Priority Based Dynamic resource allocation in Cloud Computing

Chandrashekhar S. Pawar
M. E. student
Department of Computer Engineering
R. C. Patel Institute of Technology
Shirpur, India
pawar.chandrashekhar09@yahoo.com

Rajnikant B. Wagh
Associate Professor
Department of Computer Engineering
R. C. Patel Institute of Technology
Shirpur, India
raj_wagh@rediffmail.com

***Abstract*- Today Cloud computing is on demand as it offers dynamic flexible resource allocation, for reliable and guaranteed services in pay-as-you-use manner, to Cloud service users. So there must be a provision that all resources are made available to requesting users in efficient manner to satisfy customer's need. This resource provisioning is done by considering the Service Level Agreements (SLA) and with the help of parallel processing. Recent work considers various strategies with single SLA parameter. Hence by considering multiple SLA parameter and resource allocation by preemption mechanism for high priority task execution can improve the resource utilization in Cloud. In this paper we propose an algorithm which considered Preemptable task execution and multiple SLA parameters such as memory, network bandwidth, and required CPU time. An obtained experimental results show that in a situation where resource contention is fierce our algorithm provides better utilization of resources.**

***Keywords*- Cloud computing, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Resource management, Software as a Service (SaaS), Virtual machine, Virtualization.**

I. INTRODUCTION

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

Cloud providers are able to attain the agreed SLA, by scheduling resources in efficient manner and by deploying application on proper VM as per the SLA objective and at the same time performance of the applications must be optimized. Presently, there exists a more work done on scheduling of applications in Clouds [1], [2], [3]. These approaches are usually considering one single SLA objective such as cost of execution, execution time, etc. Due to combinatorial nature scheduling algorithm with multiple SLA objective for optimal mapping of workload with multiple SLA parameters to resources is found to be NP-hard [4]. The available solutions are based on the use of heuristics.

When a job is submitted to the clouds, it is usually partitioned into several tasks. Following two questions are need to consider when applying parallel processing in executing these tasks: 1). how to allocate resources to tasks; 2) task are executed in what order in cloud; and 3) how to schedule overheads when VMs prepare, terminate 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] task scheduling and resource allocation have been studied.

Typically, efficient provisioning requires two distinct steps or processes: (1) initial static planning step: the initially group the set of VMs, then classify them and deployed onto a set of physical hosts; and (2) dynamic resource provisioning: the allocation of additional resources, creation and migration of VMs, dynamically responds to varying workload. Step 2 runs continuously at production time where in contrast Step 1 is usually performed at the initial system set up time and may only be repeated for overall cleanup and maintenance on a monthly or semi-annually schedule.

In this paper we focus on dynamic resource provisioning as mentioned above in step 2. In order to attain the agreed SLA objective our proposed algorithm dynamically responds to fluctuating work load by preempting the current executing task having low priority with high priority task and if preemption is not possible due same priority then by creating the new VM form globally available resources.

In section II, we discuss works related to this topic. In section III, models for resource allocation and task scheduling in IaaS cloud computing system are presented. We propose our algorithms in section IV, followed by experimental result in section V. Finally, we give the conclusion in section VI.

II. RELATED WORK

In [9] author proposed architecture, using feedback control theory, for adaptive management of virtualized resources, which is based on VM. In this VM-based architecture all hardware resources are pooled into common shared space in cloud computing infrastructure so that hosted application can access the required resources as per there need to meet Service Level Objective (SLOs) of application. The adaptive manager use in this architecture is multi-input multi-output (MIMO) resource manager, which includes 3 controllers: CPU controller, memory controller and I/O controller, its goal is regulate multiple virtualized resources utilization to achieve SLOs of application byusing control inputs per-VM CPU, memory and I/O allocation.

The seminal work of Walsh et al. [10], proposed a general two-layer architecture that uses utility functions, adopted in the context of dynamic and autonomous resource allocation, which consists of local agents and global arbiter. The responsibility of local agents is to calculate utilities, for given current or forecasted workload and range of

resources, for each AE and results are transfer to global arbiter. Where, global arbiter computes near-optimal configuration of resources based on the results provided by the local agents. In global arbiter, the new configurations applied by assigning new resources to the AEs and the new configuration computed either at the end of fixed control intervals or in an event triggered manner or anticipated SLA violation.

In [11], authors propose an adaptive resource allocation algorithm for the cloud system with preempt able tasks in which algorithms adjust the resource allocation adaptively based on the updated of the actual task executions. Adaptive list scheduling (ALS) and adaptive min-min scheduling (AMMS) algorithms are use for task scheduling which includes static task scheduling, for static resource allocation, is generated offline. The online adaptive procedure is use for re-evaluating the remaining static resource allocation repeatedly with predefined frequency. In each re-evaluation process, the schedulers are re-calculating the finish time of their respective submitted tasks, not the tasks that are assign to that cloud.

