Asian Journal of Information Technology 15 (14): 2392-2398, 2016
ISSN: 1682-3915

Dynamic VM Allocation Using Adaptive Map Reduce Algorithm in Cloud Computing

[1]N. Senthamarai and [2]M. Vijayalakshmi
[1]SRM Easwari Engineering College, SRM University, Ramapuram, Chennai, Tamil Nadu, India
[2]DIST, College of Engineering Guindy (CEG), Anna University, Chennai, India

Abstract: Cloud computing is a most popular technology because everything like hardware, platform and software are provided as a service. It is an improving area in research which includes load balancing, virtualization and storage etc. Load balancing distributes workloads across various resources such as computers, a computer cluster and network links. The load balancing is to optimize resource usage, maximize throughput, minimize response time and avoid overload. In a cluster system, allocating resources is a critical but challenging issue. A new load balancing policy named AMRA is used which attempts to partition jobs according to the user traffic and system load and improve the performance benefits. The idea of AMRA is to allocate the job to all the servers using the system loads as parameters and dynamically tune the job size boundaries based on the current user traffic loads. AMRA then directs the jobs whose job size lie in the same boundary size to the corresponding servers. AMRA always gives high priority to small jobs and send them to the less loaded servers. The algorithm evaluates the sequential and parallel jobs based on the inputs lang. This study mainly focuses on dynamically balanced load and increases the performance of the system and reduces the overhead.

Key words: Load balancing, cluster systems, dynamic load balancing, AMRA, resource allocation

INTRODUCTION

Cloud computing is distributed computing that uses computers that are interconnecting through a real-time network like the internet. Cloud computing enables well-situated, on-demand, dynamic and reliable utilization of distributed computing assets. Cloud computing is an on demand service in which shared resources work together to perform a task to get the results in minimum possible time by the distribution of any dataset among all the connected processing units. Such virtual servers do not exist physically so they can be scaled up and down at any time (Fig. 1).

Characteristics: Cloud computing exhibits several characteristics (Haryani and Jagli, 2014).

On-demand self-service: Cloud service providers such as Amazon Web Services (AWS), Microsoft, Google, IBM and Salesforce.com provides services such as applications, email, network or server service with no human interaction with every service provider.

Broad network access: Cloud capabilities offered over the network are accessed through standard mechanisms that encourage use by mixed thin or thick client platforms such as mobile phones, laptops along with PDAs.

Resource pooling: The provider is able to pool computing resources such as storage space, processing, memory, network bandwidth, virtual machines. Email services are pooled together to supply multiple clients using multiple-tenant model with diverse physical and virtual resources dynamically assigned and reassigned according to end user demand.

Rapid elasticity: Cloud can provision services quickly and elastically, automatically to swiftly scale out and rapidly released to scale quickly in.

Measured service: Cloud computing source usage can be measured, controlled and reported to both the provider and consumer to the utilized service in a transparent manner. Cloud computing services apply a metering ability which enables to control and optimize resource use based on pay per use policy.

Necessity of load balancing: Load balancing (Haryani and Jagli, 2014) is a computer network method for distributing workloads across multiple computing resources; for example, computers network links a computer cluster, disk drives or central processing units. Load balancing plans to optimize maximize throughput, resource use, minimize response time and evade overload of any one of the resources.

Corresponding Author: N. Senthamarai, SRM Easwari Engineering College, SRM University, Ramapuram, Chennai, Tamil Nadu, India

