Improve Resource Allocation for Cloud Computing Environment

R.Navinkumar, MCA., R.Ramamoorthy (Final MCA).,

Assistant Professor,
Department of Computer Application, Nandha Engineering College, Erode, Tamil Nadu, India.

Abstract*—The elasticity and the lack of upfront capital investment offered by cloud computing is appealing to many businesses. There is a lot of discussion on the benefits and costs of the cloud model and on how to move legacy applications onto the cloud platform. Here we study a different problem: how can a cloud service provider best multiplex its virtual resources onto the physical hardware? This is important because much of the touted gains in the cloud model come from such multiplexing.*

Cloud computing allows business customers to scale up and down their resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. This project presents a system that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use.

The project introduces the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources. It develops a set of heuristics that prevent overload in the system effectively while saving energy used.

I. INTRODUCTION

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software and information remain provided to users over the network. Cloud computing providers deliver application by the Internet, which are accessed after web browser, though the business software and data are stored on waiters at a remote location.

Cloud providers are bright to attain the agreed SLA, by scheduling resources in effectual manner and by deploying application on good VM as per the SLA objective and at the same time performance of the applications necessity be optimized. Presently, here exists a more work done on scheduling of applications in Clouds [1], [2], [3]. These methods are usually seeing one single SLA objective such as cost of execution, execution time, etc. Owing to combinatorial countryside scheduling algorithm with multiple SLA objective for best mapping of load with multiple SLA parameters to resources is originate to be NPhard [4]. The available explanations are based on the use of heuristics.

Once a job is submitted to the clouds, it is usually divided into several tasks. Next two questions are need to consider when applying parallel dispensation in executing these tasks: 1). how to assign resources to tasks; 2) task are executed in pardon order in cloud; and 3) how to schedule overheads when VMs prepare, dismiss or switch tasks. Task scheduling and resource allocation can solve these three problems. In embedded systems [5], [6] and in high performance computing [7], [8] chore scheduling and resource allocation have been studied.

Classically, efficient provisioning needs two distinct steps or processes: (1) initial static planning step: the originally group the set of VMs, formerly classify them and deployed onto a set of bodily hosts; and (2) dynamic resource provisioning: the allocation of additional resources, creation and migration of VMs, dynamically responds to variable workload. Step 2 runs unceasingly at production time anywhere in contrast Step 1 is usually performed at the early system set up time and may only be recurrent for overall clean-up and upkeep on a monthly or semi-annually schedule.

Now this paper we focus on dynamic resource provisioning as stated above in step 2. In order to attain the agreed SLA objective our proposed algorithm dynamically replies to fluctuating work load by pre-empting the current executing task taking low priority with high importance task and if pre-emption is not possible due similar priority formerly by creating the new VM form worldwide available resources.

In section II, we deliberate works related to this subject. In section III, models for resource distribution and task scheduling in IaaS cloud computing system remain presented. We propose our algorithms in section IV, trailed by experimental result in section V. Lastly, we give the conclusion in section VI.

II. RELATED WORK

In [9] author proposed architecture, using feedback control attitude, for adaptive management of virtualized resources, which is founded on VM. In this VM-based building all hardware resources are joint into common shared space in cloud computing substructure so that presented application can access the required capitals as per there need to see Service Level Objective (SLOs) of application. The adaptive manager use in this architecture is multi-input multi-output (MIMO) reserve manager, which includes 3 controllers: CPU controller, memory controller and I/O controller, its goalmouth is regulate multiple virtualized resources use to achieve SLOs of application by using control inputs per-VM CPU, memory and I/O allocation.

The influential work of Walsh et al. [10], proposed a general two-layer architecture that usages utility functions, accepted in the context of dynamic and autonomous resource allocation, which consists of local agents and global arbiter. The accountability of local agents is to calculate efficacies, for given current before forecasted workload and variety of resources, for apiece AE and results are transfer to global arbiter. Anywhere, global arbitrator computes near-optimal configuration of resources founded on the results providing by the local agents. In global arbitrator, the new formations applied by assigning new capitals to the AEs and the new configuration computed also at the end of fixed control intervals or in an event activated manner or anticipated SLA violation.