The dynamic resource allocation based on distributed multiple criteria decisions in computing cloud explain in [12]. In it author contribution is tow-fold, first distributed architecture is adopted, in which resource management is divided into independent tasks, each of which is performed by Autonomous Node Agents (NA) in ac cycle of three activities: (1) VMPlacement, in it suitable physical machine (PM) is found which is capable of running given VM and then assigned VM to that PM, (2) Monitoring, in it total resources use by hosted VM are monitored by NA, (3) In VMSelection, if local accommodation is not possible, a VM need to migrate at another PM and process loops back to into placement. And second, using PROMETHEE method, NA carry out configuration in parallel through multiple criteria decision analysis. This approach is potentially more feasible in large data centers than centralized approaches.

The problem of resource allocation is considered in [13], to optimize the total profit gained from the multi-dimensional SLA contracts for multi-tire application. In it the upper bound of total profit is provided with the help of force-directed resource assignment (FRA) heuristic algorithm, in which initial solution is based on provided solution for profit upper bound problem. Next, distribution rates are fixed and local optimization step is use for improving resource sharing. Finally, a resource consolidation technique is applied to consolidate resources to determine the active (ON) servers and further optimize the resource assignment.

Using steady state timing models, this [14] paper reports a study of cloud HPC resource planning. In it author propose quantitative application dependent instrumentation method to investigate multiple important dimensions of a program's scalability. Sequential and parallel timing model with program instrumentations can reveal architecture specific deliverable performances that are difficult to measure otherwise. These models are introduces to connect multiple dimensions to time domain and application speed up model is use to tie these models in same equation. The ability to explore multiple dimension of program quantitatively, to gain non-trivial insight. For target processing environment authors use Amazon EC2.

In previous years, the aims of distributed system have been centered on the decoupling of interfaces from service oriented architectures (SOA) [16], implementation, subscription model, hosting models and social collaboration. Recently, Internet-based distributed, multi-tenant [17] applications connective to internal business applications, known as software as a service (SaaS) [18], are gaining popularity. The previous work [19-21] on web application scalability implemented for static load balancing solution with server clusters but the dynamic scaling of web applications in virtualized cloud computing has not been much discussed. Because such kinds of work load require minimum response time and high level of availability and reliability from web applications.

A solution for dynamic scaling of web application provided in [22] by describing an architecture to scale web application in dynamic manner, based on threshold in a virtualized cloud computing environment. Architecture 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 routing and balancing user requests to web application deployed on Apache HTTP server that are installed in Linux virtual machine. As per the demand these virtual machines are started and provisioned by a provisioning sub-system. But the action of provisioning and de-provisioning of web server virtual machine instances control by a dynamic scaling algorithm based on relevant threshold of web application.

III. USE TECHNIQUES

In this section we are describing SLA based resource provisioning and online adaptive scheduling for Preemptable task execution, these two methodologies which are combined in proposed algorithm for effective utilization of cloud resources to meet the SLA objective.

A. Cloud Resource provisioning and scheduling heuristic

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

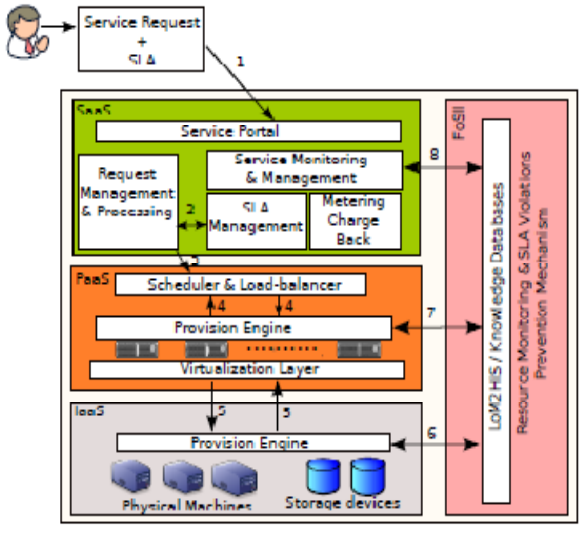

Figure 1. Cloud Provisioning and Deployment model

Service deployment requests from customers is place to the service portal (step 1 in Figure1), which forward the requests to the request management and processing component to validate the requests with the help of SLA(step 2). If the request is valid, it is then pass to the scheduler and load-balancer (step 3). For deploying the requested service, scheduler selects the appropriate VMs, as per SLA and priority, through the provisioning engine in PaaS layer and the load-balancer balances the service provisioning among the running VMs (step 4). The VMs on the virtualization layer manage by provision engine and the virtualization layer interacts with the physical resources with the help of the provision engine in IaaS layer (step 5).