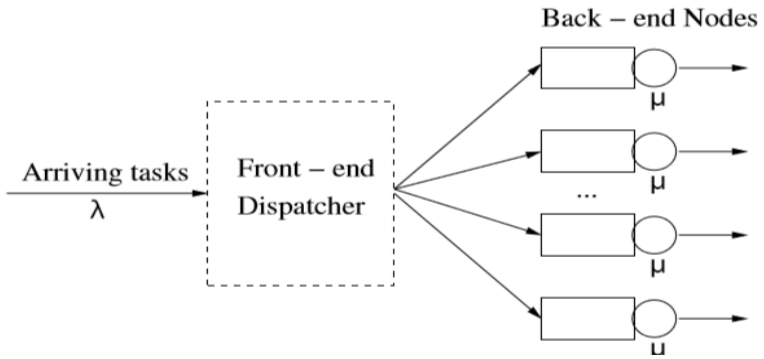

Fig. 1: Model of a clustered server

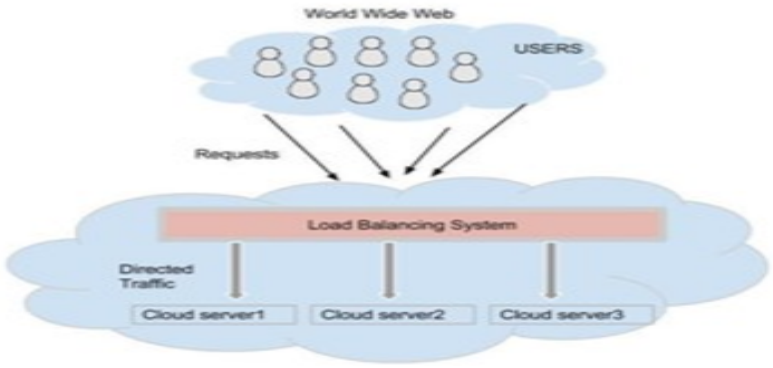

Fig. 2: Load Balancing system in cloud computing

Load balancing in the cloud differs from load-balancing architecture and implementation by using commodity servers to perform the load balancing because it's difficult to predict the number of requests that will be issued to a server. Load balancing is an important challenge in cloud computing. It is a mechanism that distributes the dynamic local workload evenly across all the nodes in the whole cloud to avoid a situation where some nodes are heavily loaded while others are idle or doing little work. It helps to attain a high customer satisfaction and resource utilization ratio, consequently improving the overall performance and resource utility of the system (Fig. 1).

Dynamic load balancing in cloud computing: Figure 2 dynamic load balancing algorithm (Haryani and Jagli, 2014) assumes no previous knowledge about job actions or the global state of the system, i.e., this load balancing decisions are exclusively based on the existing or current state of the system. In the distributed one, the dynamic load balancing algorithm is executed by all nodes present in the system and the various tasks of load balancing are shared among them. The interaction among nodes to realize load balancing can take two forms:

- Cooperative
- Non-cooperative

In the cooperative, the nodes work side-by-side to attain a common goal, such as, to advance the overall response time, etc. In the non-cooperative, every node works independently in the direction of a goal local to it, i.e. to improve the response time of a local task. The advantage of distributed type is that even if one or more nodes in the arrangement fail, it will not cause the total load balancing process to stop; it instead would influence the system performance to a little extent. In non-distributed type, either one node or a group of nodes perform the task of load balancing. Dynamic load balancing algorithms of non-distributed nature can be of two forms:

- Centralized
- Semi-distributed

In the centralized form, the load balancing algorithm is executed just by a single node in the total system which is called the central node. This node is exclusively in charge of balancing the load in the whole system. The other nodes interact merely with the central node. In semi-distributed form, nodes are partitioned into clusters and the load balancing in each cluster is of centralized form. A central node can be chosen in each cluster that takes care of load balancing inside that particular cluster. Hence, the load balancing of the complete system is done

