Simulation Modelling Practice and Theory 108 (2021) 102275

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

CGP: Cluster-based gossip protocol for dynamic resource environment in cloud

Shashank Srivastava [a], Sandeep Saxena [b], Rajkumar Buyya [c], Manoj Kumar [d], Achyut Shankar [e,*], Bharat Bhushan [f]

[a] Motilal Nehru National Institute of Technology, Allahabad, India
[b] Galgotias College of Engineering and Technology, Greater Noida, India
[c] Cloud Computing and Distributed Systems (CLOUDS) Lab School of Computing and Information Systems, The University of Melbourne, Australia
[d] School of Computer Science, University of Petroleum and Energy Studies, Dehradun, India
[e] Department of CSE, ASET, Amity University, Noida, India
[f] Department of Computer Science & Engineering, Sharda University, Greater Noida

ARTICLE INFO

Keywords:
Gossip Protocol
Cloud Computing
Task Scheduling
Virtual Machine
Inter/Intra-cluster Gossip

ABSTRACT

Cloud computing offers computational, storage, and applications capabilities as subscription oriented services. It offers a platform explore computational assets and use them based on "pay per use arrangement". Thus it opens ways to access boundless assets with negligible equipment and programming at the customers' end. This paper focuses on the advancement of a cloud administration's provisioning structure by building up a dynamic load-balancer for the cloud. It proposes a framework and protocol for the resource environment in the cloud. Distributed Hash Table (DHT) protocol has been utilized for a service query to perform a job agreed by the user. For load balancing, gossip protocol has been used for inter/intra-cluster gossip. For inter-cluster gossip, the load is balanced among the leaders of every cluster. The proposed protocol uses the inter-cloud resource management, where a leader is selected from the cloud that interacts to other cloud and decides on virtual machine (VM) migration. The decision about job allocation is not acknowledged by a single machine, which generates the scalable architecture of the proposed protocol. The protocol considers the current load situation and decides at the time of request submission. This protocol is adaptable, reliable and scalable and supports green computing by utilizing server solidification.

1. Introduction

Cloud computing paradigm involves the use of a massive variety of networked computers, which are dynamically provisioned [30], for executing programs or applications on multiple computers simultaneously. Such service appears to be provided by necessary hardware; however, virtual hardware is used to provide the service. There are principally three cloud classes: public cloud, private cloud and hybrid cloud. A public cloud is easily accessible and open. Any user can use it over the network by paying. However, the private cloud is managed only by a corporation which is operated by third parties and hosted internally or externally.

Although, the hybrid cloud comprises the public cloud and private cloud. If a task is deployed on the private cloud and requires a lot

* Corresponding author.
E-mail addresses: rbuyya@unimelb.edu.au (R. Buyya), ashankar2711@gmail.com (A. Shankar).

https://doi.org/10.1016/j.simpat.2021.102275
Received 31 May 2020; Received in revised form 12 January 2021; Accepted 16 January 2021
Available online 18 January 2021

of machine power, some public service provider will be contacted for the job. Cloud computing tenders four service models namely: Software as a Service (SaaS), Network as a Service (NaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Software as a Service (SaaS) provides the software to users. Platform as a Service abbreviated as PaaS offers its developers a platform for the package, compiler, server, etc. While in the case of IaaS, users are provided with virtual hardware, and it can be utilized as per the demand. Fig. 1 depicts the basic cloud environment architecture.

XaaS is a general, collective term that refers to the delivery of anything as a service. It recognizes the vast number of products, tools and technologies that vendors now deliver to users as a service over a network – typically the internet – rather than provide locally or on-site within an enterprise. Azure Functions are the *function as a service (FaaS)* launched by Microsoft. Microsoft Azure Functions is designed by keeping the developers in mind. This service helps the developers to accelerate the application designing and development process. Microsoft's idea behind this FaaS is to eliminate the time consumed by the application infrastructure development process. Now when this is eliminated, the user can easily create the software application and upload its code. Like all other FaaS, the code runs when it gets triggered. For this, the developers need to set the trigger functions as well. Triggers can come from anywhere, like the application or from other cloud services hosted. This cloud service runs the code whenever it is triggered. The users now have to pay only for the time which is consumed by the function to run. Also, this service is hosted on Microsoft's Public Cloud. For the cloud to provide a virtual layer to the higher layer user within the type of IaaS, PaaS and SaaS, that too consisting of a very reliable, value-effective and challenging way, the issues related to task scheduling and resource allocation must be addressed in a very organized manner. One of the most critical aspects of providing these services is the correct management of the request from the client, that is rapidly growing not just in terms of quantity, but also in terms of the required machine energy. Poor management of the resource in the case of cloud model could hamper it, to a forceful increase in the value of service delivery. For example, in the above case, a non-required resource is allotted to a user. Thus, the user inclines to acquire resources. These resources are usually idle, rather than being utilized by some other deprived jobs that want them.

To utilize the non-required resource and drop wastage, job affinity is addressed in this work by creating a call throughout the allocation of resources. Let us take into account the following case. In this paper, there are two resources A, B and two two jobs X, Y.. Currently suppose job X needs hundred sec on resource A, a hundred and fifty sec on B and Y needs a hundred sec on B and hundred and fifty seconds on A. Then throughout resource allocation, we need to contemplate this affinity of job and assign X on A and B on Y to make sure economical utilization of resources. Otherwise, the job can take more time than allocated on both the machines, thereby tending to delay and resulting in mismanagement of resources. As results of non-utilized resources in a proper manner, there is a decrease in efficiency, performance and an increase in the cost for users and cloud service suppliers, thereby inflicting loss within the business. With the beginning of technology alike cloud computing, power is essential for the working of a server. The Data-centres alone take up 0.5% of the total power consumed by the entire world. A study reveals that in the year 2005, the server and cooling unit took 1.2% of the energy consumed by the U.S., and it is rapidly increasing [18]. There is an estimated increase to four times by 2020 [17]. Carbon dioxide is an emitted gas in all the produce energy that causes significant concern for the environment. To reduce the consumption of power, the proposed protocol practices server consolidation. If the service is dead, then instead of substituting it through a physical machine, the virtual machine is allocated to some underloaded machine. If the task is completed and it results in the physical machine to under-load, henceforth the virtual machine is transferred to another underloaded machine. The protocol proposed takes into account various factors, such as performance, scalability and capabilities, etc. Scalability from the machine point of view means that if we tend to increase the number of machines, productivity should not decrease compared to existing performance. If, in case the number of jobs submitted to the cloud upsurges, then there should be no resultant degradation in the performance.

In proposed work, a dynamic resource management protocol is implemented that keeps on observing resources allocated and creates selections that are supported by the present state of the machine. It considers amendment in demand and capability of the cloud, etc., for deciding allocating resources. As soon as users finish the job that requires enormous processing power in a short time, at

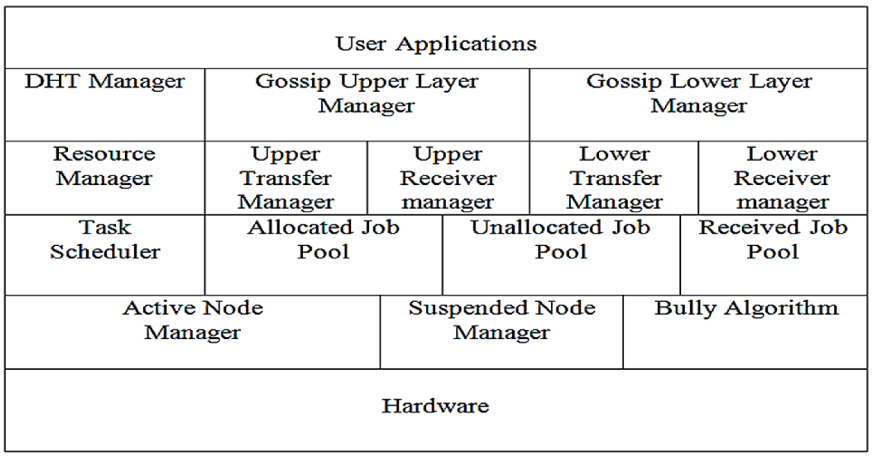

Fig. 1. Leader Node Architecture

that moment, it is challenging for the user to manage all the resources, taking into account all the factors, such as heterogeneity, an affinity for the job, costs, etc. Hence, an automatic resource management system is required to control the system that takes into account all the necessary factors. The protocol makes cloud adaptable by making endless selections for resource management. This selection makes the cloud configured to make changes in conditions such as many submitted jobs, mostly when many users have stopped acquiescing jobs, making changes to the infrastructure etc. The discussed protocol offers flexibility; it provides management for users on a larger scale and is inexpensive for resources. Our protocol makes the system fault-tolerant without assigning responsibility to a single machine and keeping the creation of choice dispersed. The notion of cloud virtualization is used by the proposed protocol to consolidate servers and give an attempt towards reducing the number of different machines in the cloud. For example, if four users want one virtual machine each, then in place of allocating four other physical machines, we tend to allocate four virtual machines that can share physical resources.

The proposed methodology will reduce energy consumption. Virtualization uses hardware resources efficiently and uses system resource downtime. Virtualization results in isolation between applications, because they can run on entirely different virtual machines and other operating systems. Therefore, it becomes complicated for them to attach. Thus, the two applications run in isolation as independent applications, and they cannot attack each other's resources being on different machines. Virtualization makes the cloud quickly accessible. It's a lot of time-consuming tasks to introduce a replacement node or remove a node from the cloud and set it up. However, virtualization allows, to create new virtual machines with totally different functionality and a simple API. It solves several issues that are generally faced by cloud providers. Virtualization helps maintain compatibility through recent and present applications, thus saving the critical investment that service providers could make to update hardware and software to keep them in the latest model.

The rest rest of rest of the paper is organized as follows. Section II deliberates several approaches in the field of resource management. In Section III, it defines the workings of the architecture for the leader node and cluster node. It also presents our proposed protocol and scheduling algorithm for the management of the resource. Section IV to VI provides result analysis and VII emphases on conclusion and future direction.

2. Related work

Resource management has always remained a tricky and challenging issue in computing. In this paper, we consider resource management approaches closely related to our work. This section offers us a quick summary of various challenges and problems that have occurred throughout resource management in the cloud.

There are preemption capabilities present in the mechanism for resource allocation and task scheduling which is proposed by Li et al. [8]. Authors have proposed two algorithms, namely the list scheduling better termed as adaptive way scheduling (ALS) and min-min scheduling (MMS) which is also an adaptive algorithm. These algorithms take advantage of tasks which can perform in parallel. It is the responsibility of the schedular to complete all the resource allocation and scheduling tasks. This allocation being a centralized task of the scheduler, is at risk of causing a single point of failure. To overcome the above problem, Yazir et al. [10] have proposed a mechanism for allocation of resource, an agent ANA (Autonomous Node Agent) performs the allocation. The agent node preserves the physical machine anomaly if the machine surpasses or become deprived of the number of resources below the threshold value for each dimension or does not conform to SLAs. Their approach has certain restrictions. Besides, the migration time of the virtual machines was not taken into account. Several migrations should be restricted to avoid these issues and latency has increased because of the unlimited range of migrations. The heterogeneity of the resources is taken into consideration in this approach.