The LoM2HiS framework monitors the low-level resource metrics of the physical resources at IaaS layer [25] (step 6). If SLA violation occurs, reactive actions provide by the knowledge database techniques [26] in FoSII (step 7). The requested service status and the SLA information are communicated back with the service portal (step 8).

Provisioning can be done at the single layers alone. However, approach which we considered in [24] aims to provide an integrated resource provisioning strategy. Thus, scheduling heuristics in [24] considers the three layers.

An aim of scheduling heuristic in [24] is to schedule job on VMs based on the agreed SLA objective and creating new VMs on physical resources based on availabilities resources. This strategy helps to optimized application performance and at the same time reducing the possibilities of SLA violations. And, the integrated load-balancer (Algorithm 1 Load-balancer given below) in the heuristic ensures high and efficient resource utilization in the Cloud environment.

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

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

Algorithm 1 Load Balancer

1. Input: AP(R,AR)
2. availableVMList //list of available VMs form each cloud
3. usedVMList //list of VMs,currently provision to certain job
4. deployableVm=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

Therefore, the first VM from the appropriate and available VM list can be selected for the deployment of the new job request. Lastly, the selected VM will be added to the list of used VMs so that the load-balancer will not select it in the next iteration (lines 8-15).

B. Preemptable task execution

When a scheduler receives customer's service request, it will first partition that service request into tasks in the form of a DAG. Then initially static resource allocation is done. In [11] authors proposed two greedy algorithms, to generate the static allocation: 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 definitions used for listing the task are provided as follow. The earliest start time (EST) and the latest start time (LST) of a task are shown as in (1) and (2).The entry-tasks have EST equals to 0. And 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 concerned in [11] is heterogeneous the execution time of each task on VMs of different clouds are not the same. AT(v_i) is the average execution time of task v_i. The critical node (CN) is a set of vertices having equal EST and LST in the DAG. Algorithm 2 shows a function forming a task list based on the priorities.

Algorithm 2 Forming 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 and cloud resources allocated to them are available, the assigned task will start its execution. This task is removed from the list after 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 dependencies among tasks not considered in original min-min algorithm. So in the dynamic min-min algorithm used in [2], authors maintain the task dependencies by updating the map able task set in every scheduling step. The tasks whose predecessor tasks are all assigned are placed in the map able task set. Algorithm 3 shows the pseudo codes of the CMMS algorithm.

A cloud scheduler record execution schedule of all resources using a slot. When an AR task is assigned to a cloud, first resource availability in this cloud will be checked by cloud scheduler. Since best-effort task can be preempted by AR task, the only case when most of resources are reserved by some other AR task. Hence there are not enough resources left for this AR task in the required time slot. If the AR task is not rejected, which means there are enough resources available for the task, a set of required VMs are selected arbitrarily.

The estimated finish time of task may not be same as actual finish time due to the resource contention within individual cloud. Hence to adjust the resource allocation dynamically based on the latest available information authors in [2] propose an online adaptive scheduling procedure.

In proposed online adaptive procedure the remaining static resource allocation will be re-evaluate repeatedly with a predefined frequency. In each re-evaluation, the schedulers will re-calculate the estimated finish time of their tasks. Note that a scheduler of a given cloud will only re-evaluate the tasks that are in the jobs submitted to this cloud, not the tasks 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 for-loop
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 priority based scheduling algorithm we have modified the scheduling heuristic in [24] for executing highest priority task with advance reservation by preempting best-effort task as done in [11]. Algorithm 4 shows the pseudo 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 shown above in Algorithm 4, the customers' service deployment requests (R), which is composed of the SLA terms (S) and the application data (A) to be provisioned, is provided as input to scheduler (line 1 in Algorithm 1). When service request (i.e. job) arrive at cloud scheduler, scheduler divide it in tasks as per there dependencies then the Algorithm 2 is called to form the list of tasks based to their priority (line 2). In the first step, it extracts the SLA terms, which forms the basis for finding the VM with the appropriate resources for deploying the application. In next step, it collects the information about the number of running VMs in each cloud and the total available resources (AR) (line 3). According to SLA terms appropriate VMs (AP) list is form, which are capable of provisioning the requested service (R) (lines 4-5).

Once the list of appropriate VMs is form, the Algorithm 1-load-balancer decides which particular VM is allocated to service request in order to balance the load in the data center of each cloud (lines 6-9).