In [11], writers propose an adaptive resource allocation algorithm for the cloud system with pre-empt able tasks in which algorithms regulate the resource allocation adaptively based on the efficient of the actual task executions. Adaptive list scheduling (ALS) and adaptive min-min scheduling (AMMS) algorithms are used for task scheduling which includes still task scheduling, for static resource allocation, is generated offline. The online adaptive process is use for re-evaluating the residual static resource allocation repeatedly with predefined incidence. In each re-evaluation process, the schedulers are re-calculating the surface time of their respective submitted tasks, not the tasks that are assign to that cloud.

The dynamic resource distribution based on distributed multiple criteria choices in computing cloud clarify in [12]. In it author influence is two-fold, first dispersed architecture is adopted, in which resource organisation is divided into independent tasks, apiece of which is performed by Autonomous Node Agents (NA) in ac series of three activities: (1) VM Placement, in it appropriate physical machine (PM) is originate which is capable of seriatim given VM and then assigned VM to that PM, (2) Nursing, in it total resources use by hosted VM are monitored by NA, (3) In VM Selection, if local lodging is not possible, a VM need to migrate at another PM then process loops back to hooked on placement. Then second, using PROMETHEE method, NA carry out shape in parallel through multiple criteria decision analysis. This approach is possibly more feasible in large data centres than centralized approaches.

The problem of resource allocation is careful in [13], to optimize the total income gained from the multidimensional SLA agreements for multi-tire application. In it the upper certain of total profit is provided by the help of force-directed resource assignment (FRA) heuristic algorithm, in which initial explanation is based on providing solution for profit upper certain problem. Next, distribution rates are fixed and local optimization step is use for refining resource sharing. Lastly, a resource consolidation practise is applied to consolidate capitals to determine the active (ON) servers and further enhance the resource assignment.

Using steady state timing models, this [14] paper intelligences a study of cloud HPC resource preparation. In it author propose measureable application dependent instrumentation technique to investigate multiple important dimensions of a agenda's scalability. Consecutive and parallel timing model with program arrangements can reveal architecture specific deliverable presentations that are difficult to quantity otherwise. These models are introduces to attach multiple dimensions to time domain and application speed up model is use to tie these models in same equation. The aptitude to explore multiple dimension of program quantitatively, to gain non-trivial vision. For target processing setting authors use Amazon EC2.

In earlier years, the aims of dispersed system have been cantered on the decoupling of interfaces after service oriented architectures (SOA) [16], application, subscription model, hosting models then social

collaboration. Lately, Internet-based dispersed, multitenant [17] applications connective to internal business applications, recognised as software as a service (SaaS) [18], are ahead popularity. The previous work [19-21] on web application scalability applied for static load balancing solution by server clusters but the dynamic scaling of web applications in virtualized cloud computing has not been much deliberated. Since such kinds of work load require minimum retort time and high level of availability and dependability from web applications.

A explanation for dynamic scaling of web application provided in [22] by describing an architecture to scale web application in dynamic

method, based on dawn in a virtualized cloud computing environment. Building consists of front-end load balancer, a no. of web application virtual machine. In it apache HTTP Load Balancer is a front-end load-balancer for steering and balancing user requests to web application deployed on Apache HTTP server that are installed in Linux virtual machine. As per the request these virtual machines are started and provisioned by a provisioning sub-system. Then the action of provisioning and de-provisioning of web server cybernetic machine cases control by a dynamic scaling algorithm based on relevant verge of web application.

III. USE TECHNIQUES

In this section we are recitation SLA based resource provisioning then online adaptive scheduling for Pre-emptible task execution, these two practises which are combined in proposed algorithm for actual utilization of cloud resources to see the SLA objective.

A. Cloud Resource provisioning and scheduling heuristic

The service requests from customers crowd by combining the three different layers of resource provisioning as shown
in following figure 1[24].