Warneke et al. [4] assumed that there might be a change in the heterogeneous resources during execution. The defined architecture in this work has a job manager and have multiple task managers. Whenever any job is submitted, it is transmitted to the job manager that manages allocation and deallocation of VM. The proposed model in work involves the planning of job scheduling, with heterogeneous resources in the system. Since there is some parallelizing of the task, it increases the execution speed. It uses a job manager that accomplishes the entire cloud, thus eliminating all the problems associated with a centralized system. In their paper, the authors have not mentioned anything concerning load. If we tend to persevere making VM for each task on the node, this may produce plenty of loads. Therefore, there should be some limit on the VM to be created. Chang et al. [7] in this paper developed a mechanism that describes the independence of task for concurrent executions. This resulted in speed up for the performance of resources and heterogeneity. The issue of dependencies among tasks is considered; if two jobs are interdependent, then they cannot execute in parallel to one another. Therefore, they need to be implemented in consecutive order. Resources used in a prior task will be exploited by later study too. Therefore, a set of resources to satisfy the demands needs to seek out. Equal value is allotted to every resource and therefore, the resource that satisfies the condition of minimum value, and most demand is chosen. Their approach is not contemplated towards job attraction and not dynamic. Lee et al. [6] proposed a protocol taking into account the heterogeneity of resources along with the affinity of the jobs. It is primarily designed for information-analytical applications wherever a great deal of information is concerned. It tries to stay the dimensions of minimum cluster and equality between the jobs. It considers the case of heterogeneousness, wherever a cloud consists of entirely different machines taking different competences.

By keeping the size of the cloud to a minimum, it is seeking to reduce operational value. The planned planning takes into account the heterogeneity of the resources, the equity between the jobs, the similarity of the jobs. It is designed with information about analytical applications in mind. It does not balance the load during the execution of the activities of the tasks. Xiao et al. [9] proposed an algorithm to prevent congestion and green computing. Three thresholds have been identified by hot, warm, and cold. The servers are combined to reduce energy consumption. If the temperature of the machine drops to the green computing threshold, consolidation occurs, to transfer the VM, the machine with the lowest temperature is selected, and again the asymmetry of the resources is taken into

account at the destination. In their work, the authors ignored the importance of virtual machine migration, and user control is centralized wherever the scheduler works, which causes a bottleneck and a single point of failure. Peer-to-peer cloud architecture is proposed by Ranjan et al. [3] it takes into account cloud service discovery and load balancing. The primary bottleneck and single point of failure are eliminated through distributed design via DHT. Their work provides load balancing and discovery of required services for the provision of the services. However, what to do if there is substantial demand for a specific physical machine and alternative physical machines are not loaded enough. In this case, the resources are not used. We can transfer jobs to an underloaded physical machine from a heavily loaded machine. And if some machine is idle, the node can be suspended, resulting in reduced energy consumption. Ergo et al. [5] a method is discussed to distribute resources among jobs based on a rank calculated using a matrix to compare associated classes. In the event of discrepancies in the activity ranking, these discrepancies are detected and removed using the evoked bias matrix. The service standard is measured by the information measure of the network, the execution of the activity, the costs of executing the task and the irresponsibility of the task. There is a pool of jobs where all tasks are collected, and calculations are made for the allocation of resources. All measures and responsibilities for the distribution of resources are allocated on a machine, which causes an overload at peak times or a node failure, which can lead to a failure of the entire system. Wuhib et al. [1] proposed the P* protocol and developed middleware for dynamic resource allocation. The proposed protocol runs on this middleware. This protocol distributes resources between sites. It additionally considers overhead enclosed within the modification in configuration of cloud further, because of the memory constraints. The proposed P* protocol assigns resources on the knowledge of the need for a module. If a module requires a larger number of resources, more resources will be allocated to it. Therefore, this helps to achieve maximum-minimum fairness between sites. Thus, the modular utility is that the minimum between all its instances and the utility of the cases is determined by the relationship between the allocated resources and the demand for a resource. The proposed protocol does not take into account heterogeneity and applies to only one data centre in the cluster.

Yanggratoke et al. [2] presented a generic resource management protocol (GRMP)protocol for resources allocation with a change in input dynamically. It tries to cut power consumption by consolidating underutilized servers and suspending servers that don't seem to be needed or don't have any job, i.e., are idle. It contains a machine pool service that manages the pool of active and passive machines. A lively machine is the one that runs all the elements defined in the previous section. However, a passive machine is the one that doesn't run any part. Since the active machine runs all features, it consumes power. On the contrary, since the passive machine is in standby mode and does not run any element and does not consume energy. Once a machine is underutilized, it transfers the whole load to the other machine and suspends it. Once a machine is overloaded, then the resource manager sends a message within the machine pool service that sends a wake-up LAN packet and returns machine-id by simply activating to the requesting machine. This protocol tries to reduce power consumption; however, it does not contemplate heterogeneous surroundings, and it does not apply to the datacenter having one cluster. Femminella et al. [19] have discussed decoupling the functional services from the physical equipment's when they are executed. They proposed an architecture to track the current status of distributed and virtual service functions. The agents monitoring the architecture makes use of gossip protocol to exchange the information in a virtualized environment. The work aims to increase network scalability. Mohiuddin et al. [20] insight about how to make use of data centres effectively and the ways to manage power consumption. Subsequently, with growing client size and increasing the number of applications utilizing from them, it is really a challenging task to make the working of a data centre in power saving mode. The identification of idle server is concluded, and hence the server is made to hibernate. They have implemented a unique classification technique that is helping them in balancing the load for the machines. Sri et al. [21], have argued about the fact too dynamic allocation of resources and auto-scaling should be supported in the cloud environment. This mechanism should be done along with load balancing. They have utilized theoretical analysis, provided adaptive provisioning of the resources. Kommeri et al. [22] tried to devise a mechanism to load balance the virtual machines in different computing clusters. Cloud technologies were applied to optimize physical usage hardware. Aslam et al. [23] have argued that vcloud computing is an integral part of both industry and an academic point of view. The research till now has just focused on load balancing and sharing of resources. They presented issues and recommendations for further research work. Theja et al. [24] have discussed the optimization techniques for effective management and utilization of resources. They have devised different VM selection policies by using a genetic-based algorithm termed as adaptive genetic algorithm (A-GA). It takes into account additional migration time, correlations among the VMs and random selections along with the utilization of CPU time for each machine. Gupta et al. [25] discussed the heuristic techniques for allocation and management of the resources in the cloud environment. The paper discussed various techniques used for management of techniques related to the cloud for developing and deploying the heuristics. Hamzaoui et al. [26] survey the resource utilization for the cloud environment along with energy consumption. The paper deals with

Table 1
Various well-known techniques and their performance parameters

Technique Name	Algorithm Mechanism	Performance Metrics
Yazir et al. [10]	abstract state machine, 'na' is modeled as a distributed ASM	$T_r = 60\%$
Wuhib et al. [1]	gossip protocol	95% confidence interval for the value was found to be less than 10%
Femminella et al. [19]	Gossip-based monitoring function using DN-based transport network connection	ND bandwidth save by 50%
Mohiuddin et al. [20]	workload aware virtual machine consolidation method	active server power consumption reduced 15%
Proposed Method	Cluster-based Gossip Protocol and Distributed Hash Table (DHT) protocol	90-95% jobs execution time save

the approaches for resource utilization and the different scheduling approaches. Sohrabi et al. [27] discussed various game theory and optimization methods to minimize the cost and maximize the profit. Aggarwal [28] has talked about the cloud being an emerging technology and its impact on the IT industry. The users have a dynamic nature in this. The parallel processing techniques with the help of various heuristics have been developed, and the SLA agreement is also taken into consideration. Li et al. [29] depicted the consolidation or VMs for the utilization of the data centers. Reduction of the violations for the SLA along with ways to reduce power consumption has been actively discussed in the literature. The prediction model issued to predict the host using a classifier scheme known as naïve Bayes. A comparison with more relevant techniques is shown in Table 1.

3. Proposed resource management protocol

Physical machines in the cloud differ in their ability to perform various things. Some of them are cheap for procedural activities, while others are cheap for storage activities. We used this nature of heterogeneity in the model proposed. The jobs are allocated to physical machines based on their requirements. Some of the jobs will run faster on a physical machine, while others will slow down on an equivalent physical machine. Likewise, a job that requires storage space will run faster on the second machine rather than the first. In this article, we tend to consider affinity for work instead of assigning work to a physical machine.

Currently, energy consumption is a problem that causes emissions of carbon. We can shift a machine to energy-saving mode if the machine is idle or we don't need a physical machine for any job. Physical machines are distributed according to their dynamic state according to their needs and not at the beginning. This results in lower costs for cloud users because physical machines are not allocated to them in an idle state. Users indicate the need for jobs while sending to the cloud. Jobs are distributed among resources based on user needs. There is a need a mechanism to track all physical machines, so that doesn't have to look for the best physical machine for every request. Therefore, we use DHT with some skill. The service provider cannot control request; it depends on the needs of the users. All users can submit jobs that require virtual machines with the same processing capabilities. Some of the physical machines are heavily loaded and need load balancing so that they can perform all the operations efficiently. The need for an activity determines the affinity for a particular machine but does not imply limits when completing a job on another physical machine. Therefore, an idle machine can be made to have a charge of VMs. If from a cost point of view, it is advantageous to transfer the VMs to another physical machine, otherwise the VMs are not shared. The hash DHT [14–16] requires attributes of physical machines. To search for a physical machine suitable for a particular job, user-defined requirements are accepted as hash keys.

There are various attributes of a machine, such as a processor speed, memory, number of cores and GPUs. These attributes create five research areas. There are entirely different approaches to visualization in multidimensional space, such as the kd tree [11], SFC [11], quadrant tree [13], and the NR tree [12]. They have their own advantages and disadvantages, as the SFC suffers from a space curse. However, it is easy to use and works well in an extremely dynamic environment. Likewise, a Kd tree does well in gigantic dimensions. However, he suffers from a dynamic environment. We choose among them based on specific cloud settings.

3.1. Architecture

Two architectures have been developed: the first for the cluster leader and the second for the entire physical machine in the clusters, except for the leader node. The difference in the architectures is that the services they provide and the protocols with which they collaborate and therefore, the services they provide. The infrastructure provided by cloud providers for computing is the hardware. Fig. 1 demonstrates the leader node architecture consisting of various elements. The DHT manager manages all DHT related activities, such as adding the last cluster, deleting the cluster after it exits, etc.

