorset algorithm to configure the application placement change. The new configuration is then send to the priority scheduler plugin where dispatcher used this new configuration and distribute the task along the server. The application scheduler plugin is used to increase the demand satisfaction ratio as well as to efficiently use the resource available. Each application is embedded in to separate virtual machine [16] to reduce the fault during run time.

LNM in each node and layer 7 switch with Priority scheduler collect the application placement information, resource usage of each instance, and total request number of each application. This information are send to the Usher central controller above which the application scheduler runs.

Application scheduler is invoked only when it is needed to make the following decision.

Application Placement: For each application it allocate a set of server to run its instance

Load Distribution: For each application it predict the future demand depend up on the current request rate and past statistics and then decide how to distribute this load among the running servers.

This decisions are forwarded to the dispatcher in layer 7 switch block and to the LVM in the local node for execution. The command to the LVM consist of

- Command to standby or wakeup
- Command to start new application or stop running application
- Command to allocate local resource along the application

LNM at the local node adjust the local resource allocation among the VM(Virtual Machines). Here Xen Hypervisor is used It can change the CPU allocation among the VMs by adjusting the weights in the CPU schedulers [19]. Self-ballooning techniques are used to allocate memory among the VM. This will allocate the memory dynamically among the VM depend up on the need so it reduce the unwanted usage of memory resource. The decision interval of the scheduler depend on application demand change and on frequent placement change.

V. DETAILS OF ALGORITHM

Here in this paper two algorithms are proposed one is to schedule or arrange the application based on its priority and the second algorithm is for resource scalability to provide high demand satisfaction ratio.

Priority scheduling algorithm:

In this method the request from each user is consider to having three metrics, Priority (For free user it is 0 and for priority user or paid user it is 1), Arrived time (This is the time when application request reached the server), Living time (Max time up to which request is valid, exceeding this will break the SLA). Normally this added when the application request reached the datacentre depend on the user priority. Then depend on this three metrics the scheduling algorithm will schedule the task or application request adaptively.

Algorithm_1

```
Input = user request with SLA (Priority, Arrived time, living time );
Dead time= living time-Arrived time;
Arrange Task ascendingly depend on dead time;
Add to Entry queue;
x=0;
C=Average Execution time of tasks;
Monitor A=Available Virtual Machine (from usher CTRL);
If (A is sufficient for current Request)
        send : request to dispatcher
        Result=success;
        call alg 2;
Else
    1:  update (current request rate in app info)
        run scaling algorithm;
        if (scaling algorithm= success);
        send : request to dispatcher;
        result=success;
        call alg_2;
        else if(scaling algorithm= failed);
        if (free user request in queue );
        x=x+10;
        move :x% of free user request to temp queue;
        goto 1:
        else :
```

```
        result=failed;
        wait for C time;
        Remove Dead task from Entry queue;
        Goto 1;
End
```

Algorithm_2

```
If (free request in queue )
Remove dead task return failed;
Arrange remaining task in ascending order with dead time;
X=0;
2: Monitor A =Available Virtual Machine
If(A<=80% total VM)
X=x+10;
Update info list with X% free user request;
Run scaling algorithm
If (scaling algorithm =success)
        move X% free request to dispatcher;
        result = success;
        goto 2:
else
remove updated free instance from infolist
Call alg_1
Goto 2:
Else
Call alg_1;
end
```

Auto Scaling Algorithm:

This algorithm is used to automatically scale the resource allocated to the current process. This is done by using the following step. Here this problem is formed as a class constrained Bin Packing problem and solved using the proposed Modified Semi online Colour set Algorithm. Here each Bin denote the Server or host in the data centre and class denote the application instances. Colourset is the set of instance from same application.

Initialization

i) Sort the list of unfilled colour set in descending order
ii) Use greedy algorithm to add the new colour into sets according to the position
iii) If there are still new colours left after filling up all unfilled sets then partition the remaining new colours into additional colour set using greedy algorithm
iv) If we run out of new colour before filing up all but the last unfilled go to next step
v) Consolidate _unfilled_set Procedure
vi) Sort the list of unfilled colour set in descending order
vii) Use the last set in the list to fill the first set in the list
viii) Repeate the previous step until there is only one unfilled set left in the list
ix) Procedure fill (s1,s2)
x) Sort the list of colors in s1 in ascending order of their number of items.
xi) Add the first colour in the list into s2
xii) Repeate above steps until either s1 become empty or s2 become full.
xiii) If free s>0 return success;
xiv) If free s<0 return fail;

