Multi-Agent Model for Job Scheduling in Cloud Computing

Khaled M. Khalil, M. Abdel-Aziz, Taymour T. Nazmy, Abdel-Badeeh M. Salem

***Abstract*—** Many applications are turning to Cloud Computing to meet their computational and data storage needs. Effective Cloud Computing is possible, however, only if resources are scheduled well. Nowadays, Internet of Things (IoT) paradigms, management of such huge number of job processing systems is a challenging problem. In addition, heterogeneity is a typical characteristic of today's cloud (caused by incremental upgrades and combinations of computing architectures). This paper presents a multi-agent model for job scheduling in Cloud Computing Systems. We created a model based on the multi-agent systems paradigm. Then we got the model executed in NetLogo and the simulation input is Google Cluster Workload Traces data set. The model consist of group of autonomous agents and has an ability to cooperate with the other agents in the system. Due to these abilities of agents, the structure of the system is more suitable to handle dynamic changes and load as number of machines and requests increase. We compare different parameters for the scheduling model for hybrid and proposed multi-agent based architecture. Our simulation results indicate that a distributed architecture with multi-agent system and local blackboards for agent groups holding the cluster state yield 30% better average end-to-end resource utilization and 10% delay performance than the hybrid architecture.

***Keywords*—**Agent-Based Job Scheduling, Multi-Agent Systems, Job Scheduling, Cluster Scheduling, Internet of Things (IoT).

I. Introduction

THE amount of data in our world is exploding, and this is due to the extensive use of applications based on multimedia and social media. Analyzing such large data sets has become a key basis of competition. Cloud Computing (or simply cloud) infrastructure enables to transforms such information and data into actionable insights. Mell and Grance [1] define Cloud Computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Job Scheduling is a vital activity of the Cloud Computing [2]. The management of the cloud resources requires making scheduling decisions involving cloud resources assigned to users' jobs. The scheduling problem deals with the coordination and allocation of resources so as to efficiently satisfy the users' needs. Cloud users send their applications to the Cloud Computing System. Applications consist of specific number of jobs. Tasks are created per job and they may or may not depend on each other. Tasks generally require the use of different kinds of resources, e.g., computation, memory, communication (network), internal/external storage resources, GPU, or specific instruments.

The goal of the Cloud Scheduler is to meet user demands in terms of cost and response time while increasing providers' profit and resource utilization [3]. Due to the novelty of the Cloud Computing field, there is no standard task scheduling approach to handle the dynamic workload of users [2]. Limited communication bandwidth is preventing the current algorithms to be applied in cloud. The scheduler does not have control over the resources or the jobs. The complexity of cloud applications, the diverse user requirements and the system heterogeneity would result in inefficient scheduling in the case of a manual procedure. So the scheduler must make best-effort decisions to utilize the resources and provide best response to users' jobs.

The architecture of the schedulers has evolved over the last few years moving from monolithic design to more flexible, distributed and hybrid design (monolithic, two-levels, shared-state, distributed, and hybrid scheduling) [4], [5], and [6]. The monolithic schedulers are working in determined space and have the supervision to define the assignment over all available resources. All workloads are handled by the same scheduler and all tasks run through the same scheduling policy. The scheduling policy is simple and uniform, but this has led to increasingly sophisticated schedulers developed. To address these problems, a two-level schedulers have been developed by separating resource allocation and task placement [7]. This allows the logic of the task placement to be customized in the form of queues supporting the different users' applications. This approach has a drawback that the task allocation system cannot see all possible options and hence be subject to degrade the performance of the tasks execution. Shared-State schedulers share an out-of-date copy of the cluster state to the task allocation systems. This is resulting in conflicted and rejected requests to schedule tasks that may happen between schedulers due to dynamically changes to the resources. On the other hand, fully distributed schedulers have no coordination between them at all. In distributed architecture, schedulers are

Faculty of Computer and Information Science, Ain shams University, Cairo, Egypt.
Khaled M. Khalil (e-mail: kmkmohamed@gmail.com).
M. Abdel-Aziz (e-mail: mhaziz@cis.asu.edu.eg).
Taymour T. Nazmy (e-mail: timor.mohammad@cis.asu.edu.eg).
Abdel-Badeeh M. Salem (e-mail: absalem@cis.asu.edu.eg).

assigning resources to the workload [5]. Each scheduler has its own allocation policy and hence it is difficult to support complex jobs with specific policies. Hybrid schedulers seek to handle all the above issues by combining both monolithic and distributed architectures in one system. Two types of schedulers are in the system; a distributed scheduler for the short or low priority tasks and a centralized scheduler for complex and services (long duration) tasks.