The DHT manager works only with cluster leaders; consequently, leaders represent the whole cluster in different clusters. Clusters interact with various clusters through their leader. The Gossip Manager manages gossip protocol. Two gossip managers are used, one of which manages resources between clusters and the other within the cluster. The top-level gossip protocol manages resources between the leaders of all clusters, and the lower-level resource manager manages resources between the physical machines in the cluster. The transfer manager at the upper and lower levels takes into account completely different constraints and therefore decides whether to transfer the virtual machine to another node. The receiver manager on both the levels, i.e. lower and upper decides whether the virtual

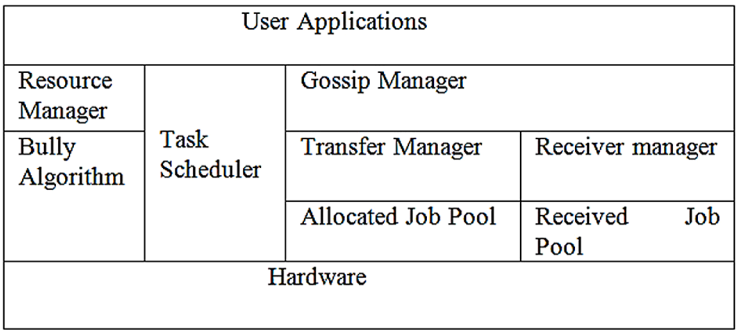

Fig. 2. Cluster Node Architecture

machine will be hosted, taking into account all aspects. The scheduler schedules a task on the physical computer on which it runs, based on the Bully algorithmic scheduling rule used. A dedicated activity pool contains all the activities presented to this physical machine that runs on a VM. The bully algorithm dynamically elects a leader from a bunch of distributed computer processes. The process with the highest process ID number from amongst the non-failed processes is selected as the coordinator.

The job pool receives jobs that contain all the collection of jobs that are (VMs) received from the opposite physical machine. It is used in making decisions based on priority through task scheduling. The active machine manager manages all physical machines that currently have virtual machines if the physical machine is idle or meets a threshold condition, this machine is stopped, and control of this machine is transferred to the administrator of the suspended nodes, which controls the suspended physical machines also known as the manager. In the case, a physical machine is idle and transferred to the acting manager. A node invitation arrives, then assign the physical machine to be used or used under some criteria. Fig. 2 shows the primary nodes, and cluster nodes being a part of it. They all have similar properties, as shown in the previous paragraph.

3.1.1. Proposed protocol

On the physical computer, we are having three pools: an unallocated activity pool, a distributed activity job pool and a received job activity pool. The distributed job pool contains tasks assigned to a physical computer and running on a virtual machine. The unallocated activity pool has activities or jobs allocated to the physical computer, but not performed any task (waiting to receive the CPU). The activity pool received contains activities (VMs) received from another machine to this physical machine.

Two protocols are used in our proposed work: Gossip Protocol and DHT. DHT is used to search for resources in the cloud. Fig. 3 shows completely different leaders collaborating on DHT, and each leader runs DHT in the cluster. The leader represents the group of the cluster from which contain it. Knots other than the leader do not start running the DHT. The gossip protocol is used to balance the load in the cluster. The Gossip protocol is run within a cluster, in which all the physical machines in the cluster participate. Besides, it works among leaders to support the workload between clusters. Therefore, although one cluster receives tons of requests, while others don't receive any, the cluster is not overloaded. However, the load between clusters is balanced. The term node and the physical machine are interchangeable. Fig. 4 shows that the leader node of each cluster participates in the Gossip protocol, and Fig. 5 shows all the DHT and Gossip protocols among the leaders.

Each physical machine supports two lists of nodes: a list of overloaded nodes and a list of underload nodes. These two lists are managed using priority queuing. The timestamp and processor load are used as a priority: the longer the time or less load, the higher the priority. The leader has two overloads and two in the list of underloaded nodes. Therefore, other nodes have a list of underloaded and overloaded nodes. The list of underloaded nodes contains a list of physical machines where the load is less than the specified value, and the list of congested nodes includes a list of nodes where the load exceeds the specified value. Whenever the allocation or deallocation of jobs takes place, they are accepted into the pool, as the load growth necessarily increases by a certain amount. In this case, we incline to implement the gossip protocol for load balancing. The gossip protocol tries to resist the load of several segments. If it is an excessive load, we tend to check the list of nodes with the insufficient load. If there is no node in this list, it sends a message to the primary node, i.e. the leader informing it. If the list of underloaded nodes contains some information, send a message asking if this node continues to under-load. If the recipient is not sufficiently loaded, i.e. underload, send a confirmation and add the sending node to the list of loaded nodes or update the timestamp in the list of congested nodes. If receiving node being overload, it transmits a negative acknowledgement and adds the sender to its list of overload nodes.

The sender of the request adds the recipient node to its list of overloaded nodes and removes it from the list of underloaded nodes if it exists there.

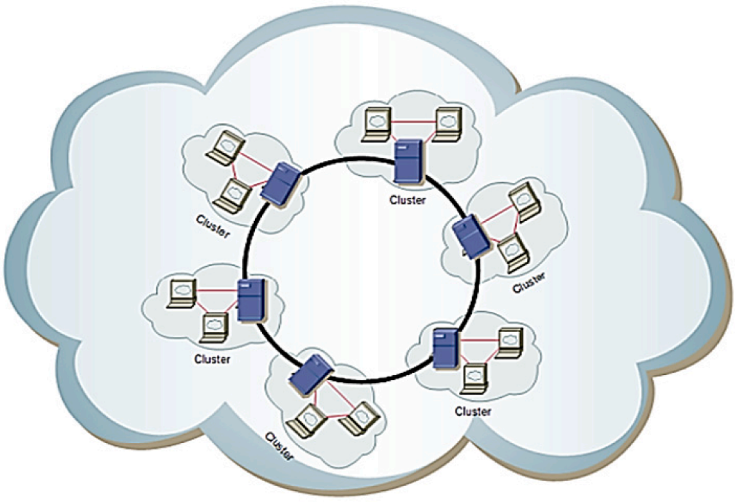

Fig. 3. Nodes arranged using DHT

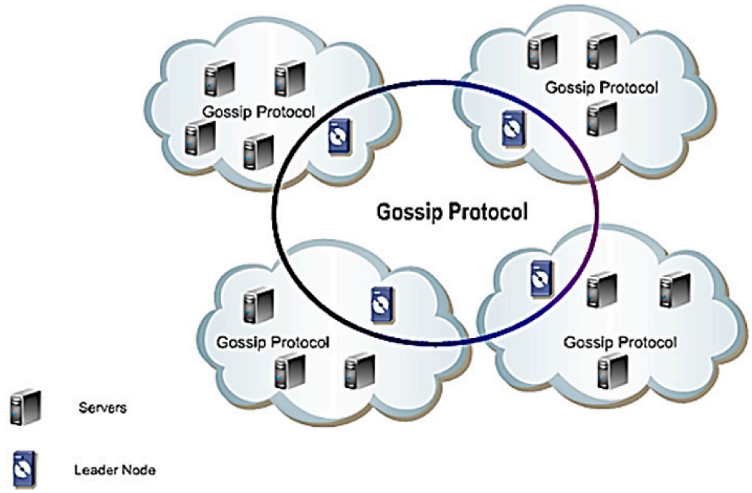

Fig. 4. Leaders in Gossip Protocol

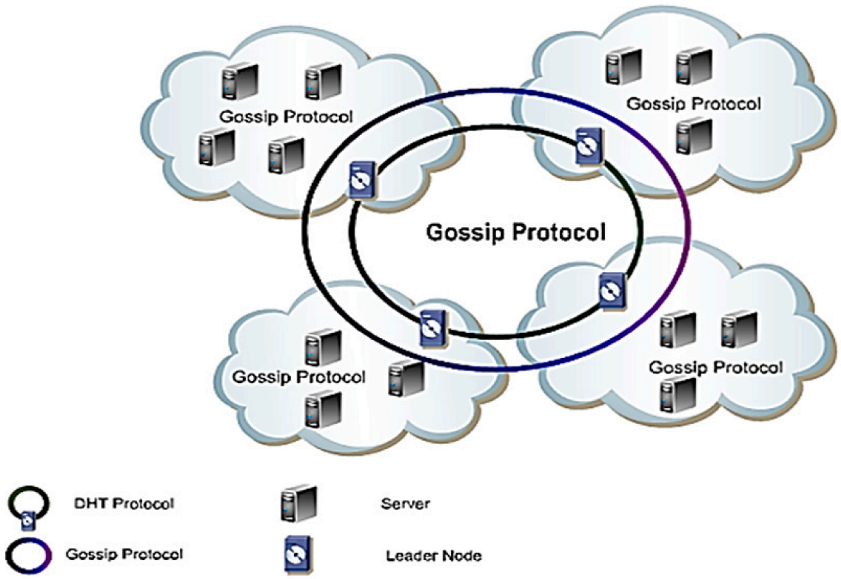

Fig. 5. Gossip and DHT Protocol among Leaders

$$priority\ queue = k \times time_{stamp} + (1-k) \times (maximumload - load \tag{1}$$

We use the priority queue for an insufficiently loaded list of nodes. Then we add the nodes that last requested several jobs or the nodes towards which the virtual machines are drifted. Likewise, in the case of an overloaded node, we have an updated list of congested nodes. If a node receives a request to power the machine, it adds the requesting node to the list of overloaded nodes of the receiving node. If the requesting node receives a negative acknowledgement from the receiving node, then we add the receiving node to the list of congested nodes of the requesting node.

As soon as the node is overloaded, it checks the list of the under-loaded nodes and sends a message to the under-loaded. If a positive confirmation is received, the change occurs in the priority queue timestamp of the underloaded node. At the identical time, a node that is receiving changes its list of overloaded nodes. If the sending node is already in the full list of nodes at the receiving node, the timestamp changes, otherwise it is removed from the underloaded list and added to the full list of nodes. With a negative acknowledgement, an underloaded node identifier is transmitted by the receiving node k. The requesting site confirms the node timestamp received in negative form and compares the timestamp of the most recently updated nodes in the priority list. The request is sent to the node, which takes longer than the other nodes in the priority list. This continues until the underloaded list is vacant. Once it is empty, an overload signal is generated and sent to the supervisor or the leader. If the node is not loaded sufficiently, the call for participation is sent to the node from the complete list of nodes. To request a job, if the recipient is full, it replies with a migrating virtual machine. The

sender of the virtual machine updates the timestamp in the list of overloaded nodes. If the requesting node is already in the underloaded list, the timestamp changes, otherwise the underloaded node is detached from the full list of nodes and added to the underloaded list of nodes. The requestor node modifies the timestamp of the entire node. If the recipient of the request is by now underloaded, it changes the list of overloaded and overloaded nodes, which it possesses in sequence. It sends a negative confirmation with the address of the full node. The requester compares the timestamp of the node received in the negative confirmation with the timestamp of the first node in the full list of nodes. The next request depends on the node with a higher timestamp, and the request is sent to this node. Also, the requesting node changes its full list of nodes accordingly.