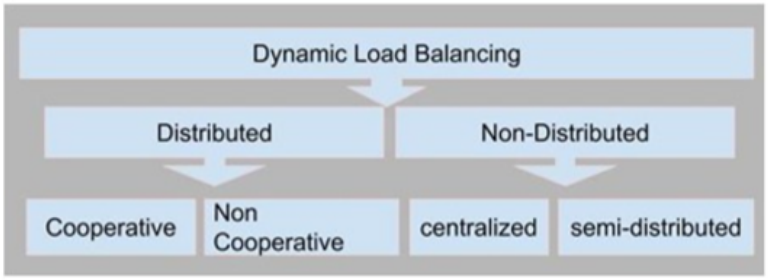

Fig. 3: Summarizing dynamic load balancing techniques

via the central nodes of each cluster. Centralized dynamic load balancing form takes only fewer messages to arrive at a decision as the number of overall interactions in the system decreases when compared with the semi-distributed case. Hence, centralized algorithms can create a bottleneck in the system at the central node and also the load balancing process is rendered hopeless once the central node crashes. Therefore, this algorithm is mainly suited for networks with small size (Teo and Ayani, 2001; Harchol-Balter and Downey, 1997) (Fig. 3).

Various dynamic load balancing policies

Honeybee foraging behavior: This algorithm proposed a decentralized honeybee-based load balancing technique that is a nature-inspired algorithm for self-organization. In this algorithm, the servers are grouped under Virtual Servers (VS) and each VS is having its virtual service queue. Each server processing a request from its queue calculates a profit or reward. If this profit high, then the server stays at the current virtual server, otherwise the server returns to the forage. Global load balancing is achieved through local server actions. System performance is also enhanced with increased system but throughput is not increased with an increase in system size.

LBVFT (Load Balancing Technique for Virtualization and Fault Tolerance): LBVFT proposed a Load Balancing Technique for Virtualization and Fault Tolerance in Cloud Computing to assign the tasks to the virtual nodes depending on the Success Rates (SR) and the previous load history. In load assigning technique, assignment of a load is done by the Load Balancer (LB) of the Cloud Manager (CM) module by higher success rate and the lower load of the available nodes. It enhances the performance of the system (Mi *et al*., 2006).

VFT (Virtualization and Fault Tolerance): A VFT technique used to reduce the service time and to increase the system availability utilizes a Cloud Manager (CM) module and a Decision Maker (DM) to manage the load balancing, virtualization and to handle the faults. As a first step virtualization and load balancing are done and as a second step fault tolerance is achieved by redundancy and fault handler. VFT is designed for providing better fault tolerance (Das and Khilar, 2013).

Global heat diffusion algorithm: The algorithm proposed two efficient delay adjustment schemes to address the latency problem. The uniform adjustment scheme performs a uniform distribution of the load variation among the neighbor servers. The adaptive adjustment scheme performs a limited degree of user tracking but without the need to communicate with neighbor servers. The global heat diffusion algorithm is used to avoid the delay and increase the performance of the system (Ray and Sarkar, 2012).

Random: Under the Random policy, (Tai *et al*., 2014) a server is chosen uniformly at Random to serve each incoming job by treating all jobs with the same priority. Since this algorithm does not need to keep any history information of jobs, it is widely implemented in real systems because of its simplicity.

Round robin: This policy assigns each request to different servers in a circular order, handling all requests without any priority. Round Robin algorithm is simple and easy to implement. As this policy does not consider any characteristics of requests, it can cause unbalanced load for each server when job sizes vary a lot (Zaharia *et al*., 2008).

Join Shortest Queue (JSQ): During the arrival of a new job, it believes that a server with the least number of jobs in the queue (i.e., the shortest queue) would complete that job in the shortest time. Hence, the newly arrived job should be assigned to that particular server.

Adapt load: It builds the histogram of job sizes and partitions the work into equal areas. Each server will be dedicating to process jobs with similar sizes. The size interval boundaries are adjusted dynamically according to the history. ADAPTLOAD has been proven to achieve high performance by reducing the number of small jobs from waiting behind large ones (Tai *et al*., 2014).

Efficient and Enhanced Algorithm (EEA): In this study, a new enhanced and efficient scheduling algorithm is proposed and then implemented in cloud computing environment using CloudSim toolkit. Here, when the VM finishes the processing the request and the Datacenter Controller receives the cloudlet response, it notices the load balancer of the VM de-allocation. The Load Balancer updates the status of VM in allocation table and easily calculates the expected response time. So the overall response time and data centre processing time is improved (Sharma and Banga, 2013).