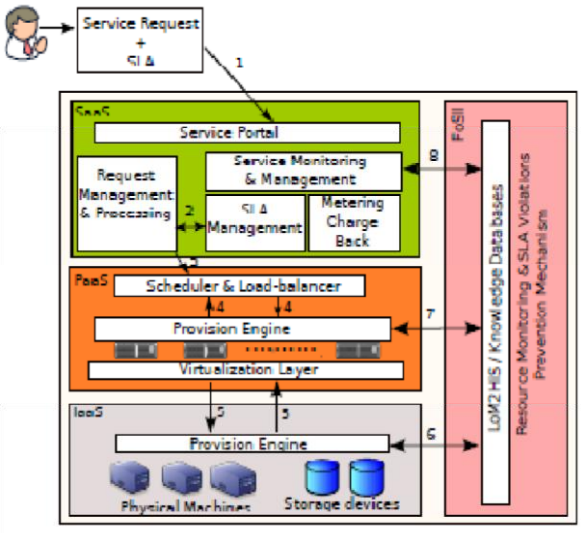

Figure 1. Cloud Provisioning and Deployment model

Service deployment requests after customers is place to the service portal (step 1 in Figure1), which forward the requests to the appeal management then processing component to authenticate the requests with the help of SLA (step 2). If the request is valid, it is formerly pass to the scheduler then load-balancer (step 3). Aimed at deploying the requested service, scheduler picks the appropriate VMs, as per SLA and priority, ended the provisioning engine in PaaS layer and the load-balancer balances the service provisioning amongst the running VMs (step 4). The VMs on the virtualization layer achieve by provision trainthen the virtualization layer interconnects with the physical resources with the help of the provision locomotive in IaaS layer (step 5).The LoM2HiS framework screens the low-level resource metrics of the physical assets at IaaS layer [25] (step 6). If SLA violation happens, sensitive actions deliver by the info database techniques [26] in FoSII (step 7). The requested service location and the SLA information are connected back with the service portal (step 8).

Provisioning can be complete at the single coatings alone. Though, approach which we careful in [24] aims to provide an combined resource provisioning plan. So, scheduling heuristics in [24] reflects the three layers.

An aim of scheduling experiential in [24] is to schedule job on VMs based on the agreed SLA objective and creating new VMs on bodily resources based on availabilities resources. This policy helps to enhanced application performance then at the same time reducing the potentials of SLA violations. Then, the combined load-balancer (Algorithm 1 Load-balancer given below) in the heuristic protections high and effectual resource operation in the Cloud setting.

The customers' service deployment requests (R) is provide as input to scheduler which contain of the SLA terms (S) and the request data (A) to be provisioned. Formerly it gets the total available physical resources (AR) formerly the number of running VMs in the data center in cloud. The SLA footings are used to find a list of suitable VMs (AP) talented of provisioning the requested service (R).

The load-balancer is available below in Algorithm 1. Suitable VM list is provided as input to it (line 1 in Algorithm 2). In its procedures, in order to know how to balance the load between the VMs it first discoveries out the number of available consecutively VMs in the data centre (line 2). In the next step, it gets a list of VMs which are already billed to job i.e. list of used VMs. (line 3). It clears the list if this tilt is equivalent to the number of running VMs, because that income all the VMs are currently allocated to some applications (lines 4-7).

Algorithm 1 Load Balancer

1. Input: AP(R,AR)
2. Available VM List //list of available VMs form each cloud
3. Used VM List //list of VMs, currently provision to certain job
4. Deployable Vm=null

5. **if** size(usedVMList)=size(availbleVMList) **then**
6. clear usedVMList
7. **End if**
8. **for** vm in (AP,R,AR) **do**
9. **if** vm not in usedVMList **then**
10. Add VM to usedVMList
11. deployableVm= vm
12. Break
13. **End if**
14. **End for**
15. Return deployableVm

So, the first VM from the appropriate and available VM list can be selected aimed at the deployment of the new job request. Finally, the selected VM will be added to the list of used VMs so that the load-balancer will not choice it in the next repetition (lines 8-15).

