Optimizing Resource Allocation in Cloud

Yong Shi, Ignatius Jyosthna Lingareddy, Kun Suo, and Tu N. Nguyen
College of Computing and Software Engineering
Kennesaw State University
Marietta, USA
Contact: yshi5@ kennesaw.edu

***Abstract*— As an emerging technology, Cloud computing is vastly superior to traditional self-hosting computing methods because it separates computations from infrastructure. Every cloud provider must provide task scheduling, a resource allocation function that involves work being delegated to computational resources within the cloud system through a broker. Only a few of cloud task scheduling methods are accessible, such as Sufferage, Max-Min, and Min-Min; nevertheless, each is constrained by performance trade-offs. We propose a new task scheduling algorithm MinPlus for resource allocation, which exceeds several existing algorithms in terms of performance. Experiments are conducted for the MinPlus algorithm with the free planform CloudSim, and results are compared to existing algorithms that show decreased turnaround time, enhanced resource load balancing, and increased throughput.**

Keywords*— *Task Scheduling, Resource Allocation, Min-Min, MinPlus

I. Introduction

The cost barrier is lower with cloud computing than with traditional computer solutions, and customers are only billed for the resources they consume. Developers can more effectively use their development time by removing the complexity of procuring, configuring, providing, and maintaining computer resources. Additional advantages of cloud computing include scalability, high composability, and integrations with other cloud services. Most providers allow users to completely tailor their bandwidth, storage, computing power, etc. Cloud computing is used in some way by numerous businesses and organizations. Distributed/parallel computing, network topology, and job scheduling are some of the elements of cloud computing that are currently being researched.

In this paper, we first discuss existing algorithms for resource allocation, then propose a new task scheduling algorithm called MinPlus, followed by experimental results and analysis, as well as conclusion and future work.

II. Related Work

In cloud, tasks must be allocated to computer resources on single-threaded processors as well as distributed or networked computing systems. Every cloud provider seeks to maximize productivity and cut costs by distributing tasks across resources in a scalable, flexible way. The execution of activities by cloud service providers involves both multiplexing and multitasking, which mean doing numerous tasks at once and arranging multiple flows synchronously. While load balancing and overall service quality are both crucial elements, many algorithms intend to increase throughput while shorten completion times. The algorithms FCFS (First come First served), Min-min, Max-min, Round Robin, and Sufferage are examples of those developed by academics during the past few decades [1].

Every task scheduling technique is applied to minimize one performance parameter while compromising on other metrics. FCFS, which assigns tasks to resources on the order of request time, is the simplest scheduling strategy. Round Robin is a technique that enables tasks to share computing resources for an equal amount of time, although it is more typically employed with single-threaded systems, and for Round Robin, turnaround times are lengthy. Using the Min-min technique, computer resources are allocated for the job that has the highest likelihood of being finished first. The Max-min strategy, on the other hand, allocates resources to the task that has the lowest likelihood of being finished first. According to a priority strategy, each job is given a priority, and jobs with the same priority are finished in the order they were added to the queue [2]. Tasks with higher priorities are completed before those with lower priorities.

When designing task scheduling algorithms consideration should be given to load balancing to maintain continuous resource use and prevent any resources from being overburdened or underutilized. For load balancing, various factors will be assessed such as storage, CPU, and memory. Numerous load balancing techniques have been developed by researchers [5, 6]. The family of static algorithms falls under one of these categories and requires the user to be familiar with the system beforehand. Dynamic algorithms are another type that doesn't need any prior knowledge [3].

Analysis shows that researchers have employed heuristic algorithms to solve the scheduling issues [7]. Some research focuses on VM deployment methods that consider disk utilization when analyzing the features of data-intensive processes and CPU usage when operating the cloud system employed in KIAF [8]. For the scheduling and evaluation of scheduling parameters, the authors of [9] employ a pre-processing ETC matrix technique.