When a leader receives an overload message from a set of already defined nodes, it considers the cluster overloaded and tries to contact other clusters to migrate the virtual machine. The process is similar to the Gossip protocol in the upper level, except that it distributes the load between clusters.

3.1.2. Transferring policies

Few parameters are considered, before migrating virtual machines to other nodes. Migration for the virtual machines to other nodes is only possible if the nodes are overloaded/underloaded. We begin to monitor the loading of our system in certain events, such as sending jobs, migrating virtual machines or completing the job. After a while, we find the weighted average load on the system. If it is greater than the default value, the virtual machine must be transferred to other nodes with the low load.to save energy, we are trying to share the load on a node from an insufficiently loaded node. A node is called super underutilized if it has very few virtual machines that can be moved to other nodes and paused. The above technique can save energy. Migration decision is taken based on the following factors: communication delay (Cod), expected delay (Exd) for getting CPU to this job, execution time (Ext) on the current node, reconfiguration delay (Red), time to execute (Tio) on another node, expected delay in getting CPU to this VM on the transferred node (Edo). For transfers, the following condition must be true:

$$Exd + Ext > Cod + Red + Tio + Edo \tag{2}$$

4. Proposed algorithms

In this section, the proposed algorithms are explained. Algorithm 1 provides the steps for joining new node in the cluster and Table 2 describe the symbols used in the algorithms. Consider k be the new node, u_l be the underloaded list and o_l be the overloaded list.

If a node wants to enter the cloud then algorithm 1 is used. With the help of Line 1 a desired node can be found in any cluster. Let's go to the leader (Line 2) of the cluster. Using the attributes, the hash value is computed (Line 3). The node finds its position (4) using DHT and moves to that position. The nearby node value is found using find_neighbor (line 5). By means of the neighbor_node computed in the previous step, an overloaded list of neighbors is added to the list of overloaded nodes of the new node in line 6, 7. Likewise, the list of underloaded nodes is also copied (Line 8, 9) from an adjacent (neighbor) node. In the algorithm, new_node sends a broadcast message (line10), new_node is added by all neighbors to the list of under-loaded nodes.

Since, in DHT the leader is also participating, it is also necessary to add nearby DHT nodes (line 11). Each node, when it wants to be added goes to the function latest_node_join () algorithm. A node, computed in the higher level gossip protocol and the two list one an overloaded list of nodes (line 13) and second an underload (Line 14) is copied from this node. The leader node is transmitted (broadcasted) in such a way that everyone can have this node in their list of neighbors (line 15).

K1 - over utilization, K2 – underutilized, K3 – super underutilized (Algorithms 2–3)

The virtual machine migrates (line 1) while waiting for the node to become underloaded or overloaded. If the virtual machine is accepted (line 2), the timestamp in list_underloaded_node is updated in line 3 and the number of jobs are reduced (line 4) in the pool of jobs that are unallocated. In case of non-acceptance of virtual machine (line 5), we remove (line 6) the node from the list of insufficiently loaded nodes and add it (line 7) to the list of overloaded nodes. If underloaded_node_list (line 8) is empty, a message (line 9) is sent to the leader that the cluster is overloaded. Virtual machines are enquired (line 10) from other nodes if the load is between k2 and k3. If the virtual machines (line 11) are transferred from an overloaded node, we change the timestamp to a modified value (line 12) of the list of overloaded nodes, accept (line 13) the virtual machine and increase the pool of activities received (line 14). And if it receives a negative confirmation (line 15) from the nodes, then it removes (line 16) this node from the list of overloaded nodes and adds it (17) to the list of overloaded nodes.

Table 2
Symbols used

Symbol	Description
u_l	underloaded list
o_l	overloaded list.
$atr(k)$	Attributes of node
C	cluster list
k.leader	leader of node k
Exd	expected delay
Tio	time to execute
Ext	execution time

Algorithm 1
Joining a node into the cluster using DHT

Input: Attributes of node *atr*(*k*), cluster list *C*, underloaded list *u_l* and overloaded list *o_l*
Output: updated underloaded list *u_l* and overloaded list *o_l*
1. Locate *k* and leader of *k*
2. Goto(k.leader)
3. Hash_fun=function_hashing(atr(k)) // Using DHT
4. *k*→find_neighbor(C_k) // Allocate node k to cluster c_i with maximum matching attributes
5. for (i=0, *i*<=neighbor(*k*),i++):
6. if (*i*=overloaded):
7. *o_l* = *o_l*+*i*
8. elseif(i=underloaded)
9. *u_l* = *u_l*+*i*
10. Broadcast(*k*) // local gossip
11. k.leader=latest_join_node(k)
12. for (i=0, *i*<=|C|,i++):
13. *o_l*(*Ci*) = *o_l*(*Ci*) + *o_l*(C_k)
14. *u_l*(*Ci*) = *u_l*(*Ci*) + *u_l*(C_k)
15. Broadcast(*k*.leader) // global gossip

Algorithm 2
Gossip protocol

Input: **C**luster list *C*, underloaded list *u_l* and overloaded list *o_l*, Job_poolList JobList
Output: updated underloaded list *u_l* and overloaded list *o_l*
1. for (i=1, i<=|*u_l*|,i++): // Lower_Layer_Gossip_Protocol
2. if node[*i*]←ACK:
3. Update_TimeStamp(*u_l*[*i*])
4. Assign node[*i*] a job from Job_List
5. Job_List=Job_List-1
6. Elseif node[*i*]←NACK
7. *o_l*= *o_l*+ *u_l*[*i*]
8. *u_l*= *u_l*- *u_l*[*i*]
9. If |*u_l*|==0:
10. Send Overloaded Msg to leader
11. for (i=1, i<=|*o_l*|,i++):
12. if node[*i*]←ACK:
13. Update_TimeStamp(*o_l*[*i*])
14. node[*i*] releases a job
15. Job_List=Job_List+1
16. Elseif node[*i*]←NACK
17. *o_l*= *o_l*- *o_l*[*i*]
18. If queue_size() > s5 // Upper_Layer_Gossip_Protocol
19. If (queue[rear].time_stamp_current] – queue[front]. time_stamp()) > s4)
20. ++front
21. Upper_layer_utilization()
22. Else
23. Return false;
24. Else
25. Return true;

Algorithm 3
Node Transfer Policy

Input: Attributes of node *atr*(*k*), cluster list *C*, underloaded list *u_l* and overloaded list *o_l*
Output: updated underloaded list *u_l* and overloaded list *o_l*
1. If Ask_status(node[m]) is overloaded
2. Return NACK
3. For m = 0 to nofjobs
4. If Exd + Ext > Cod + Red + Tio + Edo
5. Transfer_job[m]
6. Return ACK
7. Return NACK;

If the queue size exceeds the specified value (line 18) and the timestamp is less than the value specified, of the first message (line 19), then we reject (line 20) the first message received from the queue and call (line 21) the procedure. If the size of the queue is greater (line 22) than the value, it returns false (line 23), otherwise (line 24) in all other cases we return true (25).

Using ask_status (line 1), we check if we can transfer the virtual machine to node [i]. If migration cannot be completed due to an

overloaded machine, NACK is returned (line 2). For all virtual machines (line 3), it is checked whether the migration of the virtual machine is advantageous (line 4). If the migration of the virtual machines is allowed, we transfer (line 5) the VM and return (line 6) the ACK, otherwise NACK is returned (line 7).

Receive_broadcast_message(node) /*new node join broadcast message */
1. Add node to Underloaded_node_list

When a new node broadcasts its joining message, then everyone adds (line 1) it to under loaded node list.

5. Perforamcen evaluation

We evaluated the performance of our protocol and compare it with DHT and Gossip-based approach using in-house simulation. For comparison, in this paper execution time of jobs and network overhead is compared with existing methods. The impact of increase in the number of clusters and the number of nodes in the clusters on network overhead and execution time has been shown. We have considered that a job is any process that keeps the processor busy. We cannot evaluate the execution time of instance because it has an interactive session, and it is challenging to calculate the exact execution time. Therefore, we consider that the process is started when an instance is launched, and an instance terminates with the termination of the process. Job refers to the processes associated with the instances in the coming sections.

5.1. Comparison based on job execution time

We compare the proposed approach with a gossip-based approach and a DHT based approach. For comparison of 1000 jobs, we take 30 seconds to execute the job on the best available physical machine. There are 100 machines exist in the cloud, and all the machines differ in their configuration. We are going to consider best, average and worst-case time for all jobs to execute. According to gossip-based approach, all the jobs will be submitted to the cluster. In our work, GPB, DPB and PA represent Gossip protocol-based, DHT Protocol Based and Proposed Approaches respectively. However, BC, AC, GBWC, DBWC describe the best case for all the approaches, worst case for all the approaches, worst case for gossip protocol, worst case for DHT based protocol. Now, time to execute all the jobs assigned to a machine is

$$\Gamma = \sum x_i \tag{3}$$

Where i = 0, 1, 2…m is the time taken by a job to execute on a machine. Since all the jobs have the same execution time, we can write Γ as

$$\Gamma = m \times x \tag{4}$$

Where Γ is the total time required to execute jobs and n is the total number of jobs that need to be executed by a single machine in the described setup in the previous paragraph. All the machines get ten jobs. Therefore:

$$\Gamma = 30 \text{ seconds} \times 10 = 300 \text{ seconds}$$

Now, we are not considering the time taken for the management of the cluster. Therefore, time taken by the cluster in the best case will be 300 seconds, only when all machines have the best configuration to execute the jobs. Similarly, according to the DHT based approach, time taken will be 30 seconds, only when all the machines are best suited for the job and have almost the same configuration but not precisely the same. Time to execute the job will be 300 seconds since in both cases; each machine is best suited for the job and gets the job as soon as it is submitted to the cloud. Therefore, all of them will take 300 seconds. In our approach, all the jobs are allocated to the same cluster, and they will get executed at the same time. Fig. 8 shows the worst-case scenario for a gossip-based approach, where two types of jobs A and B are submitted to the cloud. Also, the cloud has two types of machines P and Q. On one hand, where P requires 30 seconds for executing the job A and 60 seconds to implement job B, machine Q requires 60 seconds for job A and 30 seconds for job B. In the worst case, all the P-type of machines will execute B type of jobs, and Q type machine will execute A typing job. Therefore, the time required by this approach will be 600 seconds, and the time needed by the DHT based approach will be 3000 seconds. However, our approach needs 300 seconds for the execution of the job.