B. Preemptable task execution

Once a scheduler finds customer's service request, it will first divider that service request into shops in the form of a DAG. Previously initially static resource allocation is done. In [11] authors forthcoming two greedy algorithms, to make the static distribution: the cloud list scheduling (CLS) and the cloud min-min scheduling (CMMS).

1) Cloud list scheduling (CLS):This CLS is similar to CPNT [27]. The meanings used for listing the task are provided as shadow. The earliest start time (EST) and the latest start time (LST) of a task are exposed as in (1) and (2).The entry-tasks have EST equals to 0. Then The LST of exit-tasks equal to their EST.

$$EST(v_i) = \max_{v_m \in pred(v_i)} \{EST(v_m) + AT(v_m)\} \ldots (1)$$

$$LST(v_i) = \max_{v_m \in succ(v_i)} \{LST(v_m)\} - AT(v_i) \ldots (2)$$

As the cloud system worried in [11] is varied the execution time of each task on VMs of different clouds are not the same. AT( ) is the average execution time of task . The critical node (CN) is a set of apexes having equal EST and LST in the DAG. Algorithm 2 shows a function starting a task list based on the priorities.

Algorithm 2 Founding a task list based on priorities

Require (input): A DAG, Average execution time

AT of every task in the DAG

Ensure (output): A list of task P based on priorities

1. The EST is calculated for every task
2. The LST is calculated for every task
3. The T_p and B_p of every task are calculated
4. Empty list P and stack S, and pull all task in the list of task U
5. Push the CN task into stack S in decreasing order of their LST
6. **While** the stack S is not empty do
7. **If** top(S) has un-stacked immediate predecessors **then**
8. S ← the immediate predecessor with least LST
9. **Else**
10. P ← top(S)
11. Pop top(S)
12. **End if**
13. **End while**

Once the above algorithm 2 form the list of task according there priority, we can allocate resources to tasks in the order of formed list. When all the predecessor tasks of the assigned task are finished then cloud resources allocated to them are accessible, the assigned task will start its effecting. This task is removed from the list afterward its assignment. This procedure is repeats until the list is empty.

2) Cloud min-min scheduling (CMMS):Min-min scheduling is popular greedy algorithm [28]. The dependences among tasks not careful in original minmin algorithm. Thus in the dynamic min-min algorithm used in [2], authors uphold the task dependences by updating the map able task set in every preparation step. The tasks whose precursor tasks are all assigned are placed in the map able task set. Algorithm 3 shows the quasi codes of the CMMS algorithm.

A cloud scheduler record implementation schedule of all resources using a slot. Once an AR task is assigned to a cloud, first reserve availability in this cloud will be checked by cloud scheduler. Then best-effort task can be pre-empted by AR task, the only case once most of resources are earmarked by some other AR task. Later there are

not enough resources left for this AR task in the obligatory time slot. If the AR task is not disallowed, which means there are enough resources obtainable for the task, a set of required VMs are selected randomly.

The estimated finish time of task may not be same as real finish time due to the resource argument within individual cloud. Later to adjust the resource allocation animatedly based on the latest available information writers in [2] propose an online adaptive scheduling process.

In future online adaptive procedure the remaining static resource distribution will be re-evaluate recurrently with a predefined incidence. In each re-evaluation, the schedulers will re-calculate the projected finish time of their tasks. Note that a scheduler of a assumed cloud will only reconsider the tasks that are in the jobs succumbed to this cloud, not the errands that are assigned to this cloud.

Algorithm 3 Cloud min-min scheduling (CMMS)

Require: A set of tasks, m different clouds ETM matrix

Ensure: A schedule generated by CMMS

1. For a mappable task set P
2. **While** there are tasks not assigned **do**
3. Update mappable task set P
4. **For** I = task $v_i \in$ P **do**
5. Send task check requests of v_i to all other cloud schedulers
6. Receive the earliest resource available time response and And list of task with their priorities form all other cloud scheduler
7. Find the cloud $C_{min}(v_i)$ giving the earliest finish time of v_i, assuming no other task preempts v_i
8. **End for**
9. Find the task-cloud pair (v_k, $C_{min}(v_k)$) with earliest finish time in the pairs generated in forloop
10. Assign task v_k to cloud $D_{min}(v_k)$
11. Remove v_k form P
12. Update the mappbale task set P
13. **End while**

IV. SCHEDULING ALGORITHM

In proposed importance based preparation algorithm we have adapted the scheduling experiential in [24] for executing highest priority task with advance reservation by pre-empting best-effort task as done in [11]. Algorithm 4 shows the quasi codes of priority based scheduling algorithm (PBSA).