III. MinPlus for resource allocation

To minimize overhead, encourage quick performance, and enhance load balancing for various resources across computer

resources, task scheduling aims to plan work on computing resources as effectively and efficiently as feasible. In this section, we will first discuss three well-known and related job scheduling algorithms [10, 11]. Next, we will lay the groundwork for the proposed algorithm, MinPlus, and provide a thorough explanation of the reasoning behind each policy.

Each job in the task queue must first be assigned to the virtual machine that will finish it as quickly as possible, according to the Min-Min scheduling principle. The run time of a job on a virtual machine and how long it takes virtual machine to become accessible are added to determine the completion time. There must be a queue in which tasks need to wait before being assigned, assuming a machine typically handles only one job during a certain period. After every possible combination has been tried, the assignment with the shortest turnaround time is then handed to the virtual machine. Brokering is preferred over high-requirement activities for tasks with low computational requirements.

The Sufferage scheduling policy is based on the "sufferage" number, which is generated for each queued job and reflects tasks that "suffer" from the system by resulting in an inefficient pairing. The sufferage heuristic describes the difference for each queued work between the first computed minimum completion duration and the second minimum completion duration.

In contrast to Min-min, which selects the task that will appear to be completed first, Max-min decides which task will be completed last. This prioritizes jobs that demand more resources to prevent long execution queue wait times and subsequent starvation.

The first completion time with the smallest delay, the first completion time with the longest delay, or the completion time with the largest sufferage value are the three most popular techniques for distributing tasks to cloud resources. Some approaches can cause the cloud system to behave unpredictably such as starvation and other unexpected scenarios. Few of these methods employs resource load balancing, rather they apply straightforward heuristics based on completion lengths. More complex heuristics must be employed to concentrate on several performance indicators because the variety of the data and jobs is more than anticipated. A unique strategy is needed to address the problems with the existing algorithms.

[12] proposes an algorithm BMin based on the Min-Min algorithm. BMin distributes a specific task to a specific virtual machine in each round given unassigned tasks and available virtual machines by first figuring out how long it will take for each task to be completed on each virtual machine. To choose the "best" virtual machine for a particular task, they use a different approach than the original Min-Min method. Furthermore, it considers the variance and distribution of completion times to make final decision which task should be selected in the current round to acquire resource from the cloud. BMin outperforms the original Min-Min algorithm [12]. However, it does not account for other factors that can have an impact on how resources are allocated, like how long a given job takes to run compared to other tasks running on the same virtual machine, or how quickly various virtual machines complete a same operation.

We propose a new algorithm MinPlus, which not only calculate the completion time, but also take into consideration other aspects such as comparison of execution time and analysis of virtual machine available time. Details of our algorithm are as follows.

MinPlus Algorithm:

Suppose we have a set of m tasks (same as cloudlets in the CloudSim package) $C = \{c_1, c_2, \ldots, c_m\}$, and a set of n virtual machines $VM = \{vm_1, vm_1, \ldots vm_n\}$ in cloud, the goal of our algorithm is to assign each task from C to a virtual machine in VM and gain the best turnaround time, throughput, and virtual machine load balance.

Like Min-Min and BMin, the first step of MinPlus is to compute the time required for all tasks in C to be executed on all virtual machine in VM. For a given task c_i, i=1, 2, …, m, and a given virtual machine vm_j, j = 1, 2, …, n, the completion time T_{ij} of c_i, on vm_j is based on two items. One is how much time (E_{ij}) it takes for vm_j to execute c_i once vm_j is free, the other is at what time (R_j) vm_j is free. Thus, $T_{ij} = E_{ij} + R_j$. Since tasks are constantly assigned to virtual machines, each virtual machine's availability (R_j) changes dynamically. Our algorithm will monitor each machine's availability (R_j) and at the same time dynamically assign tasks to virtual machines.

Our algorithm is executed step by step. In each step, we select a task and assign it to a virtual machine. So, we will need to make two decisions at each step: 1) which task should we select? 2) which virtual machine should we assign this task to?