Now let's consider the best case for DHT based approach, we have the same scenario as for the first case except the one difference that the machines have precisely the same configuration. In that case, the time required by the gossip-based approach is 300 seconds and DHT based approach is 30,000 seconds. Similarly, for the average case, the time needed for the gossip-based approach approximately is 400 seconds and by DHT based approach is approximately 600 seconds. Our method takes 300 seconds in our case. In this case, for the gossip-based approach, T is 60. Therefore:

$$\Gamma = 10 \times 60 = 600 \text{ seconds}$$

For DHT based approach, since all the jobs are distributed among only two machines, both the machines will get 500 jobs each.

$$\Gamma = 500 \times 300 = 15,000 \text{ seconds}$$

Now let's consider the worst-case for DHT based approach, we have the same scenario as for the first case except that all the

machines have the same configuration. The time required by the gossip-based approach is the same as described in the best case, 300 seconds. For DHT based approach

$$\Gamma = 1000 \times 30 = 30,000 \text{ seconds}$$

Similarly, for the average case, the time required by the gossip-based approach, we consider that out of 10 jobs five have got the required machine, and five did not get the opportunity. Therefore, the time required to execute jobs is

$$\Gamma = 5 \times 30 \text{ seconds} + 5 \times 60 \text{ seconds} = 450 \text{ seconds}$$

In the average case of DHT based approach, all the jobs get assigned to half of the machines, and half are always waiting for jobs to get allocated to them. Therefore, the time to execute jobs can be represented as follows:

$$\begin{aligned} \Gamma &= \frac{X \times T_i}{T_m} \ seconds \\ \Gamma &= 30 \text{ seconds} \times \left(\frac{1000}{50}\right) \text{seconds} = 600 \text{ seconds} \end{aligned} \tag{5}$$

Where T_i is the total number of jobs, T_m is the total number of machines. The proposed approach takes 300 seconds in average case (Fig. 6). Comparisons have been made based on job execution time.

5.2. Comparison based on network overhead

BC, AC, GPB, DPB, PA all have the same meaning as described earlier, and WC represents the worst case for all the approaches. Let's consider the network overhead caused by all of them. Then, for load balancing purpose only, we are not considering load caused by graph construction in case of DHT based approach and heartbeat lists updating in case of our method. If we consider 1,000 jobs and 100 machines, then network traffic generated only for load balancing is 4,000 messages best, worst and average case. In the case of gossip-based approach, network overhead is approximately 2,000, 100,000 and 200,000 messages in best, average and worst-case.

$$\Delta = (p + s + i + r) \times m \tag{6}$$

In the above equation (6), Δ is network overhead, p is a message for publishing, s message for subscribing, i is a message for informing and r reply message. Therefore, for a DHT based approach, network overhead is in the best, worst and average case.

$$\Delta = (1 + 1 + 1 + 1) \times 1000 = 4,000$$

In case of gossip-based approach network overhead in best case is

$$\begin{aligned} \Delta &= (J_s + D_s + n_e) \times m \\ &= (1 + 1 + 2) \times 1000 \\ &= 4,000 \ messages \end{aligned} \tag{7}$$

J_s is job sending message, D_s is job distributing message m is total number of machines and n_e is enquiry message. In average case, half of the jobs get assigned correctly and half of them are assigned correctly only after contacting 50machines therefore network overhead is

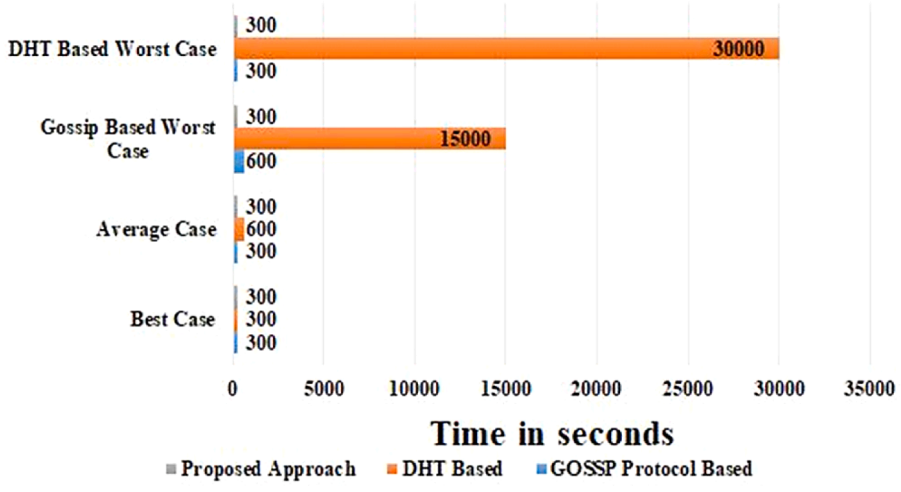

Fig. 6. Comparison based on job execution time

$$\Delta = (1 + 1 + 200) \times 500 + (1 + 1 + 2) \times 500 = 1,01,000 + 2,000 = 103,000 \; messages$$

In the worst-case, all the jobs get assigned after contacting all the machines in the cluster, therefore:

$$\Delta = (1+1+200) \times 1000 = 202,000 \; messages$$

In the proposed approach, network overhead is given by the following formula

$$\Delta = (J_s + D_s + n_e + S_D) \times n \tag{8}$$

Where S_D is the messages used by DHT protocol to send a job to the leader of the cluster, therefore in the best case:

$$\Delta = (1+1+1+2) \times 1000 = 5,000 \; messages$$

In the average case, half of the jobs get assigned correctly, and half of the jobs send enquiry message, and 1/4 jobs get assigned correctly after an enquiry to half of the machines and remaining 1/4 jobs get assigned correctly after an enquiry to all the machines and enquiring half of the cluster; therefore, network overhead is

$$\Delta = (1 + 1 + 1 + 2) \times 500 + (1 + 1 + 1 + 20) \times 250 + (1+1+1\,(20+10)) \times 250 = 2500 + 5750 + 8250 = 16,599 \; messages$$

In worst case all the jobs get assigned after enquiry to all the machines in the cluster and inter cluster. Therefore, network over is

$$\Delta = (1 + 1 + 1 + 40) \times 1000 = 43,000 \; messages$$

From the above discussion, we can see the inference that × out method gains easily in case of load balancing. It performs better than the gossip-based protocol, but it does not perform better than DHT in case of network overhead. Fig. 7 shows the network overhead analysis.

5.3. *Effect of increase in number of servers on job execution time*

Let us observe the effect of an increase in the number of servers in a cluster. For experimentation, consider four clusters, each having five nodes and 1000 jobs. Each job takes 30 seconds for the best-suited virtual machine and 60 seconds for the other virtual machines. In the average case, consider 200, 300, 300 and 200 jobs will be assigned respectively to different clusters. Therefore, time to execute 200 * 4(cluster) is 1200 ((200 /5) * 30). After 1200 seconds, two clusters are completely idle, and two have 100 jobs. During load balancing, 33 jobs will be transferred from both the overloaded nodes to the under-loaded nodes. Therefore, the time to execute these 200 is 396 ((66 /5) × 30) seconds. Total time required to execute the job is 1200 + 396, which is 1,596. Fig. 9 shows the effect of an increase in the number of servers on job execution time.

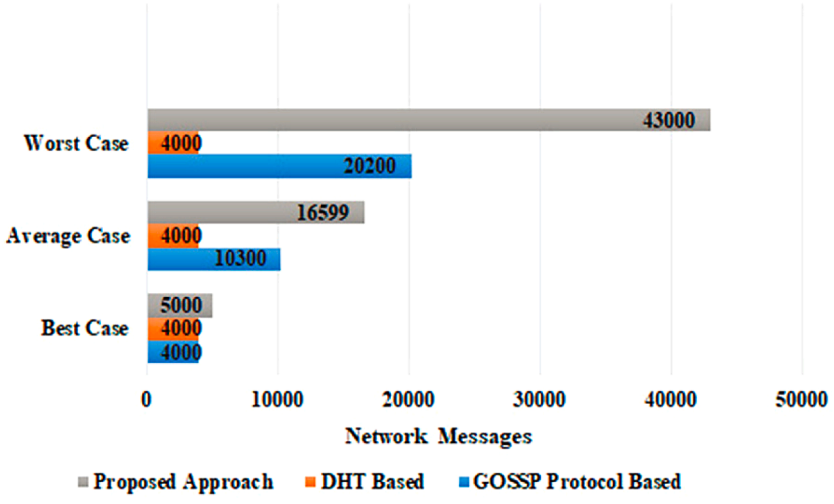

Fig. 7. Comparison based on network overhead

$$\begin{aligned} \Gamma &= \sum X_m/m \\ T &= (200 \times 30)/5 + T_o \\ &= (1200 + T_o))seconds \end{aligned} \tag{9}$$

Here, m is the number of servers in the cluster. After 1200 seconds, two clusters are completely idle, and two have 100 jobs. During load balancing, 33 jobs will be transferred from both the overloaded nodes to the under-loaded node. Therefore, the time to execute is

$$T_o = 66 \times 30 / 5 = 396 \; seconds$$

Total time required to execute the job is 1200 + 396, which is 1,596 seconds. Fig. 8 shows the effect of an increase in the number of servers on job execution time. Similarly, time to execute the same number of jobs in the same environment but an increasing number of nodes in a cluster from 5 to 10 is calculated as follows. For 200 jobs, the time required is

$$\Delta = (200 \times 30) / 10 + (66 \times 30) / 10 = 798 \; seconds$$

Similarly, we can prove that the time required in case of 15servers in a cluster is 532 and for 20 servers in the node is 399 seconds.

$$\Delta = (200 \times 30)/15 + (66 \times 30)/15 = 532 \; seconds$$

For 20 servers:

$$\Delta = \frac{200 \times 30}{20} + \frac{66 \times 30}{20} = 399 \; seconds$$

This proves that with the increase in the number of nodes, the time required to execute the job decreased linearly. For the above calculation, job transferring overhead is not considered. Similarly, time to execute the same number of jobs in the same environment but an increasing number of nodes in a cluster from 5 to 10 is calculated as follows:

For 200 jobs time required for execution is 600 seconds ((200 / 10) × 30) with 15 servers and time to execute 200 jobs is 198 ((66 /10) × 30) with 20 servers. Therefore, the time to execute all the jobs is 798 seconds. Similarly, we can prove that the time required in case of 15 servers in the cluster is 399 seconds and for 20 servers in the node is 200 seconds. This demonstrates that with the increase in several nodes, the time required to execute the job decreases linearly. For the above calculation, instance migration overhead is not considered.