The application of intelligent agent technologies is considered a promising approach to improve system performance in complex and changeable environments [8]. Thus, employment of multi-agent systems simulation can optimize the total system output. In this paper, we propose a multi-agent simulation model for the job scheduling in Cloud Computing. We are running our experiments using data of one cell cluster of more than 12 thousands machines and more than 52 thousands jobs with millions of tasks. We are using google cluster data [9] for our experiments. Finally, we compare the results of the proposed model with the hybrid architecture.

The remainder of this paper is organized as follow, in Section II we provide literature review on related work. In Section III, we discuss the proposed multi-agent model while in Section IV we present and analyze the experiments results. Finally, Section V concludes the study.

II. BACKGROUND

A. Cloud Job Schedulers

Christodoulopoulos et al. [10] and Schopf [11] mentioned that scheduling in such dynamic environments can be viewed as hierarchical problem with two levels. The first level finds the resources needed for the task execution. There are many variations in this level; such as selecting the resources based on the best-fit manner, using prediction to estimate availability of the resources, and using prediction to estimate load of other tasks running on the resources machines. At the second level, tasks are assigned to the selected resource for execution. At this level, tasks may wait for the resources to be available or other tasks with lower priority can be preempted for the higher priority tasks to run. Each scheduler should take in consideration the possible expand of needed resources by the tasks to avoid high rate of failure. In addition, Schopf [11] discussed subtasks that describe the detailed behavior of each level.

Salot [2] did a survey of various scheduling algorithms in Cloud Computing environment. There are main two categories of scheduling algorithms; Static Scheduling algorithms and Dynamic Scheduling algorithms. Static Scheduling algorithms assume a prior information about all machines in the cluster and scheduling decisions are made before execution of the tasks. Singh et al. [12] did a survey on the different approaches in Static Scheduling such as Homogeneous Distributed Computing System (HMDCS), Heterogeneous Distributed Computing System (HTDCS), Monte Carlo Algorithm (MCA) … etc. If there are changes to the machines, then the static scheduler must know the changes to avoid failure in scheduling decisions. This approach depends heavily on the assumption that all machines share their status globally with the scheduler on a real-time basis. This is not efficient from the practical side. On the other hand, Dynamic Scheduling algorithms study the cluster state for scheduling decisions before assigning tasks and during application execution. Kumar and Balasubramanie [13] did a survey in their related work on different dynamic scheduling approaches.

Also, Salot [2] went through further categorization of these two types of scheduling algorithms as: First Come First Serve, Round Robin, Min-Min, Max-Min, Most-Fit, and Priority. Each of these subcategories has their own advantages and limitations. First Come First Serve is fast and simple but will reduce the efficiency of the scheduler in response to higher priority tasks in front of the long low priority tasks. Round Robin is based on the First Come First Out approach to be fair in handling users' jobs. But long tasks will suffer from high rate of interruptions and if there is no way to backup and restore the tasks results then long tasks will be delayed for a long time. Min-Min is biased towards small tasks to be executed at first but this will delay long tasks with high priority. In addition, high priority tasks will be kept waiting for execution for a long time if system has a stream of short tasks. Max-Min is biased towards the large tasks which is an opposite of the Min-Min. Most-Fit is working on the assumption that each task should have a most-fit set of resources available in the system for execution. Tasks in this technique are suffering from low rate of success. Priority schedulers are based on giving a priority for each task and get queues for each priority to process them in First Come First Out manner. Local priority can be given to the task for serving within the machine that accepted the task assignment. All these algorithms were handled as policies applied to the resources queues in the cloud scheduling system. First Come First Serve is the most common policy with applied priorities.