Algorithm 4 Priority Based Scheduling Algorithm (PBSA)

1. **Input:** UserServiceRequest
2. //call Algorithm 2 to form the list of task based on priorities
3. get globalAvailableVMList and gloableUsedVMList and also availableResourceList from each cloud schedular
4. // find the appropriate VMList fromeach cloud scheduler
5. **if** AP(R,AR) != ϕ **then**
6. // call the algorithm 1 load balancer
7. deployableVm=load-balancer(AP(R,AR))
8. Deploy service on deployableVM
9. deploy=true
10. **Else if** R has advance reservation and best-effort task is running on any cloud **then**
11. // Call algorithm 3 CMMS for executing R with advance reservation
12. Deployed=true
13. **Else if** globalResourceAbleToHostExtraVM **then**
14. Start newVMInstance
15. Add VMToAvailbaleVMList
16. Deploy service on newVM
17. Deployed=true
18. **Else**
19. queue serviceReuest until
20. queueTime > waitingTime
21. Deployed=false
22. **End if**
23. If deployed then
24. return successful
25. terminate
26. **Else**
27. return failure
28. terminate

As exposed above in Algorithm 4, the customers' service deployment requests (R), which is calm of the SLA terms (S) and the application data (A) to be provisioned, is provided as input to scheduler (line 1 in Algorithm 1). Once service request (i.e.

job) reach at cloud scheduler, scheduler divide it in tasks as per there dependencies formerly the Algorithm 2 is called to procedure the list of tasks based to their importance (line 2). In the first step, it extracts the SLA terms, which forms the basis for finding the VM with the suitable resources for deploying the application. In next step, it collects the material about the number of running VMs in each cloud and the total available resources (AR) (line 3). Rendering to SLA terms appropriate VMs (AP) list is form, which are capable of provisioning the requested service (R) (lines 4-5).

When the list of suitable VMs is form, the Algorithm 1load-balancer chooses which particular VM is allocated to service appeal in order to equilibrium the load in the data center of each cloud (lines 6-9).

Once there is no VM with the suitable resources running in the data center of any cloud, the scheduler checks if service request (R) has advance registration then it search for best-efforts task running on any cloud or not, if it originate best-effort task then it calls Algorithm 3 CMMS for performing advance reservation request by pre-empting best-effort task (lines 1012). If no best-effort task is found on any cloud then scheduler forms whether the global capitals containing of physical capitals can crowd new VMs, if yes then, it mechanically starts new VMs with predefined resource sizes to provision service requests (lines 13-17). Then when global resources are not adequate to host extra VMs, the provisioning of service request is place in queue by the scheduler until a VM with suitable resources is available (lines 18-22). If after a certain period of time, the service requests can be scheduled and deployed, before scheduler returns a scheduling success to the cloud admin, then it returns failure (lines 23-28).

V. EXPERIMENTAL RESULTS

A. *Experiment setup*

We assess the performance of our priority based scheduling algorithm finished simulations. By different set of jobs simulation is done in 10 runs. In each run of imitation, we simulate a set of 70 different service requests (i.e. jobs), and each service appeal is composed of up to 18 sub-tasks. We reflect 4 clouds in the imitation. All 70 service requests will be succumbed to random clouds at random arrival time. Amongst these 70 service request, 15 requests are in the AR modes, though the rest are in the best effort modes, with different SLA objectives. The limits in Table 1 are set in simulation randomly rendering to their maximum and minimum values. Since we focus only on the preparation algorithms, we do our imitations locally without applying in any exiting cloud system or using VM interface API.