5.4. Effect of increase in number of clusters on job execution time

Let us consider the same set up as defined in the previous subsection except for several clusters.

Concerning Fig. 9, Let us suppose that the number 1, 2, 3 and 4 are clusters. In the case of one cluster, proposed approach behaves mostly like a gossip-based approach where many nodes in the cluster are 20. In the average case, 600 jobs will get the desired machine and get executed in 30 seconds, and 400 jobs will be executed on another machine, and get executed in 120 seconds. Therefore, time to execute the job is (600 / 20) × 30 + (400 / 20) × 60 = 2,100 seconds. Now, if the number of clusters is increased to 2, then the number of servers in one cluster is 10. But the probability of getting jobs to the best available server increases, suppose it increases to 650 time to execute jobs is (650 / 20) × 30 + (350 / 20) × 60 = 32.5 × 30 + 350 × 3 =2,025. Similarly, considering an increase in the number of

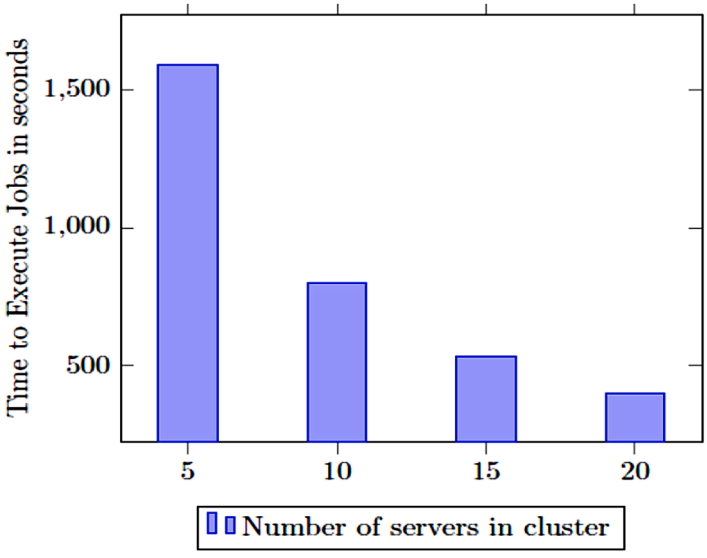

Fig. 8. Effect of Increase in the number of servers on job execution time

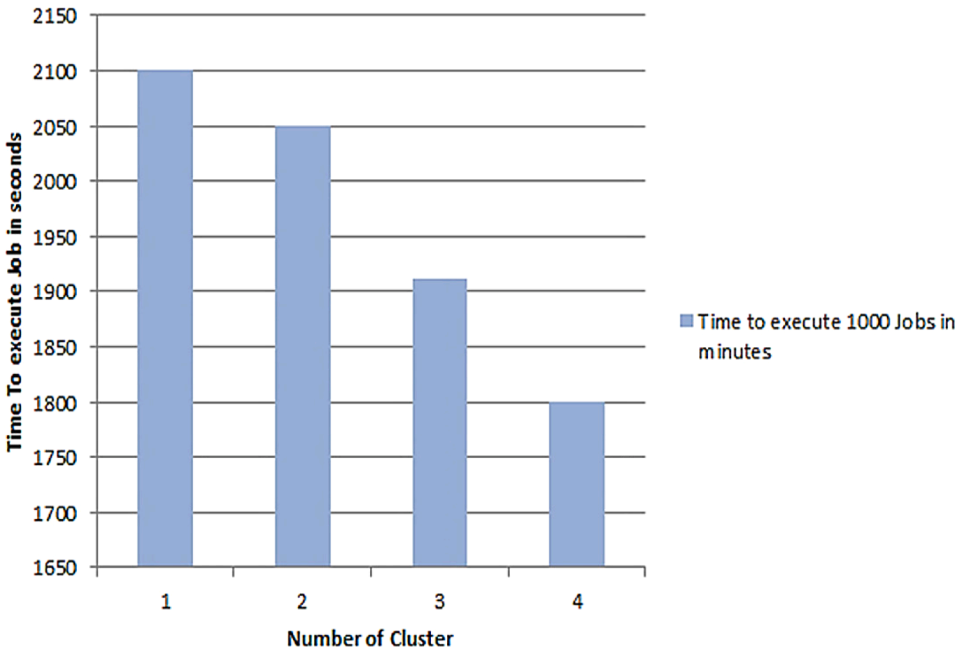

Fig. 9. Effect of increase in several clusters on the job execution time approach.

clusters to 3 and 4 gives 1912 (725 jobs assigned according to job affinity) and 1800 (800 jobs assigned according to job affinity). Fig. 10 shows the effect of an increase in the number of clusters on job execution time.

$$\Delta = (600 \times 30) / 20 + (400 \times 60) / 20 = 2,100 \; seconds$$

$(600 / 20) \times 30 + (400 / 20) \times 60 = 2,100$ *seconds*.

Further, if number of clusters is increased to 2 then number of servers in one cluster is 10. But the probability of getting jobs to the best available server increases, suppose it increases to 650 time to execute jobs is

$$T = (650 \times 30) / 20 + (350 \times 60) / 20 = 2,025 \; seconds$$

Similarly, considering increase in number of clusters to 3,725 jobs will be assigned according to job affinity. Therefore,

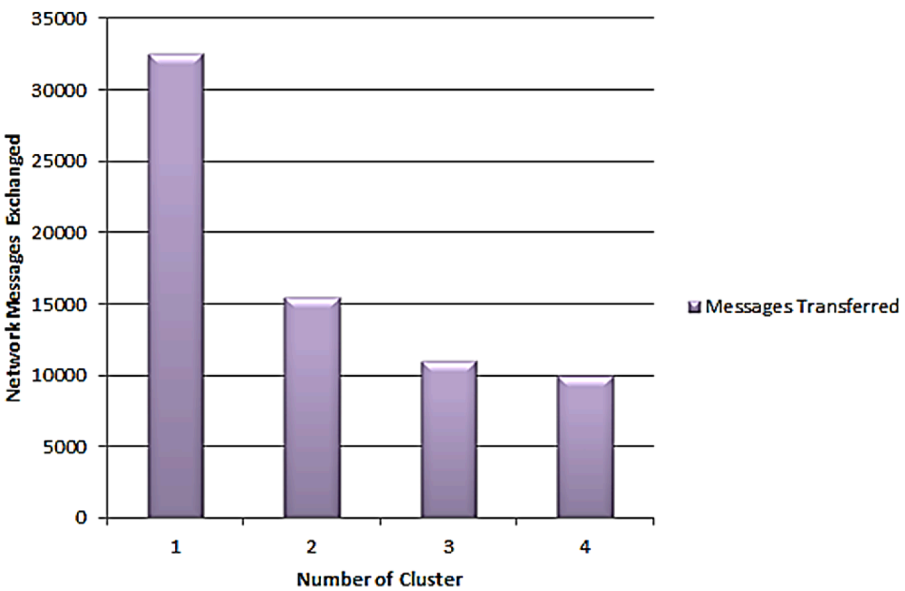

Fig. 10. Effect of Increase in cluster on Network Message Exchange (i.e. Network Overhead)

$$\Delta = (725 \times 30) / 20 + (275 \times 60) / 20 = 1,912 \; seconds$$

For four clusters, 800 jobs will be assigned according to job affinity. Therefore:

$$\Delta = (800 \times 30) / 20 + (200 \times 60) / 20 = 1,800 \; seconds$$

5.5. Effect of increase in clusters on network overhead

For these results, suppose a cluster contains 20 servers and 1000 jobs are being submitted to the cluster, and only one cluster exists in the cloud. After submitting the job, suppose half the servers are overloaded. Then half the jobs will be assigned after two messages between leader and server. But this half is assigned based on the number of servers overloaded; therefore, network overhead is

$$\Delta = (1+1+1+2) \times 500 + (1+1+1+20) \times 500 = 14,000 \; messages$$

If servers are divided into two clusters, then 10 servers will be in one cluster. Therefore, the overhead will then be:

$$\Delta = (1+1+1+2) \times 500 + (1 + 1 + 1 + 10) \times 500 = 9,000 \; messages$$

Similarly, if the number of clusters is increased to 3, then two clusters will have six servers, and one will have eight servers; therefore, overhead is:

$$\Delta = (1+1+1+2) \times 500 + (1+1+1+7) \times 500 = 7,500 \; messages$$

For 4 clusters, the number of servers in one cluster is five therefore overhead is

$$\Delta = (1+1+1+2) \times 500 + (1+1+1+5) \times 500 = 6,500 \; messages$$

If the number of clusters increased too much and many nodes in the cluster are very few, then it behaves like DHT based approach. Fig. 10 shows the effect of an increase in the cluster on network overhead.

5.6. Formal analysis with CCS model

For the purpose of verification of the proposed model, we have used a communication process called the calculus of communicating process (CCS). The main operations for the construction of agents are as follows:

1 Action Prefixing (a.P)
2 Choice operator(A+B)
3 Parallel Composition (A|B)
4 Restriction (P\L)
5 Re-labelling (P [a/b])

Using the prefix Action, we indicate the process that is acting "a"; therefore, it acts similar to "P". The selection operator allows us to define a process that behaves like A or B. A process is determined by the parallel composition whose behaviour is the result of the simultaneous execution of A and B but must be synchronized by the action of the agents of synchronization. When two processes interact at ports A and A, we can replace the label with τ. This determines that the output of one process is the input of another process. The relabeling operator defines processes that behave like "p", but execute "b" when "p" executes "a". The cluster-level abstraction is used. In a cluster and within the clusters for, the upper-level gossip protocol and the lower level gossip protocol the same algorithm is used, but the upper-level protocol works only when the next cluster is overloaded, and the Gossip protocol lower level runs when the physical machine is under load or overloaded. Therefore, in the upper layer, the same modelling can be used, except that only the primary node or the leader will participate in the upper level and the cluster nodes will participate in the lower level.

In the model that is proposed, there is a sender, a recipient and a Leader along with some communication agents. A node can act as a sender or recipient depending upon the load, and the leader is static. A leader receives a job from outside the cluster and allocates the work among all nodes and can act as a sender and recipient based on the state of the load. The sender is an overloaded node, and the recipient is an underloaded node. The sender can be underloaded after the transfer of the instance. The communication among the component is made through an input and output port. These output ports and input ports are denoted as BLOBs in the component circle on the label transition graph. The recipient receives the jobs from the external ports in the receiving port and distributes them between the nodes using the sender. Overline signifies the output port. As soon as the anode is overloaded, it transmits information via info node. The recipient receives this information through the information port, i.e. 's' port and sends the data via info_r, which arrives through the additional port. Then the origin migrates the instance through the sending process received by the recipient through the additional port. The result is passed on to the leader, who passes the result.