Dave et al. [14] listed the main performance metrics of scheduling algorithms; makespan, execution cost, job rejection ratio, execution time, and user satisfaction level. Makespan is the time at which all the work in the workload was completed. Execution cost is the total cost of execution of the job tasks. Job rejection ratio is defined as the total rejected jobs to the total number of jobs submitted. Execution time is the time from when the job tasks are scheduled till it gets its final results. User satisfaction level is defined as how far the scheduler system satisfy the terms defined by the service provider to users. Burkimsher et al. [15] added flow and peak in-flight count metrics. Flow can be defined as the count of number of jobs completed over the makespan. Peak in-flight count is the maximum number of in-flight jobs at any time. For more discussions about these metrics check [15].

On the other hand, various schemes are used to decide which particular task to run [8]. Parameters that might be considered including task priority, resource availability, license key if job is using licensed software, execution time allocated to user, number of simultaneous jobs allowed for a

user, estimated execution time, elapsed execution time, availability of peripheral devices, occurrence of prescribed events, task dependency, file dependency, and operator prompt dependency.

B. Schedulers Architecture

Cloud Scheduler has multiple goals: using the cluster's resources efficiently, working with user-supplied constraints, scheduling applications, and having the acceptable level of fairness. To reach these goals, there are five main scheduler architectures: monolithic, two-level, shared-state, distributed, and hybrid.

Monolithic Schedulers like Google Borg [3] are a centralized and static schedulers that make scheduling decisions by one and only one scheduler. These types of schedulers do not support parallel tasks. Google Borg runs everything in the cloud so it is very complicated. Borg has two priorities bands High and Low and all jobs are statistically scheduled.

Two-Level Schedulers like Apache YARN [7] and Mesos [16] are centralized and static schedulers that separate the resource allocation from the task management operations. In such schedulers, systems have one resource manager that grants resources to multiple independent schedulers, in which each scheduler has certain policies for resources queuing for resource allocation based on the user preferences. This type of schedulers are less complex than the Monolithic ones as they delegate the pre-application scheduling work to the applications themselves while managing the distribution of resources between applications and enforcing fairness.

Shared-State Schedulers like Google Omega [4] are centralized and static schedulers with a shared state of the cluster state. They are a successor to the two-level schedulers and they take the resources offered one degree further. All resources available in the cluster are offered to the applications and conflicts are resolved at task execution time. All applications schedulers have the same view of the cluster resources which increase the scheduling performance and resources utilization. Applications with high priority are allowed to preempt lower priority and stay within the limited number of jobs and resources assigned to the application.

Distributed Schedulers like Sparrow [5] are like multiple isolated schedulers to serve the incoming workload. Each one of these schedulers works with its local, partial, and often out-of-date view of the cluster. That's why it is difficult to enforce global fairness. More or less, supporting complex-applications and specific applications policies are more complex than other architecture of schedulers.

Hybrid Schedulers like Microsoft Mercury [6] try to address all the drawbacks of the previous architectures. Two approaches are incorporated to handle the workload, group of distributed schedulers for very short and/or low-priority tasks and one centralized scheduler for the rest of workload (high priority tasks and/or long tasks like system services).

C. Multi-Agent Systems for Cloud Resource Management

The multi-agent systems features (dynamic, flexibility, autonomy, and learning) are exactly the same features that a Cloud Computing System needs for the self-management of its resources. The scheduling decision process is complex, due to the variability of the demand for services and the lack of information on the resources. This is why the agent-based job scheduling is suitable for the efficient allocation of resources enabling the dynamic and automatic adaptation.

Some of the essential characteristics of Cloud Computing include resource pooling and resource sharing. In cloud, computing resources are pooled to serve multiple users, and data are shared by a broad group of cross-enterprise and cross-platform users. Sim [17] mentioned that resource pooling and sharing involve (1) combining resources through cooperation among cloud providers; (2) mapping, scheduling, and coordination of shared resources; and (3) establishment of contracts between providers and consumers. In Agent-Based Cloud Computing, cooperation, negotiation, and coordination protocols of agents are adopted to automate the activities of resource pooling and sharing in clouds. All the above-mentioned challenges provide the motivations for adopting autonomous agents to allocate resources taking in consideration the dynamically changing resource demands.