Table 1 RANGE OF PARAMETERS

Parameter	Minimum	Maximum
$ETM_{i,j}$	27	120
Number of VMs in a cloud	22	120
Number of CPU in a VM	1	8
Memory in a VM	40	2048
Disk space in VM	5000	100000
Speed of copy in disk	100	1000

We reflect two situations for arrival of service request. In first situation, called as loose state, we spread arrival time of request extensively over time so that appeal does not need to contend capitals in cloud. In other state we set influx time of requests close to each other, so recognised as tight situation. The time passes from request is submitted to the request is ended, is defined as execution time.

B. *Results*

Figure 2 shows the regular job execution time in loose situation. We find out that the PBSA algorithm has the minimum normal execution time. The resource contentions happen when best-effort job is pre-empted by AR job. As resource contention less in loose state, so that projected finish time of job is close to the actual finish time. Hence adaptive process does not impact the job implementation time significantly.

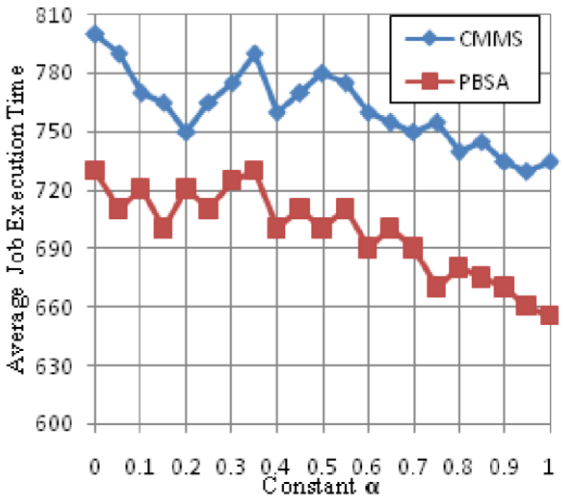

Figure 2. Average job execution time in loose situation

In figure 3 tight situation results are shown in which PBSA does better than CMMS. In tight state resource contention is more so the actual surface time of job is often later than estimated finish time. As AR job pre-empt best-effort job, the adaptive process with updated information works more meaningfully in tight situation.

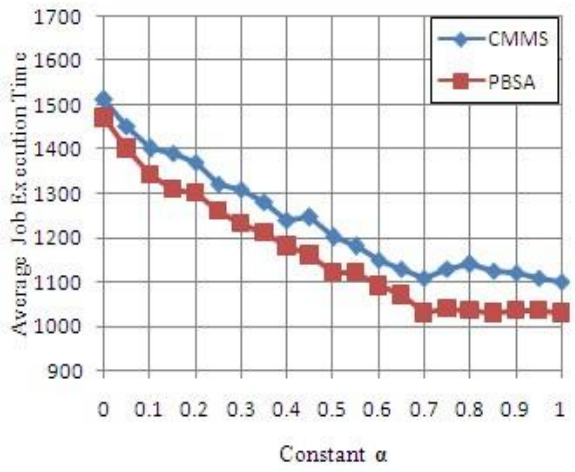

Figure 3. Average job execution time in tight situation

VI. CONCLUSIONS

In this paper, we present dynamic resource allocation mechanism for Pre-emptible jobs in cloud. We propose priority based algorithm, in which seeing multiple SLA objectives of job, for dynamic resource allocation to AR job by pre-empting best-effort job. Imitation results show that PBSA perform better than CMMS in resource contention situation.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] J. M. Wilson, "An algorithm for the generalized assignment problem with special ordered sets," Journal of Heuristics, 11(4):337–350, 2005.

[5] M. Qiu and E. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *ACM Transactions on Design Automation of Electronic Systems* (TODAES), vol. 14, no. 2, pp. 1–30, 2009.

[6] M. Qiu, M. Guo, M. Liu, C. J. Xue, and E. H.-M. S. L. T. Yang, "Loop scheduling and bank type assignment for heterogeneous multibank memory," Journal of Parallel and Distributed Computing(JPDC), vol. 69, no. 6, pp. 546–558, 2009.