In each step, for each task c_i in C that has not been assigned, we first check which virtual machine vm_j in VM we should tentatively assign c_i to. We consider the following factors:

A. T_{ij}, the completion time of c_i on vm_j

B. E_{ij}, the execution time of c_i on vm_j

C. R_j, the availability of vm_j

We calculate

$$t_i = min_j \, (T_{ij} * (E_{ij} / R_j + 1) * (E_{ij} / Sum_i (E_{ij}))) \qquad (1)$$

The right side of equation 1 contains three factors. The first factor is the completion time T_{ij}, which shows at what time c_i will be tentatively completed in vm_j. This is the criteria the original Min-Min algorithm adopts to decide which virtual machine should we assign a task to. The second factor is $E_{ij} / R_j + 1$, that shows how dramatically assigning c_i to vm_j will affect the workload of vm_j. If R_j has a large value and E_{ij} has a small value, assigning c_i to vm_j will not change the workload of vm_j much. However, if R_j has a small value and E_{ij} has a large value, assigning c_i to vm_j will greatly change the workload of vm_j, which is something we should avoid. The third factor is $E_{ij} / Sum_i (E_{ij})$, which shows compared to other tasks, how much time it takes vm_j to execute c_i. We favor a virtual machine that yields relatively small execution time of c_i, compared to execute times for other tasks on the same virtual machine.

In equation 1 we consider three factors. It is because we not only should consider the tentative completion time of c_i on vm_j, and how assigning a task affects the workload of a virtual machine, but also should analyze how a virtual machine executes different tasks. If a virtual machine, compared to other virtual machines, takes much less time to execute c_i than executing other tasks, this virtual machine should be possibly considered to receive c_i.

After we calculate t_i for all unassigned tasks in C (in this process, a virtual machine is tentatively selected for each task), we will consider which task c_i will be selected for resource allocation in the current step. We calculate

$$p_i = t_i / (var\ (T_{ij}) + (T_{ij} - T_{i1}) + (T_{ij} - T_{i2})) \quad (2)$$

in which T_{i1} is the completion time for c_i on all virtual machines that is closest to T_{ij}, and T_{i2} is the completion time for c_i on all virtual machines that is second closest to T_{ij}. The reason we should consider these factors is that, we should not only consider the variance of completion time of c_i on all virtual machines (we favor high variance since it means we should assign c_i to the best option virtual machine), but also take into account how far T_{ij} is from its nearest neighbors (we also favor high values of $T_{ij} - T_{i1}$ and $T_{ij} - T_{i2}$ since it means we should assign c_i to the best option virtual machine).

The value p_i calculated in equation 2 is used to determine which unassigned task from C to choose for resource allocation in the current step. When a task is selected and associated to its preferred VM (from equation 1), we will label task as "assigned", and update the availability R_j of that virtual machine. Next, we will update the tentative completion times of all unassigned task on this virtual machine since its availability is updated. At this point, the current step is finished, and we go into the next step, selecting the next unassigned task from C (using equation 2) for resource allocation (using equation 1). We summarize our algorithm in figure 1.

IV. Experimental Results

MinPlus is tested with i7-1185G7 CPU operating at 3.00GHz, 32.0 GB of RAM, and the Java cloud computing simulation framework CloudSim. The algorithms are assessed using CloudSim, which can simulate a wide range of cloud components [4]. It facilitates us to conduct research by reducing the number of variables in the model that are unknown. In addition to the features and add-ons offered in the core package, CloudSim is quite extensible and provides a wide range of new functionality.