In addition, Sim [7] discussed Agent Based Cloud Computing as agents do cloud service discovery, service negotiation, service composition, and cloud crawlers. Agent Based Cloud Service Discovery is to run a query to cloud services registered to match consumer functionality. Agent Based Service Negotiation is to establish SLAs (Service Level Agreements) between consumers and brokers. Agent Based Service Composition is to group related services from different suppliers into a single bundle of services. Agent Based Cloud Crawlers are agents that collect information about the cloud service providers.

Gąsior and Seredyński [18] discussed a decentralized implementation of multi-agent system for job scheduling. But they are focusing on scheduling jobs based on a security model to avoid dispatching jobs to untrustworthy resources. The security model incorporates awareness during the scheduling operation to match job's security requirements by the user with security guidelines defined for each Cloud site. Kanmani and Sukanesh [8] discussed an optimization of the scheduling using the multi-agent system paradigm and genetic algorithm for hybrid cloud. They are looking for improving the quality of service to the end user of the cloud system. They are adding an extension to CloudSim [19] for their experiments. They are normalizing the resource allocation using multi-agent genetic algorithm which takes action to change the resources allocation rates depending on the current behavior of the system.

D. Cloud Simulators

Cloud Simulation is used in evaluating architectures, algorithms, and strategies that are under research and development tackling many issues such as resource

management, application scheduling, and workload execution [19]. As the exact cloud production environment may not be accessible to the developers at the early stages of development; simulations give an overall idea on the related parameters, resource requirements, performance, and output. In addition, due to the complexity of the schedulers, Cloud Simulators are become much important [20].

Cloud Simulators are quite generic like CloudSim [19] and EmuSim [21]. But some of them tend to be more focused on simulating specific functions like peer-to-peer and overlay networks as PeerSim [22] and OverSim [23]. Suryateja [20] provided a comparative analysis of cloud simulation tools.

III. The Proposed Multi-Agent Model

User applications submit jobs to the scheduler system. Each job consists of one or more tasks that need to be scheduled to consume the cloud resources for a specific period of time. The time needed for the task is based on the task execution pattern. During the task execution time, resources assigned to the task are reserved for the task event if not totally consumed.

Scheduling a task can be defined in terms of the following agents:

1- Applications Agents: a set of applications agents created by the users.
   a. Application Class includes ID, Username, and User Preferences fields.
   b. Application Agents do submit the jobs and do report the status of them back to the user.
2- Job Agents: a set of jobs agents submitted by users' applications.
   a. Job Class includes ID, Application ID, User, and Job Name fields.
   b. Job Agents do submit the tasks for resources assignment. In addition, they are responsible for reporting the status and result of the tasks back to the job agent.
3- Task Agents: each job is consisting of a set of tasks agents. Tasks are the objects that can be assigned to resources.
   a. Task Class includes ID, Job ID, Attributes, Priority, Resources Requests, and Requested Start Time fields.
   b. Task Agents are responsible for reporting task status back to the parent job agent.
4- Machines Agents: a set of machines that hosts the resources for assignment.
   a. Machine Class includes ID, Name, Resources Set, and Attributes fields.
   b. Machine Agents are responsible for reporting the health and other statistics of the machine and to the cloud service manager.
5- Queues Agents: a set of resources queues. Each queue has a policy for sharing resources between applications.
   a. Queue Class includes ID, Policy, and Queue Data Structure fields.
   b. Queues Agents are responsible for reporting the queue status back to the cloud service manager.
6- Scheduler (s) Agents: a set of schedulers. Each scheduler has an algorithm to handle the scheduling decisions.
   a. Scheduler Class includes ID, algorithm, and metrics data fields.
   b. Scheduler Agents are responsible for handling scheduling decisions and report back performance statistics back to the cloud service manager.

Other Classes:

1- Hard Constrains: a set of constrains which must not be violated by the scheduler. Such as running the data tasks near to the data source to avoid exhausted network channels. Hard Constrain Class includes ID, Constrain Type, and Constrain Value field.
2- Soft Constrains: a set of constrains that can be relaxed if necessary to improve the scheduler performance. Such as avoiding resources unfairness, minimizing machines fragmentation, and reducing the ratio of evicted tasks. Soft Constrain Class includes ID, Constrain Type, and Constrain Value field.
3- Resources: a set of available resources to which tasks can be assigned. Resource Class includes ID, Type, and Value field.