Literature review: Mi *et al*. (2006) has proposed a two-step resource allocation policy that makes resource assignment decisions. First, instead of equally dispatching the work among all servers in the cluster, this policy biases load balancing by an effort to reduce performance loss due to autocorrelation in the streams of jobs that are direct to each server. As a second step, per-class bias guides resource allocation according to different priorities of the class. As a result, not all servers are equally utilized (i.e., the load in the system becomes unbalanced) but performance benefits are significant and service differentiation is achieved as shown by detailed trace-driven simulations.

Zhang *et al*. (2005) proposed a scheduling policy, ADAPTLOAD, monitors the incoming workload and self-adjusts its balancing parameters according to changes in the operational environment such as rapid fluctuations in the arrival rates or document popularity.

Mi *et al*. (2007) presented an analysis of the performance effects of burstiness in multi-tiered systems. Finally, they analyzed an admission control algorithm that takes autocorrelation into account and improves performance by reducing the long tail of the response time distribution.

Llorente *et al*. (2007) presented a VM migration is an important problem in modern data centers and the migration strategy greatly affects the datacenter performance. Migrations can be performed to optimize the behavior of the data center in many different dimensions. A possibility is to compact the load as much as possible to turn off the greatest amount of PMs possible. Another option is to equalize the load among the different PMs, to reject external disturbances in the best possible way.

Problem formulation and solution

New load balancing policy: AMRA: Our new load balancing policy, AMRA (Tai *et al*., 2014) which adaptively distributes work among all servers by taking account of both user traffic and system load, aiming to inherit the effectiveness of JSQ and ADAPTLOAD and meanwhile overcome the limitations of these two policies as shown in the previous section. The idea of AMRA is to rank on-the-fly all the servers according to their system loads and dynamically tune the job size boundaries based on the current user traffic loads. AMRA directs the jobs whose sizes lie in the same size boundary to the corresponding ranked servers. The dispatcher has known the size of each waiting job. Based on the observation that the majority of jobs in a heavy-tailed workload is small, AMRA gives small jobs high priority by sending them to the highly ranked, (i.e., less loaded) servers. ADAPTLOAD evenly balances the load across the entire cluster by determining boundaries of jobs sizes for each server. AMRA adopts such boundaries where the histogram of job sizes is built and partitioned into N equal areas for N servers in the system. Then, the ith server is responsible for the work locating in the ith area which allows the decreasing variation in job sizes (or job service times) on each server. Consequently, the proportion of small jobs that wait behind long ones is reduced as well and the user traffic is thus well balanced among all servers.

AMRA periodically ranks all servers based on their present system loads, e.g., server utilization or weighted queue lengths, and keeps sending the incoming jobs of similar sizes to a server with the same ranking instead of the same server that might be overloaded by the previously arrived jobs. As a result, AMRA further successfully balances the system load among all N servers and thus significantly diminishes the proportion of similar sized jobs (especially small ones) being queued on the same server during a short period. To re-rank all servers, our policy can either use server utilization (i.e., the percentage of time during which the server is busy) or weighted queue length (i.e., the length of each queue that is weighed by the size of queued jobs) to represent the load of each server. Higher weighted queue length or higher system utilization indicates the heavier load on that particular server. It follows that our policy only needs the