Algorithm MinPlus:

Define:

m: #(tasks); n: #(virtual machines in cloud); C: a set of tasks; VM: a set of virtual machines

T_{ij}: array for completion times of tasks in C on virtual machines in VM

E_{ij}: array for execution times of tasks in C on virtual machines in VM

R_j: array for availabilities of virtual machines in VM

For each task c_i in C, i=1, 2, …, m

For each virtual machine vm_j in VM, j =1, 2, …, n, from the cloud

$T_{ij} = E_{ij} + R_j$

While there are unassigned tasks in C

Begin

For each unassigned task c_i in C

Compute $l_{ij} = T_{ij} * (E_{ij} / R_j + 1) * (E_{ij} / Sum_i (E_{ij}))$

Calculate t_i and record the virtual machine that obtains t_i

$t_i = min_j (l_{ij})$

For each unassigned task c_i in C

Calculate $p_i = t_i / (var\ (T_{ij}) + (T_{ij} - T_{i1}) + (T_{ij} - T_{i2}))$

Select c_q with the smallest value of p_q

Send c_q to its preferred virtual machine vm_p associate with t_q

Remove c_q from C, update R_p, and update T_{lp} for all l

End //while

Fig. 1. MinPlus Algorithm

Throughput, turnaround time, and load balancing are a few of the performance traits that are gathered and evaluated throughout the simulation to compare MinPlus to BMin and Min-Min. The time between submitting and receiving an assignment is known as the turnaround time. The virtual machine workload standard deviation is calculated to evaluate load balancing. Throughput means how many jobs are executed within a predetermined period. We design a separate program to create datasets that contains virtual machines with various processing powers, tasks with different sizes, etc. In our experiments, we apply virtual machines numbers as 3, 6, …, all the way to 30, and task (Cloudlet) number between 10 and 100.

A. MinPlus Experiment Results

Figure 2 illustrates how throughput increases as the number of VMs increases. Figure 3 demonstrates how the average turnaround time lowers as the number of VMs rises. Both figures show different cases of cloudlet numbers ranging from 10 to 100.

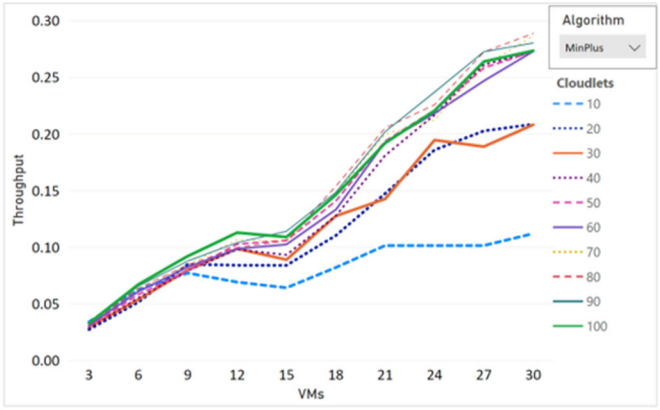

Fig. 2. Scalable performance of MinPlus in terms of throughtput

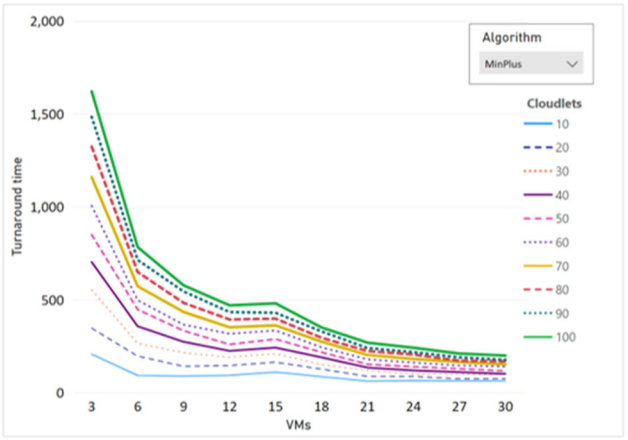

Fig. 3. Scalable performance of MinPlus in terms of turnaround time

We compare the performance of MinPlus with BMin and Min-Min. The examples in Figures 4, 5, 6, 7, 8 and 9 show that MinPlus frequently outperforms BMin and Min-Min in terms of throughput when evaluated on various numbers of cloudlets.