Fig. 1 shows the agents and classes of the proposed scheduler model. Arrows are showing the relationship between the classes. A group of local blackboard spaces is kept shared between all agents with an up-to-date state of the cluster resources. Each machine agent is responsible for updating the nearest blackboard with the machine updates. If machine agent failed to communicate the updates to the blackboard for a specific period of time then the machine is considered as lost and all tasks running at that machine is marked as lost.

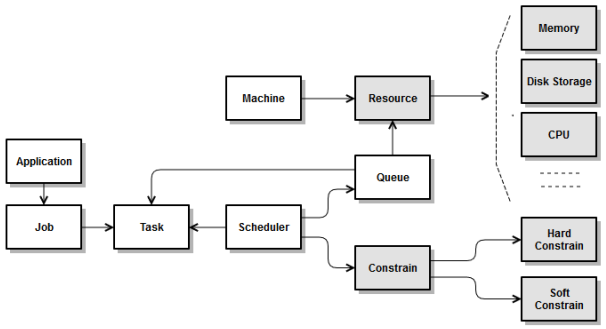

Fig. 1 Proposed System Agents and Classes

IV. Results and Discussion

A. Technical Characteristics of the Simulator and Google Cluster Data

The model is developed using NetLogo 5.3 [24]. We preferred accuracy of the modeling over execution time. So the early development versions of the model took long time for

simulation. We followed the instructions from Railsback et al. [25] to improve execution time of the model and to detect the bottlenecks. We have extra lessons learned to add to the checklist for NetLogo simulation of complex and large scale models: (1) avoid using filtration with "with" keyword for the agent sets. Instead use the "ifelse" block inside the agent set operations, (2) avoid looping in a list within the agent set block. It is showing slow performance compared to checking if the item is a member of the list or not, (3) nested operations of two or more groups of agent sets will draw the performance of the simulation model at all, and finally (4) avoid showing updates in the simulation view as this will slow down the model execution.

We used Google Cluster Trace Data [26] as input for the workload of the simulation model. The data set consists of more than 40GB of row text data collected from one cell in Google Data Center of 12,583 machines over 29 days. 552 users were submitting applications with 350,926 unique jobs. The data set includes users' jobs, tasks and heterogeneous set of machines. Machines, jobs and tasks events include when machines were added or removed from the cluster, when jobs were submitted, and full details about the tasks attribution and execution. Tasks per job are ranging from one task per job (low priority and short task) to more than 20 thousands tasks (long term service) per job. Google normalized the machines resources to specific ranges for privacy aspects and we saw no reason from our side to rescale back the values. Fig. 2 shows the state transitions of a task in the data set. More analysis and details about the data set are provided by [27] and [28]. Tasks start with Un-submitted state. Once the task is submitted, it is moved to the Pending state. When the task is scheduled and assigned to a machine, the state is changed to Running. If the task failed, killed, machine lost, then the task state will be changed to Dead. If the task is evicted due to a higher priority task then the task state is changed to Evict. Dead and Evict tasks will be resubmitted to be in the Pending state. Once the running task finish, it is moved to the "Complete" state. Finally, the task is cleaned and removed from the system.

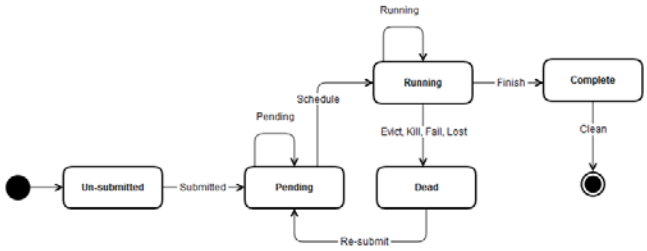

Fig. 2 State Transitions of a task in Google Cluster Trace Data

To reduce the amount of data loaded to the model, we went through some actions to cleanse the trace data. We have removed jobs and tasks that were added before the trace window and are still running. The trace data has all jobs with tasks with the same requirements. In addition, the tasks execution details included are for the version of Google Borg deployed at the cluster by the time the records extracted. So we eliminated the millions of duplicate records for the job tasks into one record per job including the number of tasks and tasks requirement for the job. This resulted in one record per job and removed records of all tasks in the data set.