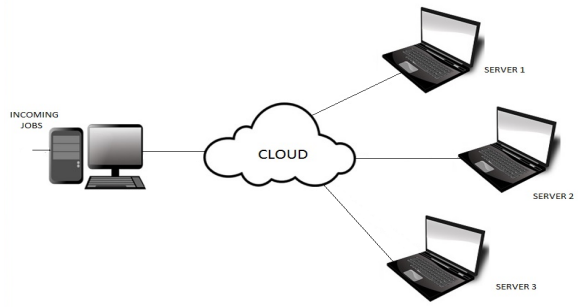

Fig. 4: Architecture

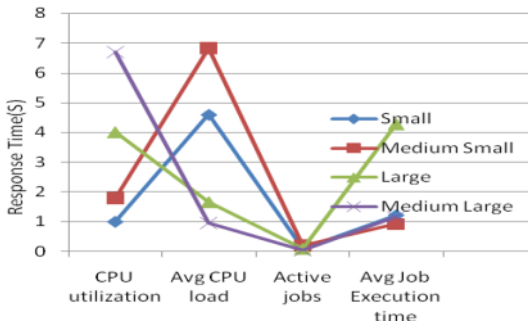

Fig. 5: Performance; vode vs CPU usage

on-line measured load information for updating the rankings of servers and thus can be used to balance the load on the fly.

Implementation: The architecture in Fig. 4 includes providing the jobs as input to the system which implements AMRA algorithms in which the jobs are categorized based on the size of jobs into four categories. Depending on the size, the jobs are fed into servers that are rank according to the system load. Small jobs are give high priority and are sent to highly ranked servers, i.e., less loaded servers.

AMRA uses the size of the partitioned jobs and the load factor of the servers to balance the load. The algorithm evaluates the size of the partitioned jobs with the load of the server and allocates equal size jobs to the same ranked server.

Algorithm for AMRA:

```
Class MAPPER
Method MAP (docid, doc d)
For all term w_ doc d do
For all term u_smalljob (w1) do
For all term u_mediumsmalljob (w2) do
For all term u_mediumlargejob
For all term u_largejob (w4) do
EMIT (pair (w1, u), count 1)
(Emit count for   each small
EMIT (pair (w2, u), count 1)
(Emit count for each medium small job)
EMIT (pair (w3, u), count 1)
(Emit count for each medium large job)
EMIT (pair (w4, u), count 1)
(Emit count for each large job)
Class REDUCER
Method REDUCE (pair p, counts [c1, c2 . . .])
S ? 0
For all count c _ counts [c1, c2…] do
S ? s + c                    (sum co-occurrence counts)
EMIT (pair p, count s)
```

Experiment results: Implementation of AMRA to achieve Dynamic load balancing is described as.

RESULTS AND DISCUSSION

Initialization of gridgain tool: The nodes are initialized using grid gain. The 'ggstart.bat' command helps to start up the gridgain tool using a command prompt. Initially, three nodes are initialized and then sent Small, Medium Small, Medium Large and Large.

Creation of nodes: The nodes are created using the gridgain tool and these nodes are interconnected in such a way that the availability and unavailability of other nodes are found in a particular node. If any active node fails, it can be identified by other nodes. The jobs are distributed to the creation of three nodes.

Performance evaluation

Code deployment: The code is deployed to other nodes from the User Interface and the User interface itself is an independent node. The system load is determined and divided into 4 nodes.

Performance: node vs CPU usage: After the code is deploying, the current detail of the server is displayed below. The bar chart explains the performance, i.e., CPU usage vs Node which is shown in Fig. 5.

Nodes restart and auto deployment: Clicking the restart button restarts all the nodes and the codes are auto-deployed. The system is configuring with the remote start/stop feature.

Job allocation: The incoming jobs are allocating to the nodes based on their CPU usage. Small jobs are allocating to less loaded nodes and large jobs are allocating to heavily loaded nodes. This can be shwon in Fig. 6-10.

Parallel job allocation: Jobs are allocating in parallel to all the nodes that are active.

Performance analysis of sequential and parallel job allocation: Performance Analysis of Sequential and Parallel Scheduling is show in Fig. 11. Here, Sequential