5.7. Language specification for a cluster

The cluster accepts the request and passes the result to the users. It can appeal for an instance if formerly launched cases are launched, sent by the same user or by different users.

For example, the current system state is S1, then when it receives a request and enters state S2. Being in state S2, it can either receive a new requestor send back the result and go to state S1. This is explained in Fig. 11. Therefore, the model is presented:

$$S1\_state \;=\; accept.S2\_state$$

$$S2\_state \;=\; accept.S2\_state \;+\; result.S1\_state$$

5.8. Language implementation for a cluster

There is an overloaded node, leader, under loaded node in the cluster. For example, a leader receives a request from the outside world and sends this request to another node based on the strategy defined in the algorithm. If the receiving node has not been loaded sufficiently, the calculation is performed, and the result is sent back to the head known as the leader node, and further, the node sends back the result is sent back to the users. The occurrence is transferred to the underloaded node when the receiving node becomes overloaded. With the help of policies used by the algorithm, the node starts the instance and passes the result to the primary node, and then the primary node passes the results to the users.

Now, from a CCS point of view, what happens within the cluster when the connection agents are Leader, Sender and Recipient. The leader node accepts the job and sends it to the nodes using the send command. It continues to wait for the result from the nodes on the result and then sends it through the result. Fig. 12 shows the different ports on which agents communicate. Accept work from outside in parallel.

$$leader = \; accept.sleader$$

$$sleader = send.arleader \;+\; accept.rsleader$$

$$arleader = result.sresult \;+\; accept.rsleader$$

$$rsleader = result.leader + accept.leader$$

The node receives a request from the leader. A node can be underloaded or overloaded; in case underloaded, it calculates, and the result is conveyed through the result. If the node is overloaded, it is called the sender and sends information about its load through the information s and waits for the recipient to reply. A positive acknowledgement is sent if the recipient is ready to accept the request otherwise a negative. If a positive confirmation is received, the instance is transferred using the sending process; otherwise, it sends the information to another node. After transferring the instance, return to the original state to receive the job.

$$Sender \;=\; send.Sending \;+\; send.Computing$$

$$Sending \;=\; info\_\ s.Receiving$$

$$Receiving \;=\; info\_\ r.Migrating \;+\; NACK.Sending$$

$$Migrating \;=\; send\ job.Sender$$

The recipient is a node with a low load so that it can accept an instance request from the leader, or it can receive an instance request from an overloaded node. In case it agrees with the activity from the leader, it starts the instance and returns the result through the result. In case, it receives a request from an information-overloaded node s, it then sends a positive confirmation or a negative confirmation via the r information, then waits for an instance on the sending process. After the calculation, the result is sent to the leader through the result and control returns to its original state. Fig. 13 explains this.

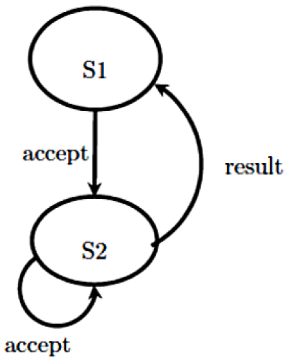

Fig. 11. Language Specification for a cluster

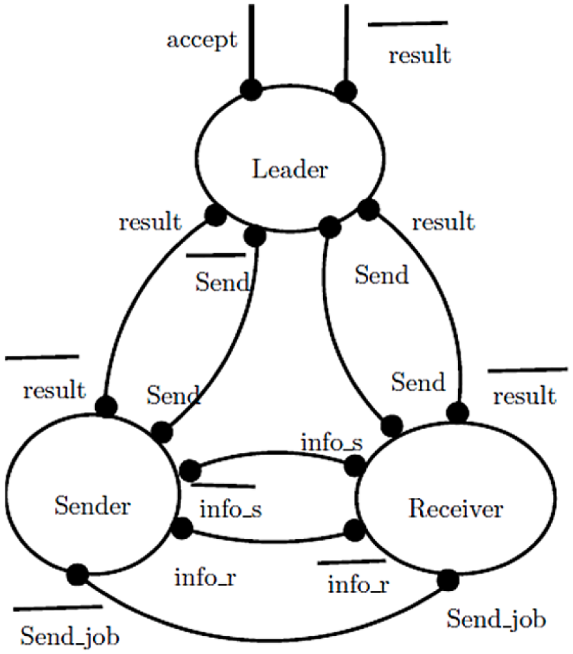

Fig. 12. Language Implementation of cluster

$$Receiver = send.Computing + info\_s.Sending$$

$$Sending = info\_r.Receiving + NACK.Receiver$$

$$Receiving = Send\ job.Computing$$

$$Computing = Result.Receiver$$

5.9. Bi-simulation proof for model

To demonstrate the bi-simulation concept, game theory is used to establish a weak double equivalence. In accordance with game theory, the attacker-defender, the attacker selects the process in the existing configuration and makes some movements for action with two actions, and the defender must respond to another process with the same action. Any number of moves can be performed by a player in weak bi-simulation. If the defender is not able to respond to any activity, then there is a loses, and if the defender keeps the attacker in an infinite loop, then the defender wins.

Attacker - Implementation
Defender – Specification
Initial state (I1, S1)

$$A : I1 \xrightarrow{accept} I2$$

$$D : S1 \xrightarrow{accept} S2$$

At the start, the state of the system is (I1, S1); our system accepts jobs and goes to the state (I2, S2). Therefore, the state achieved is (I2, S2).

$$A : I2 \xrightarrow{accept} I2,\ I2 \xrightarrow{\tau} I3,\ I2 \xrightarrow{\tau} I4,\ I2 \xrightarrow{\tau} I5$$

$$D : S2 \xrightarrow{accept} S2,\ S2 \Rightarrow^{\tau} S2,\ \ S2 \Rightarrow^{\tau} S2,\ \ S2 \Rightarrow^{\tau} S2$$

Now, there are four possibilities for the system at this stage; firstly, the sender node is overloaded; next, there is underloaded sender, third, a job is accepted by the receiver in this state, and fourthly, the job is received by the leader from the outside world. Three states are achieved by the above possibilities. These newly created states are (I3, S2), (I4, S2), (I5, S2).

$$A : I3 \xrightarrow{accept} I3,\ I3 \xrightarrow{\tau} I6$$

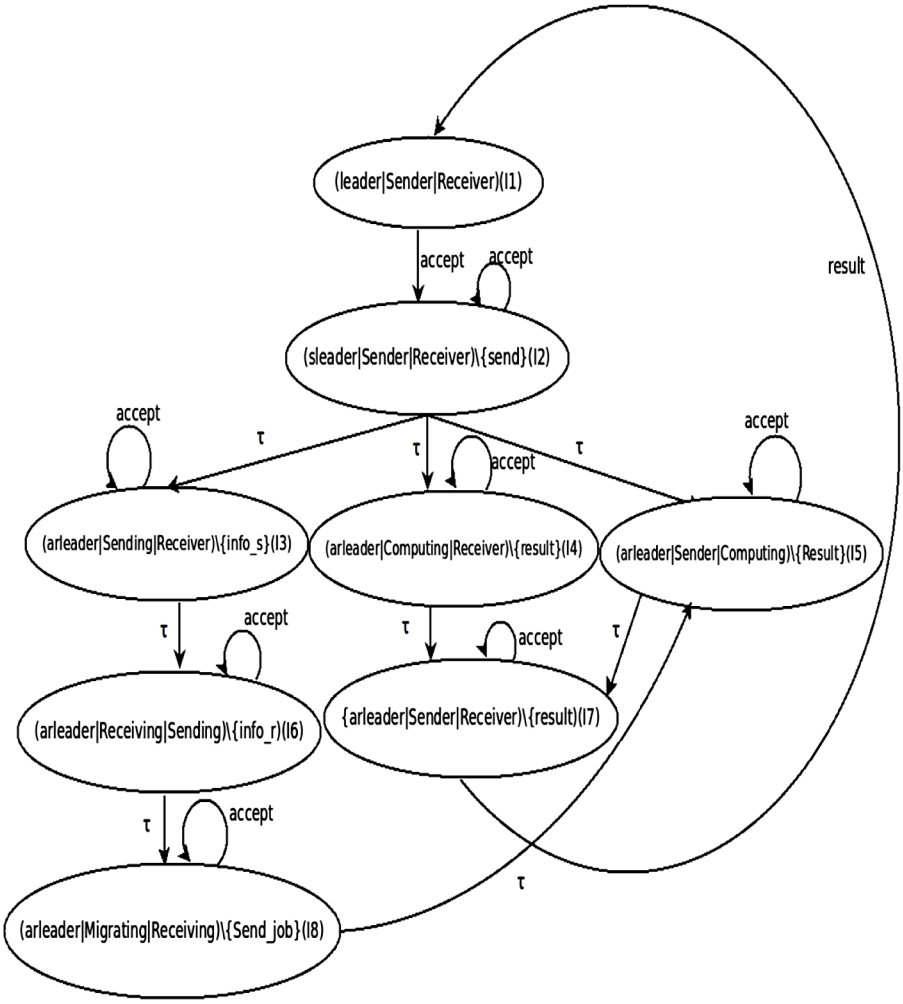

Fig. 13. States of all the agents at different instances

$$D : S2 \xrightarrow{accept} S2,\ S2 \Rightarrow^{\tau} S2$$

The job can be received from the user when the system is overloaded, or in another case, if the sender gets overloaded, it can send its information to any other node. Therefore, we get one new state (I3, S2).

$$A : I4 \xrightarrow{accept} I4, I4 \xrightarrow{\tau} I7$$

$$D : S2 \xrightarrow{accept} S2,\ S2 \Rightarrow^{\tau} S2$$

In case if the sender is under-loaded, then the system executes the job, and our system may accept a job from the user. Therefore, we get one new state (I7, S2).

$$A : I4 \xrightarrow{accept} I4,\ I4 \xrightarrow{\tau} I7$$

$$D : S2 \xrightarrow{accept} S2,\ S2 \Rightarrow^{\tau} S2$$

Receiver executes the job; in case it receives it, or there may be a new job received by the leader from the user.

$$A : I8 \xrightarrow{accept} I8,\ I8 \xrightarrow{\tau} I8$$

$$D : S2 \xrightarrow{accept} S2,\ S2 \Rightarrow^{\tau} S2$$

In another case, if sender gets overloaded and the information is directed to some other underloaded node, then data is received from underloaded node and leader receives a job from the outside world, which concludes to a new state (I8, S2).

$$A : I7 \xrightarrow{accept} I1$$

$$D : S2 \xrightarrow{accept} S1$$

The receiver computes and sends the result to the leader, and our leader later transmits the result to the user. Finally, the system comes to a new state. Hence, the system remains in an infinite loop.