B. Features of the Proposed Model

Our proposed model handles the assignment of the tasks into two phases. Phase one is to select the feasible machines. It starts with selecting random machines that satisfy the resource requirements of the task. Phase two is to calculate score for each selected machine based on the E-PVM formula mentioned in [29]. Then select the machine with the lowest score to assign the task to it. In our model, we are grouping machines by score so that selecting machines from the highest scores is trivial. This is supported by the local group blackboard. Local blackboard rather than a global blackboard to reduce the communication overhead through the network. This also reduces the fragmentation of the job tasks as being assigned to a specific group of resources in one area. Our model is biased towards getting highest priority tasks complete first. In case of resources shortage, low priority tasks are subject to be preempted to secure resources for higher priorities tasks.

We ran our experiments for several times using different seed numbers for random number generator. In all of our experiments, we measure task throughput, job throughput, preemption rate, and resources utilization. Reported metrics can be extended easily in the proposed model. Fig. 3 (a, b, c, and d) show values of the scheduling metrics. To stress the model and reach the machines limit, we've selected to run the experiments with 500 out of the 12 thousands machines available machines.

The user interface of the model includes parameters for the simulation. Table I shows the model parameters.

Table I Proposed Model Parameters for Job Scheduling

Parameter	Range	Value	Description
Number of Schedulers	1 - ∞	5	This is the number of active schedulers agents to run in parallel to schedule the tasks.
Trace Time Map	1µs - ∞	1000 µs	This is to map the simulation tick (time) to the timespan from the trace data. Time in the trace data is in micro seconds.
Number of Machines	1 - 12561	500	A random sample of specific size of all available machines to be involved in the resource assignments. Machines are selected randomly from the available set of machines.
Percent of resources per Machine	0.1 – 0.9	0.5	Percent of allowed resources per machine for assignment. Assigned tasks must not consume all resources available in the machine. Machine OS and machine agent should have some computation power and memory space to operate. In addition, high priority tasks have the possibility to burst and request more resources within the same machine it is running at.

Parameter	Range	Value	Description
Task Max Length	1 tick - ∞	3 ticks	Max length of any task generated during the simulation.
Tasks Queue Size	1 - ∞	50000	Maximum number of tasks assigned to each queue in the system.
Minimum number of machines feasible for scoring	1 - ∞	20	Minimum number of machines randomly selected for feasibility check before scheduling.
Marginal Cost Rate	0.1 – 0.9	0.3	The rate in calculation of the cost of scheduling the task to the machine based on the available resources in the machine and the number of tasks running in the machine.

Fig. 3.a shows the available and assigned memory to tasks over time. Due to an out-of-data status of the cluster resources, hybrid schedulers were not being able to find more resource for the tasks to run. The same is applied to the assigned CPUs as per Fig. 3.b. Resource consumption kept in the range of 40%. 50% is the parameter of the maximum allocated to tasks as hard constrain and the 10% is the range in which no task fit the available resources in machines. Resource utilization is improved by 30% in the proposed agent-based model. Fig. 3.c simply shows the status of the jobs compared to the submitted jobs during the experiments window. As improved throughput of tasks and utilized resources, we've got more jobs finished earlier. Fig. 3.d shows the tasks per status for the hybrid and proposed agent-based architectures. We've got 60 thousands tasks in the queue as per the limit parameter. Remaining tasks are kept un-submitted until tasks were cleaned from the queue. 10% improvement in tasks throughout due to improved resources utilization. This is resulted in more tasks complete and less number of tasks in the pending state. The figure does not show the lost tasks as it is the same for both architectures. When a machine is lost due to some reasons, all the tasks agents assigned to them will not send their status back to the job agent. After a specific timeout (1 tick in our simulation) the tasks were considered dead and resubmitted again to the queue. In summary, our simulation results indicate that a distributed architecture with multi-agent system and local blackboards for agent groups holding the cluster state yield 30% better average end-to-end resource utilization and 10% delay performance than the hybrid architecture.