New Item Arrival

If new item arrived

i) Search unfilled bin for current application
ii) If (unfilld bin has application running)
iii) Send item to bin
iv) Else(search for bin with current app)
v) Move an item to unfilled bin from founded bin
vi) Repack the current item in free space

Item Leave

If item leave

i. If (unfilled bin has application run in current bin)
ii. Repack the empty space with item from unfilled bin
iii. Else repack the item between the bin
iv. If unfilled bin become empty made it to sleep

VI. RESULTS AND ANALYSIS

Performance of the proposed system is evaluated using the Simulation tool. The performance of the proposed system is analysed using two metrics. One demand satisfaction ration another is power we evaluate the result for this two metrics and compare that with the existing system. Here the power is calculated by monitoring the number of servers in use for corresponding demand power utilization is directly proportional to number of server in use.

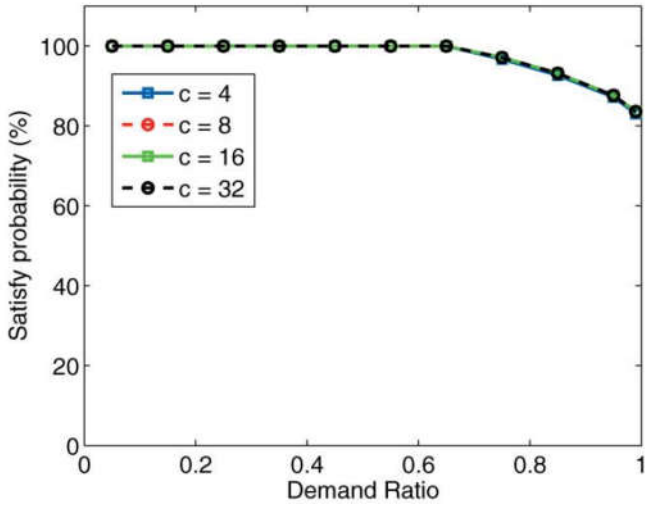

Figure 2. Demand satisfaction ratio of existing system

Figure 2. Show the demand satisfaction ratio of the existing system [1]. Here the number of server in the data centre is kept constant and the demand is increased from 1% to 100% and corresponding demand satisfaction ratio is measured.

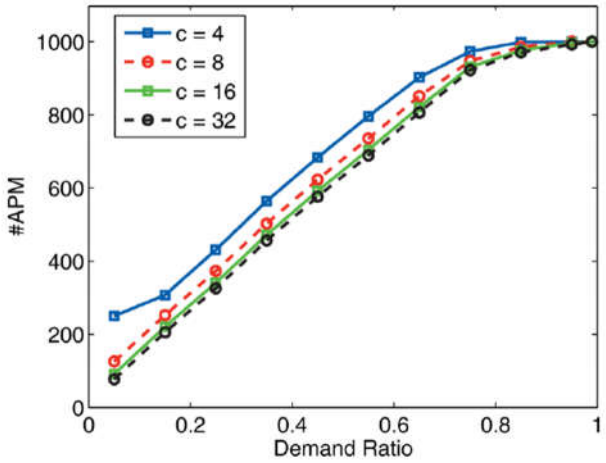

Figure 3. No. of APM vs Demand of Existing system

Figure 3. Shows the power utilization of the existing system [1]. That is it gives no. of APM (Active Physical Machine) for corresponding Demand ratio here C denote the no of different class that runs on the data centre.

Total Demand	Demand satisfaction of Priority User					Average Demand satisfaction of free user
	80% Free user Request	60% Free user Request	40% Free user Request	20% Free user Request	0% Free user request	
20%	97.56	97.42	97.04	97.43	97.52	97.21
40%	97.53	97.38	97.32	97.45	97.51	97.22
60%	97.33	97.44	97.34	97.41	97.31	97.26
80%	97.45	97.33	97.40	97.03	97.23	92.22
100%	97.33	97.45	97.42	97.68	93.34	63.34
120%	95.92	96.93	96.54	93.32	84.12	52.04
140%	95.23	96.44	94.06	85.05	71.65	41.65
160%	93.45	92.06	84.65	72.54	63.23	21.34
180%	92.75	83.78	71.08	63.23	54.62	12.30
200%	79.98	72.57	64.96	55.80	42.84	7.2

Figure 4. Demand satisfaction ratio of Proposed system.