$$A : I8 \xrightarrow{accept} I8,\ I8 \xrightarrow{accept} I5$$

$$D : S2 \xrightarrow{accept} S1,\ D : S2 \xrightarrow{accept} S1$$

The sender sends the job to the receiver on receiving information from receiver, and it results in the system to the (I5, S2) which means new state not found. Fig. 14, 15, demonstrates that the model comes to an infinite loop. Therefore, the defender wins according to the policy, i.e. attacker defender policy. Now, in game theory by changing the attacker and defender

Attacker - Specification
Defender - Implementation
Initial State (S1, I1)

$$A : S1 \xrightarrow{accept} S2$$

$$D : I1 \xrightarrow{accept} I2$$

In the starting, the system starts with an initial state (S1, I1) cloud acknowledge a new Job which leads to the new state. The new

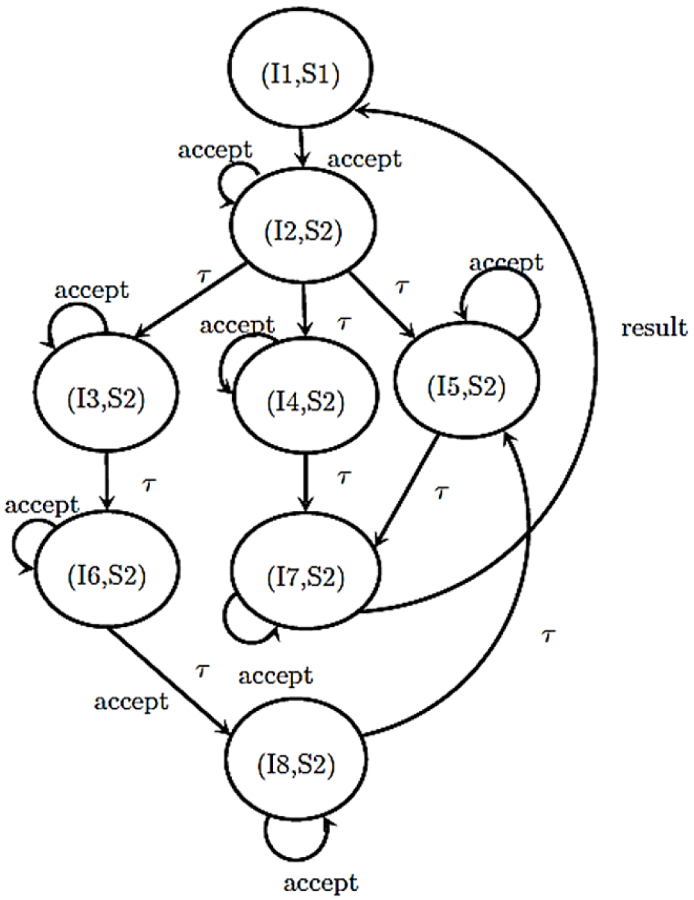

Fig. 14. State transition of the system

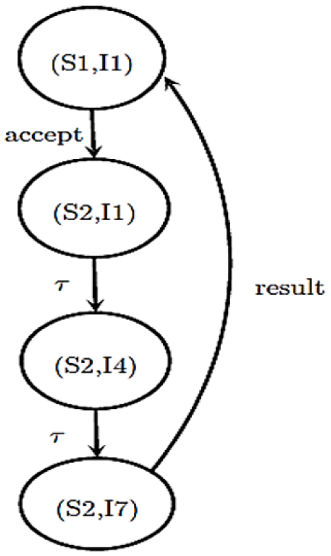

Fig. 15. State transition of the system

state is (S2, I2)

$$A : S2 \xrightarrow{accept} S2,\ S2 \Rightarrow^{result} S2$$

$$D : I2 \xrightarrow{accept} I2,\ I2 \xrightarrow{\tau} I4,\ \ I4 \xrightarrow{\tau} I7,\ I7 \xrightarrow{result} I1$$

Now the defender (specifying) has only two moves: either cluster can receive the job, or it can send the result to the user and return to the initial state. According to the system, it can also accept the job, or it can send info_s to receive and receive info_r from the recipient, then send the job to the recipient. The recipient receives the activities from the sender and, after calculating it, returns the result to the leader, who produces the system to its original state. Fig. 15 shows the change of state. And no new states have been found, and continue in an infinite cycle. Therefore, according to game theory, our model is correct.

6. Conclusions and future work

We have discussed various challenges in order to solve the problem of resource allocation. We have strained to give a solution to the issues of energy consumption and CO2 emissions by using server consolidation techniques. The model we propose makes the cloud system adaptable and scalable and because the decision on the allocation of resources is not caused by a single object. Load balancing, cost-effectiveness and dynamic allocation of resources through the development of a decentralized architecture and the implementation of a resource management protocol. We have demonstrated the correctness of the proposed protocol using the calculation of a communication system (CCS). A comparison shows that we significantly decrease overhead and increase network performance than the Fetahi Wuhib's [1] gossip approach. If all the machines in the cluster have the same configuration, our protocol acts like a gossip-based protocol. On the other hand, if all the machines have different configurations, our protocol behaves like a DHT-based approach. In the future, we will try to find a way to use DHT, in addition to creating clusters, even if all our machines have the same configuration.

References

[1] Fetahi Wuhib, Rolf Stadler, Mike Spreitzer, A Gossip Protocol for Dynamic Resource Management in Large Cloud Environments, in: IEEE Transactions on network and service management 9, 2012.
[2] R. Yanggratoke, F. Wuhib, R. Stadler, Gossip-based resource allocation for green computing in large clouds, in: 7th International Conference on Network and Service Management, CNSM 2011, Paris, France, 24-28 Oct. 2011, pp. 1–9.
[3] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, M. Parashar, in: N. Antonopoulos, L. Gillam (Eds.), Springer, 2010, pp. 195–217.
[4] Daniel Warneke, IEEE Member, Odej Kao, Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud, IEEE Transactions On Parallel And Distributed Systems 22 (6) (JUNE 2011).
[5] Daji Ergu, Gang Kou, Yi Peng, Yong Shi, Yu Shi, The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment, Springer Science+Business Media, LLC, 2011 on 19 May 2011.
[6] G. Lee, B.-G. Chun, R.H. Katz, Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud, in: 3rd USENIX Workshop on Hot Topics in Cloud Computing (Hot- Cloud'11), USENIX Association, Berkeley, CA, USA, June 2011.
[7] Fangzhe Chang, Jennifer Ren, Ramesh Viswanathan, Optimal Resource Allocation in Clouds, in: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, July 05-10, 2010, pp. 418–425.
[8] Jiyin Li, Meikang Qiu, Jain-Wei Niu, Zhong Ming YuChen, Adaptive Resource Allocation for Preempt able Jobs in Cloud Systems, in: IEEE International Conference on Intelligent Systems Design and Applications, 2010, pp. 31–36.

[9] Zhen Xiao, , Senior Member, IEEE, Weijia Song, Qi Chen, Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS 24 (6) (JUNE 2013).
[10] Yagiz Onat Yazir, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, Yvonne Coady, Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis, in: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing,, 2010, pp. 91–98. July 05-10,.
[11] P. Ganesan, B. Yang, H. Garcia-Molina, One Torus to Rule them All: Multidimensional Queries in P2P Systems, in: Seventh International Workshop on the Web and Databases (WebDB 2004), June 1718, Paris, France, 2004.
[12] Bin Liu, Wang-Chien Lee, Dik Lun Lee "Supporting Complex Multidimensional Queries in P2P Systems", Proceedings of the 25th IEEE International Conference on Distributed Computing Systems.
[13] Egemen Tanin, Aaron Harwood, Hanan Samet, Using a distributed quadtree index in peer-to-peer networks, Published online: 4 April 2006 in Springer-Verlag, 2005.
[14] H.V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, BATON: a balanced tree structure for peer-to-peer networks, in: Proceedings of the 31st international conference on Very large databases, Trondheim, Norway, 2005. August 30-September 02,.
[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker, A scalable Content-Addressable Network, in: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM, 2001, pp. 161–172.
[16] A. Rowstron, P. Druschel, Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, in: IFIP/ACM International Conference on Distributed Systems Platforms, 2001.
[17] W. Forrest, "How to cut data centre carbon emissions", http://www.computerweekly.com/Articles/2008/12/05/233748/How-to-cut-data-centre-carbon-emissions.htm. December 2008.
[18] Report to congress on server and data centre energy efficiency-public law 109-431. USEPA07b, U.S. Environmental Protection Agency, July 2007.
[19] M. Femminella, G. Reali, Gossip-based monitoring of virtualized resources in 5G networks, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2019, April, pp. 378–384.
[20] I. Mohiuddin, A. Almogren, Workload aware VM consolidation method in edge/cloud computing for IoT applications, Journal of Parallel and Distributed Computing 123 (2019) 204–214.
[21] R.L. Sri, N. Balaji, An empirical model of adaptive cloud resource provisioning with speculation, Soft Computing 23 (21) (2019) 10983–10999.
[22] J. Kommeri, T. Niemi, J.K. Nurminen, The energy efficiency of dynamic management of virtual cluster with heterogeneous hardware, The Journal of Supercomputing 73 (5) (2017) 1978–2000.
[23] S. Aslam, S. ul Islam, A. Khan, M. Ahmed, A. Akhundzada, M.K. Khan, Information collection centric techniques for cloud resource management: Taxonomy, analysis and challenges, Journal of Network and Computer Applications 100 (2017) 80–94.
[24] P.R. Theja, S.K. Babu, Evolutionary computing based on QoS oriented energy-efficient VM consolidation scheme for large scale cloud data centres, Cybernetics and Information Technologies 16 (2) (2016) 97–112.
[25] K. Gupta, V. Katiyar, Survey of resource provisioning heuristics in the cloud and their parameters, International Journal of Computational Intelligence Research 13 (5) (2017) 1283–1300.
[26] I. Hamzaoui, B. Duthil, V. Courboulay, H. Medromi, A Survey on the Current Challenges of Energy-Efficient Cloud Resources Management, SN Computer Science 1 (2) (2020) 1–28.
[27] M.K. Sohrabi, H. Azgomi, A survey on the combined use of optimization methods and game theory, Archives of Computational Methods in Engineering 27 (1) (2020) 59–80.
[28] R. Aggarwal, Resource provisioning and resource allocation in the cloud computing environment, International Journal of Scientific Research in Computer Science, Engineering and Information Technology (2018) 3.
[29] L. Li, J. Dong, D. Zuo, J. Liu, SLA-Aware and Energy-Efficient VM Consolidation in Cloud Data Centers Using Host States Naive Bayesian Prediction Model, in: In 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, IEEE, 2018, December, pp. 80–87.
[30] Buyya Rajkumar, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic, Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems 25 (6) (2009) 599–616.