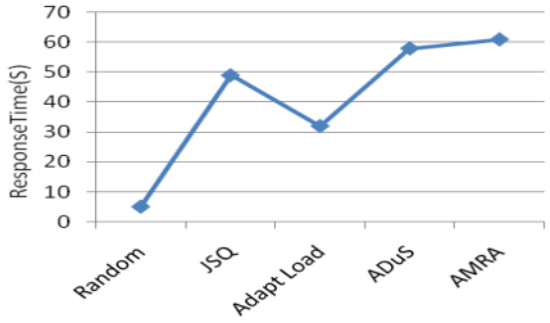

Fig. 6: Small job

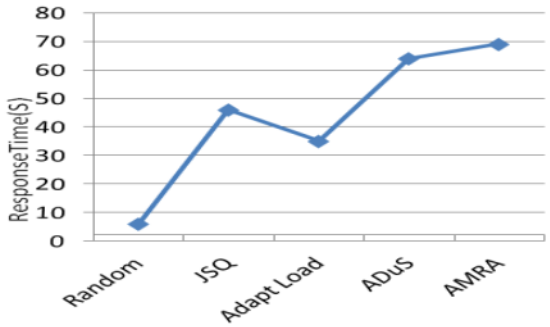

Fig. 7: Medium small job

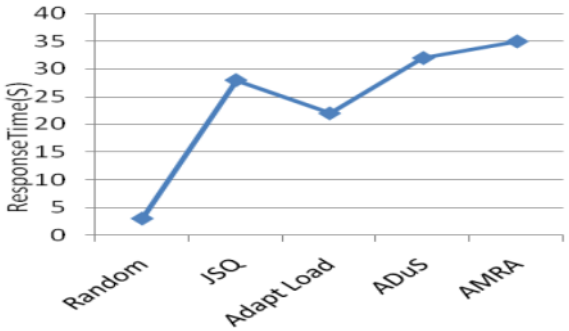

Fig. 8: Medium large job

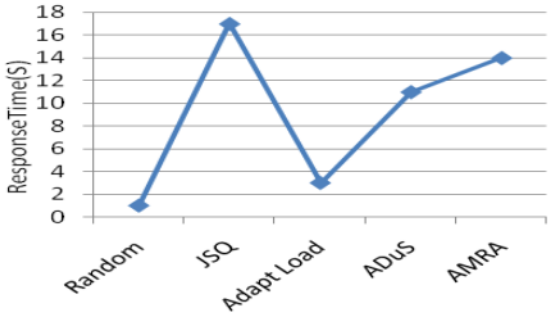

Fig. 9: Large job

scheduler requires more time when compared with the parallel scheduler to perform the same task. Hence, parallel scheduler provides better performance when compared with the sequential scheduler.

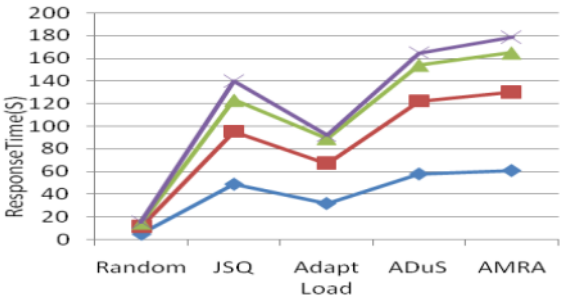

Fig. 10: Camparison od small, medium small; medium large and large

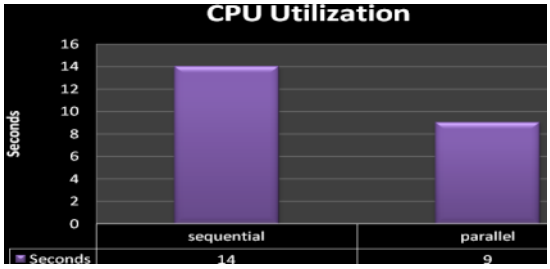

Fig. 11: Performance analysis

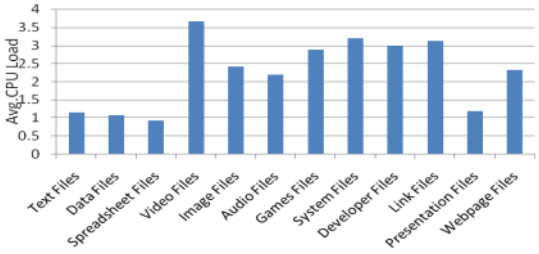

Fig. 12: Comprison of different files

Heavy tasks: As our system is designed for heavy-tailed work load condition, the system is checked to work under bursty workloads using heavy inputs such as images. Here we perform face detection tasks with images. The input is fetched from the input directory stored in the temp folder. The output of face detection is stored in the output directory in the temp folder. The composition of different files are shown in Fig. 12.