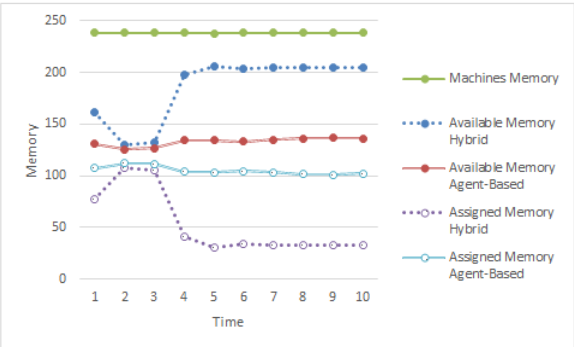

Fig. 3.a Machines Memory

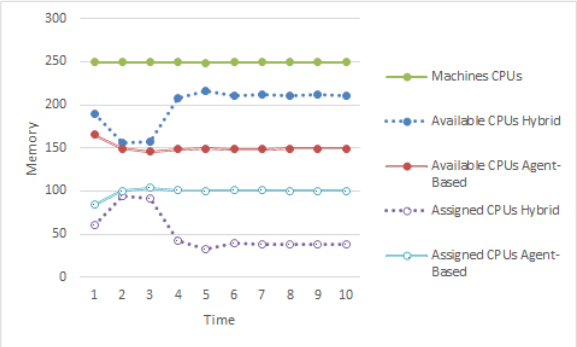

Fig. 3.b Machines CPUs

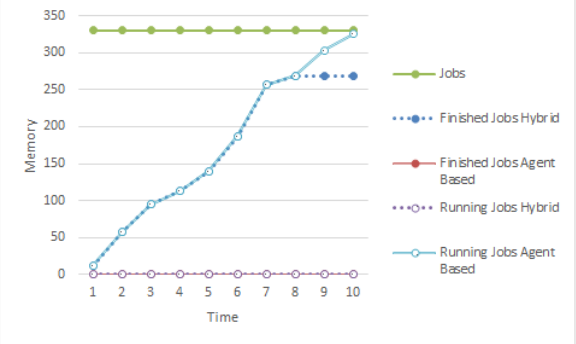

Fig. 3.c Jobs Status

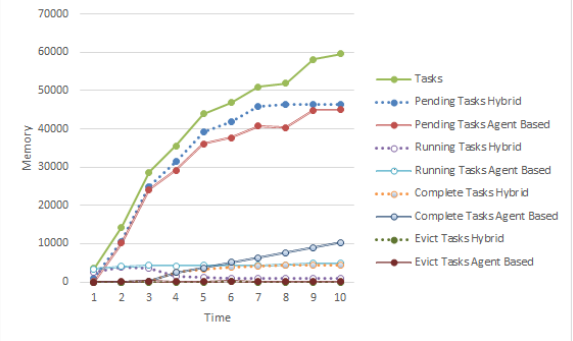

Fig. 3.d Tasks Status

V. Conclusion and Future Work

Due to the developing rate of trade, industry, and science world; scheduling is considered as one of the main discussions in cloud environment. As providing scheduling approaches which can minimize tasks runtime and increase operational power has remarkable importance in these categories. This paper presents a multi-agent model for job scheduling in Cloud Computing. Scheduling user applications is performed as a means to balance the load of the cloud system resources in order to improve the performance and throughput of the system. We have developed a NetLogo model with the proposed multi-agent model behavior and got Google Cluster Trace Data as input to the simulation. We got the experiments results shown and analyzed comparing the hybrid and proposed multi-agent based scheduler architectures. In addition, we analyzed the performance metrics of the proposed system. The proposed model is extendible to get more constrains on assigning resources applied as access to local group resources is applicable at any time. We consider this

research to be the first step towards more adoption of the multi-agent paradigm in the job scheduling domain. We are going to get the communication, negotiation and scheduling decisions improved by adopting machines learning algorithms for the scheduling agents.

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, 2011.

[2] P. Salot, "A Survey of Various Scheduling Algorithm in Cloud Computing Environment," *IJRET: International Journal of Research in Engineering and Technology*, vol. 02, no. 02, ISSN: 2319-1163, 2013.

[3] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proc. The Tenth European Conference on Computer Systems*, article no. 18. ACM, 2015.

[4] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proc. SIGOPS European Conference on Computer Systems (EuroSys)*, ACM, Prague, Czech Republic, pp. 351-364, 2013.

[5] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, Low Latency Scheduling," in *Proc. The Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 69-84, 2013.