Figure 3. Shows the demand satisfication ratio of the proposed system. It gives the demand satisfaction ratio of priority users as well as free user separately. Simulation is done by keeping the number of host as the constant and the demand is increased from 1% to 200% with different ratio of free user request from 0% to 80% and corresponding demand satisfaction ratio is noted.

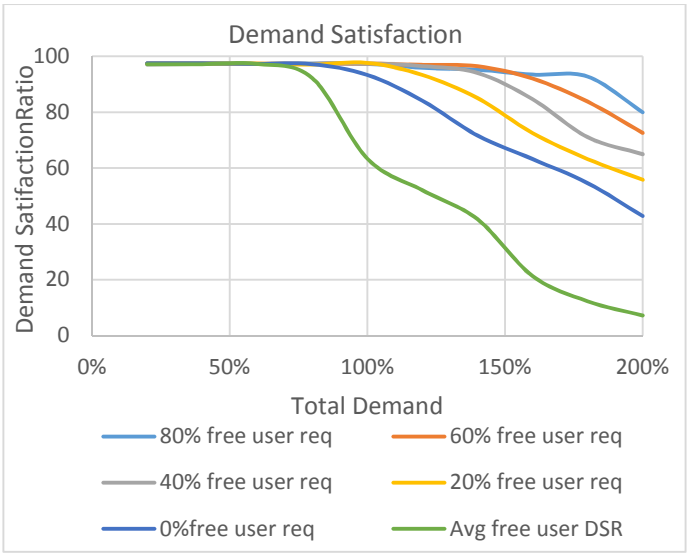

Figure 5. Demand Satisfaction Ratio Graph

Figure 5. Gives the graphical representation of the demand satisfaction ratio obtain during the simulation. This result clearly denote that up to the 80% of total demand both free user as well as the priority user get nearly 100% demand satisfaction as the total demand go beyond 100% we maintain the demand satisfaction ratio of the priority user by compromising on demand satisfaction ratio of free users. Here depend on the contribution of the priority user request the demand satisfaction free user get varied if priority user have more request then it automatically reduce the demand satisfaction ratio of free user.

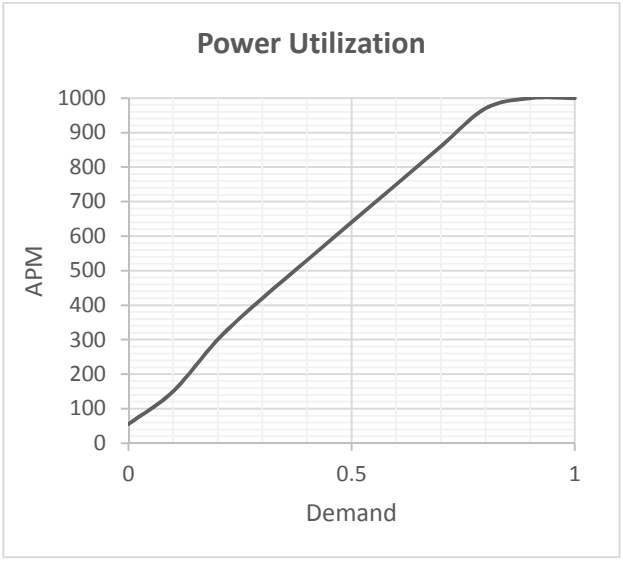

Figure 6. Power utilization of proposed system

Figure 6. Shows the number of APM (Active Physical Machine) versus Demand. Here we calculate the power indirectly by monitoring the number of APM corresponding to the total demand. This graph is Identical to the existing system power utilization, this shows that the proposed system also support green computing this is the added advantage to the proposed system.

VII. CONCLUSION

In this paper we present the system that will increase and maintain the demand satisfaction ratio of the priority uses at the same time it serve good for the free user in normal situation. This will reduce the cloud service provider cost of running for free user as well as give high demand satisfaction ratio for both priority and free users. The performance evaluation of our system is measure by the simulation and the simulation results shows that ours system is more reliable and increase overall performance of cloud while reducing the cost of cloud service providers as well as it reduce the energy consumption and support green computing.