When there is no VM with the appropriate resources running in the data center of any cloud, the scheduler checks if service request (R) has advance reservation then it search for best-effort task running on any cloud or not, if it found best-effort task then it calls Algorithm 3 CMMS for executing advance reservation request by preempting best-effort task (lines 10-12). If no best-effort task is found on any cloud then scheduler checks whether the global resources consisting of physical resources can host new VMs, if yes then, it automatically starts new VMs with predefined resource capacities to provision service requests (lines 13-17). But when global resources are not sufficient to host extra VMs, the provisioning of service request is place in queue by the scheduler until a VM with appropriate resources is available (lines 18-22). If after a certain period of time, the service requests can be scheduled and deployed, then scheduler returns a scheduling success to the cloud admin, otherwise it returns failure (lines 23-28).

V. Experimental Results

A. Experiment setup

We evaluate the performance of our priority based scheduling algorithm through simulations. With different set of jobs simulation is done in 10 runs. In each run of simulation, we simulate a set of 70 different service requests (i.e. jobs), and each service request is composed of up to 18 sub-tasks. We consider 4 clouds in the simulation. All 70 service requests will be submitted to random clouds at arbitrary arrival time. Among these 70 service request, 15 requests are in the AR modes, while the rest are in the best-effort modes, with different SLA objectives. The parameters in Table 1 are set in simulation randomly according to their maximum and minimum values. Since we focus only on the scheduling algorithms, we do our simulations locally without implementing 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 consider two situations for arrival of service request. In first situation, called as loose situation, we spread arrival time of request widely over time so that request does not need to contend resources in cloud. In other situation we set arrival time of requests close to each other, so known as tight situation. The time elapses from request is submitted to the request is finished, is defined as execution time.

B. Results

Figure 2 shows the average job execution time in loose situation. We find out that the PBSA algorithm has the minimum average execution time. The resource contentions occur when best-effort job is preempted by AR job. As resource contention less in loose situation, so that estimated finish time of job is close to the actual finish time. Hence adaptive procedure does not impact the job execution time significantly.

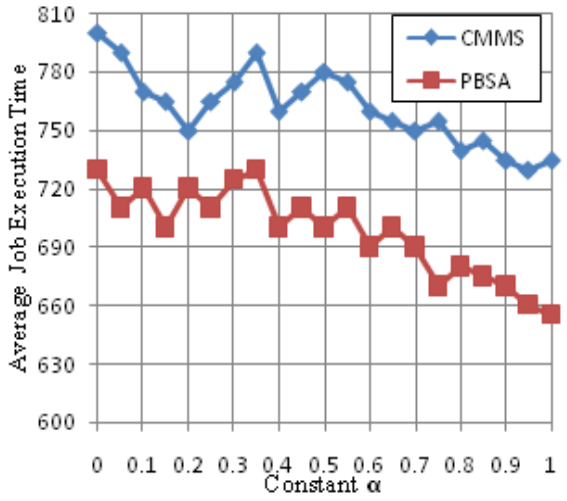

Figure 2. Average job execution time in loose situation

In figure 3 tight situation results are shown in which PBSA performs better than CMMS. In tight situation resource contention is more so the actual finish time of job is often later than estimated finish time. As AR job preempt best-effort job, the adaptive procedure with updated information works more significantly in tight situation.

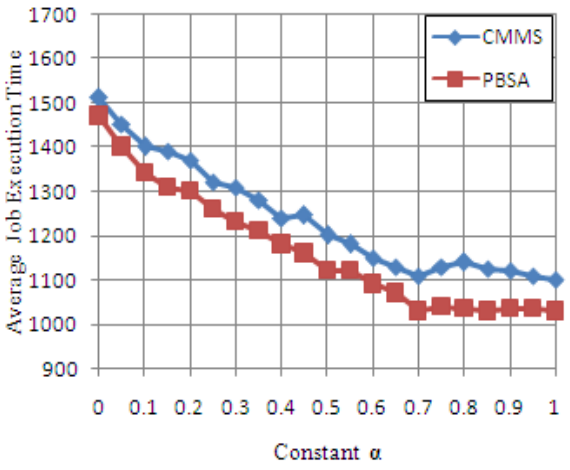

Figure 3. Average job execution time in tight situation

VI. CONCLUSIONS

In this paper, we present dynamic resource allocation mechanism for Preemptable jobs in cloud. We propose priority based algorithm, in which considering multiple SLA objectives of job, for dynamic resource allocation to AR job by preempting best-effort job. Simulation 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. 324-331.

[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.