[7] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," IEEE Transactions on Parallel and Distributed Systems, pp. 308–323, 2002.

[8] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," Parallel Computing, vol. 31, no. 7, pp. 653–670, 2005.

[9] "Adaptive Management of Virtualized Resources in Cloud Computing Using Feedback Control," in First International Conference on Information Science and Engineering, April 2010, pp. 99-102.

[10] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility Functions in Autonomic Systems," in ICAC '04: Proceedings of the First International Conference on Autonomic Computing. IEEE Computer Society, pp. 70–77, 2004.

[11] Jiayin Li, Meikang Qiu, Jian-Wei Niu, Yu Chen, Zhong Ming, "Adaptive Resource Allocation for Preempt able Jobs in Cloud Systems," in 10th International Conference on Intelligent System Design and Application, Jan. 2011, pp. 31-36.

[12] Yazir Y.O., Matthews C., Farahbod R., Neville S., Guitouni A., Ganti S., Coady Y., "Dynamic resource allocation based on distributed multiple criteria decisions in computing cloud," in 3rd International Conference on Cloud Computing, Aug. 2010, pp. 91-98.

[13] Goudarzi H., Pedram M., "Multi-dimensional SLA-based Resource Allocation for Multi-tier Cloud Computing Systems," in IEEE International Conference on Cloud Computing, Sep. 2011, pp. 324331.

[14] Shi J.Y., Taifi M., Khreishah A.,"Resource Planning for Parallel Processing in the Cloud," in IEEE 13th International Conference on High Performance and Computing, Nov. 2011, pp. 828-833.

[15] Aoun R., Doumith E.A., Gagnaire M., "Resource Provisioning for Enriched Services in Cloud Environment," IEEE Second International Conference on Cloud Computing Technology and Science, Feb. 2011, pp. 296-303.

[16] T. Erl, "Service-oriented Architecture: Concepts, Technology, and Design", Upper Saddle River, Prentice Hall, 2005.

[17] F. Chong, G. Carraro, and R. Wolter,"Multi-Tenant Data Architecture", Microsoft Corporation, 2006.

[18] E. Knorr, "Software as a service: The next big thing", InfoWorld, March 2006.

[19] V. Ungureanu, B. Melamed, and M.Katehakis,"Effective Load Balancing for Cluster-Based Servers Employing Job Preemption," Performance Evaluation, 65(8), July 2008, pp. 606-622.

[20] L. Aversa and A. Bestavros. "Load Balancing a Cluster of Web
Servers using Distributed Packet Rewriting", Proceedings of the 19th IEEE International Performance, Computing, and Communication Conference, Phoenix, AZ, Feb. 2000, pp. 24-29.

[21] V. Cardellini, M. Colajanni, P. S. Yu,"Dynamic Load Balancing on Web-Server Systems", IEEE Internet Computing, Vol. 33, May-June 1999 , pp. 28 -39.

[22] Chieu T.C., Mohindra A., Karve A.A., Segal A., "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in IEEE International Conference on e-Business Engineering, Dec. 2009, pp. 281-286.

[23] Naidila Sadashiv, S. M Dilip Kumar, "Cluster, Grid and Cloud Computing: A Detailed Comparison," The 6th International Conference on Computer Science & Education (ICCSE 2011) August 3-5, 2011. SuperStar Virgo, Singapore, pp. 477- 482.

[24] Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, Ivan Breskovic, "SLA-Aware Application Deployment and Resource Allocation in Clouds", 35th IEEE Annual Computer Software and Application Conference Workshops, 2011, pp. 298-303.

[25] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments," In High Performance Computing and Simulation Conference, pages 48 – 55 , Caen, France, 2010.

[26] M. Maurer, I. Brandic, V. C. Emeakaroha, and S. Dustdar, "Towards knowledge management in self-adaptable clouds," In 4th International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS'10) , Miami, Florida, USA, 2010.

[27] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," Parallel Computing, vol. 31, no. 7, pp. 653–670, 2005.

[28] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Non-identical Processors," Journal of the ACM, pp. 280–289, 1977.