REFERENCES

1. Zhen xiao, senior member, IEEE, qi chen, and haipeng luo "Automatic scaling of internet applications for cloud computing services" IEEE transactions on computers, vol. 63, no. 5, may 2014
2. WANG En Dong WU Nan "QoS-oriented Monitoring Model of Cloud Computing Resources Availability" International Conference on Computational and Information Sciences 2013.
3. Pawar, C.S., Rajnikanth B. Wagh "Priority based dynamic resource allocation in Cloud computing with modified waiting queue" International Conference on Intelligent Systems and Signal Processing (ISSP) 2013.
4. L. Epstein, C. Imreh, and A. Levin, "class constrained bin packing revisited," theor. Comput. Sci., Vol. 411, no. 34–36, pp. 3073–3089,2010.
5. H. Shachnai and T. Tamir, "tight bounds for online classconstrained packing," theory. Comput. Sci., Vol. 321, no. 1, pp. 103–123, 2004.
6. E. C. Xavier and F. K. Miyazawa, "The class constrained bin packing problem with applications to video-on-demand," Theor. Comput. Sci., vol. 393, no. 1–3, pp. 240–259, 2008.
7. H. Shachnai and T. Tamir, "On two class-constrained versions of the multiple knapsack problem," Algorithmica, vol. 29, no. 3, pp. 442–467, 2001.
8. H. Shachnai and T. Tamir, "On two class-constrained versions of the multiple knapsack problem," Algorithmica, vol. 29, no. 3, pp. 442–467, 2001.
9. A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in Proc. Int. World Wide Web Conf. (WWW'06), May 2006, pp. 595–604.
10. J. Famaey, W. D. Cock, T. Wauters, F. D. Turck, B. Dhoedt, and P.Demeester, "A latency-aware algorithm for dynamic service placement in large-scale overlays," in Proc. IFIP/IEEE Int. Conf. Symp. Integrat. Netw. Manage. (IM'09), 2009, pp. 414–421.
11. B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," SIGOPS Oper. Syst. Rev., vol. 36, no. SI, pp. 239–254, 2002.
12. M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: A mechanism for resource management in cluster-based network servers," SIGMETRICS Perform. Eval. Rev., vol. 28, no. 1, pp. 90–101, 2000.
13. J. L. Wolf and P. S. Yu, "On balancing the load in a clustered web farm," ACM Trans. Internet Technol., vol. 1, no. 2, pp. 231–261, 2001.
14. Jiayin Li, Meikang Qiu, Jian-Wei Niu, Yu Chen, Zhong Ming "Adaptive Resource Allocation for Preemptable Jobs in Cloud Systems," in 10th International Conference on Intelligent System Design and Application, Jan. 2011, pp. 31-36.
15. E. Caron, L. Rodero-merino, F. Desprez, and A.Muresan, "autoscaling,load balancing and monitoring in commercial and open source clouds," INRIA, rapport de recherche RR-7857, feb. 2012.
16. J. Zhu, Z. Jiang, Z. Xiao, and X. Li, "optimizing the performance of virtual machine synchronization for fault tolerance," IEEE trans. Comput., Vol. 60, no. 12, pp. 1718–1729, dec. 2011
17. J. Famaey, W. D. Cock, T. Wauters, F. D. Turck, B. Dhoedt, and P. Demeester, "A latency-aware algorithm for dynamic service placement in large-scale overlays," in proc. IFIP/IEEE int. Conf. Symp. Integrat. Netw. Manage. (IM'09), 2009, pp. 414–421.
18. J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "managing energy and server resources in hosting centers," in proc. ACM symp. Oper. Syst. Princ. (SOSP'01), oct. 2001, pp. 103–116

19. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "xen and the art of virtualization," in proc. ACM symp. Oper. Syst. Princ. (SOSP'03), oct. 2003, pp. 164–177.
20. M. Mcnett, D. Gupta, A. Vahdat, and G. M. Voelker, "usher: an extensible framework for managing clusters of virtual machines," in proc. Large install. Syst. Admin. Conf. (LISA'07), nov. 2007, pp. 1–15.
21. S. K. Garg, R. Buyya, and H. J. Siegel, "Time and cost trade off management for scheduling parallel applications on utility grids," Future Generation. Computer System, 26(8):1344–1355, 2010.
22. 12 S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in AINA '10: Proceedings of the 2010, *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 400–407, Washington, DC, USA, 2010, IEEE Computer Society.
23. M. Salehi and R. Buyya, "Adapting market-oriented scheduling policies for cloud computing," In Algorithms and Architectures for Parallel Processing, volume 6081 of Lecture Notes in Computer Science, pages 351–362. Springer Berlin / Heidelberg, 2010.