CONCLUSION

The various scenarios of Load balancing using AMRA in Cloud Computing have been studied in detail. Also recent study in these fields were explored. We thus proposed a new load balancing policy AMRA which distributes the work in the system by taking account of both user traffic and system load parallel. Using trace-driven simulations on synthetic and real traces, we showed that AMRA inherits the effectiveness of JSQ and

size-based policies and meanwhile overcomes their limitations which results in significant performance benefits. We also showed that AMRA can quickly adapt to the workload changes by monitoring user traffic and system load, repeatedly ranking the servers and partitioning the work in an on-line fashion. This study involves dynamically allocating virtual machines to the incoming jobs parallel.

RECOMMENDATIONS

In the future, we will refine our new load balancing algorithm such that it can self-adjust its parameters (e.g., the window size) to transient work load conditions. We will further implement this new policy in real systems such as clouds and data centers. We expect that this implementation of AMRA will provide a simple yet effective approach for resource allocation in large cluster environments.

REFERENCES

Das, P. and P.M. Khilar, 2013. VFT: A virtualization and fault tolerance approach for cloud computing. Proceedings of the 2013 IEEE Conference on Information and Communication Technologies (ICT), April 11-12, 2013, IEEE, JeJu Island, ISBN: 978-1-4673-5759-3, pp: 473-478.

Harchol-Balter, M. and A.B. Downey, 1997. Exploiting process lifetime distributions for dynamic load balancing. ACM. Trans. Comput. Syst., 15: 253-285.

Haryani, N. and D. Jagli, 2014. Dynamic method for load balancing in cloud computing. IOSR. J. Comput. Eng., 16: 23-28.

Lloriente, I.M., R.S. Montero, B. Sotomayor, B. Breitgand and D. Maraschini, 2011. On the Management of Virtual Machines for Cloud Infrastructures. In: Cloud Computing: Principles and Paradigms. Cloud Computing: Principles and Paradigms. John Wiley & Sons, New York, USA., pp: 157-191.

Mi, N., Q. Zhang, A. Riska and E. Smirni, 2006. Load balancing for performance differentiation in dual-priority clustered servers. Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, September, 11-14, 2006, IEEE, Riverside, California, ISBN: 0-7695-2665-9, pp: 385-394.

Mi, N., Q. Zhang, A. Riska, E. Smirni and E. Riedel, 2007. Performance impacts of autocorrelated flows in multi-tiered systems. Perform. Eval., 64: 1082-1101.

Ray, S. and A.D. Sarkar, 2012. Execution analysis of load balancing algorithms in cloud computing environment. Int. J. Cloud Comput. Serv. Archit., 2: 1-13.

Sharma, T. and V.K. Banga, 2013. Efficient and enhanced algorithm in cloud computing. Int. J. Soft Comput. Eng., 3: 385-390.

Tai, J., Z. Li, J. Chen and N. Mi, 2014. Load balancing for cluster systems under heavy-tailed and temporal dependent workloads. Simul. Modell. Pract. Theory, 44: 63-77.

Teo, Y.M. and R. Ayani, 2001. Comparison of load balancing strategies on cluster-based web servers. Simul., 77: 185-195.

Zaharia, M., A. Konwinski, A.D. Joseph, R.H. Katz and I. Stoica, 2008. Improving map reduce performance in heterogeneous environments. Proceedings of the 8th Symposium on Operating Systems Design and Implementation, December 12, 2008, USENIX Association, San Diego, California, pp: 1-7.

Zhang, Q., A. Riska, W. Sun, E. Smirni and G. Ciardo, 2005. Workload-aware load balancing for clustered web servers. IEEE. Trans. Parallel Distrib. Syst., 16: 219-233.