[6] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga, "Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters," in *Proc. USENIX Annual Technical Conference*, pp. 485-497, 2015.

[7] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: yet another resource negotiator," in *Proc. SOCC '13 The 4th annual Symposium on Cloud Computing*, article no. 5, 2013.

[8] A. Kanmani and R. Sukanesh, "Optimized Scheduling using Multi-Agent Based Genetic Algorithm for Hybrid Cloud," *International Journal of Advanced Engineering Technology*, vol. 7, no. 1, pp. 980-983, 2016.

[9] Google Cluster Workload Traces. Available: https://github.com/google/cluster-data

[10] K. Christodoulopoulos, V. Sourlas, I. Mpakolas, and E. Varvarigos, "A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in Grid networks," *Computer Communications*, vol. 32, no. 7-9, pp. 1172-1184, 2009.

[11] J. M. Schopf, "A General Architecture for Scheduling on the Grid," *Journal of Parallel and Distributed Computing (Special Issue on Grid Computing)*, 2002.

[12] K. Singh, M. Alam, and S. K. Sharma, "A Survey of Static Scheduling Algorithm for Distributed Computing System," *International Journal of Computer Applications*, vol. 129, no. 2, pp. 25-30, Published by Foundation of Computer Science (FCS), NY, USA, November 2015.

[13] S. K. S. Kumar and P. Balasubramanie, "Dynamic scheduling for cloud reliability using transportation problem," *Journal of Computer Science*, vol. 8, no. 10, pp. 1615-1626, 2012.

[14] Y. P. Dave, A. S. Shelat, D. S. Patel, and R. H. Jhaveri, "Various Job Scheduling Algorithms in Cloud Computing: A Survey," in *Proc. International Conference on Information Communication & Embedded Systems*, IEEE Computer Society, pp. 1-5, ISBN No. 978-1-4799-3834-6, Chennai, India, February 2014.

[15] A. Burkimsher, I. Bate, and L. S. Indrusiak, "A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2009-2025, 2013.

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[17] K. M. Sim, "Agent-Based Cloud Computing," *Journal IEEE Transactions on Services Computing Archive*, vol. 5, no. 4, pp. 564-577, 2012.

[18] J. Gąsior and F. Seredyński, "A Decentralized Multi-agent Approach to Job Scheduling in Cloud Environment," Angelov P. et al. (eds) Intelligent Systems'2014, *Advances in Intelligent Systems and Computing*, vol. 322. Springer, Cham, 2014.

[19] R. N. Calheiros, R. Ranjan, C. E. AF. De Rose, and R. Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," arXiv preprint arXiv:0903.2525, 2009.

[20] P. S. Suryateja, "A Comparative Analysis of Cloud Simulators," *I.J. Modern Education and Computer Science*, vol. 4, pp. 64-71, 2016.

[21] R. N. Calheiros, M. A. Netto, C. AF. De Rose, and R. Buyya, "Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications," *Software: Practice and Experience*, vol. 43, no. 5, pp. 595–612, 2013.

[22] A. Montresor, and M. Jelasity, "Peersim: A scalable p2p simulator," *Peer-to-Peer Computing, IEEE Ninth International Conference*, pp. 99–100, 2009.

[23] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," *IEEE Global Internet Symposium*, pp. 79–84, 2007.

[24] NetLogo. Available: https://ccl.northwestern.edu/netlogo/

[25] S. Railsback, D. Ayllon, U. Berger, V. Grimm, S. Lytinen, C. Sheppard, and J. Thiele, "Improving Execution Speed of Models Implemented in NetLogo," *Journal of Artificial Societies and Social Simulation*, vol. 20, no. 3, 2017.

[26] Google Cluster Trace Data. Available: https://github.com/google/cluster-data

[27] Z. Liu and S. Cho, "Characterizing Machines and Workloads on a Google Cluster," in *Proc. ICPPW The 41st International Conference on Parallel Processing Workshops*, pp. 397-403, 2012.

[28] C. Reiss, A. Tumanvov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. The Third ACM Symposium on Cloud Computing*, article no. 7, 2012.

[29] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstorm, and A. Keren, "An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster," *Journal IEEE Transactions on Parallel and Distributed Systems archive*, vol. 11, no. 7, pp. 760-768, July 2000.