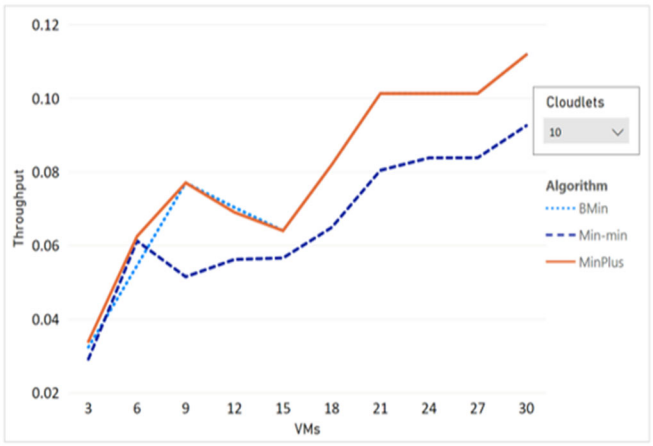

Fig. 4. Running result of BMin, Min-min and MinPlus (throughput, 10 tasks)

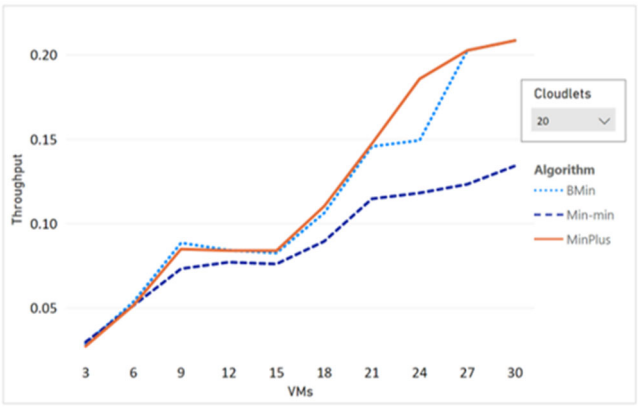

Fig. 5. Running result of BMin, Min-min and MinPlus (throughput, 20 tasks)

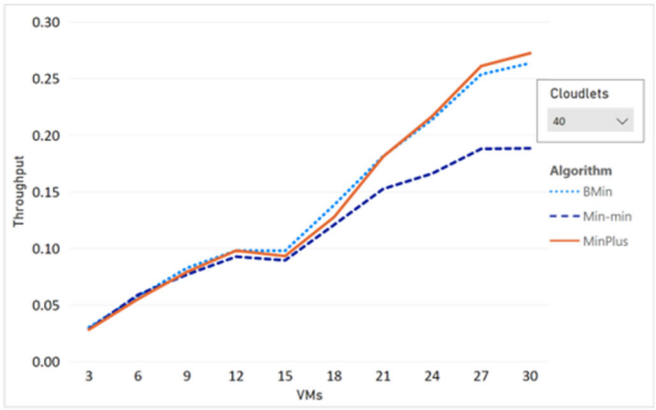

Fig. 6. Running result of BMin, Min-min and MinPlus (throughput, 40 tasks)

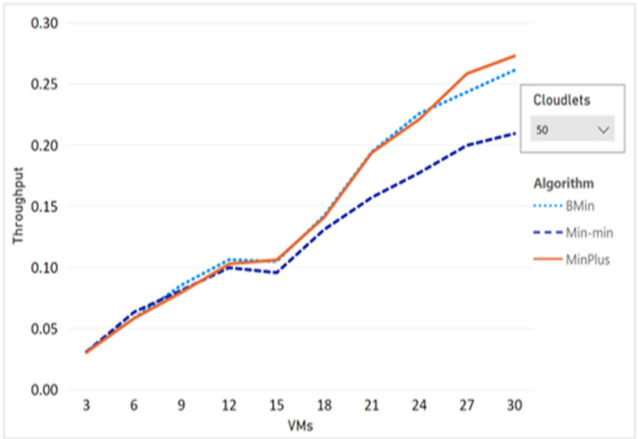

Fig. 7. Running result of BMin, Min-min and MinPlus (throughput, 50 tasks)

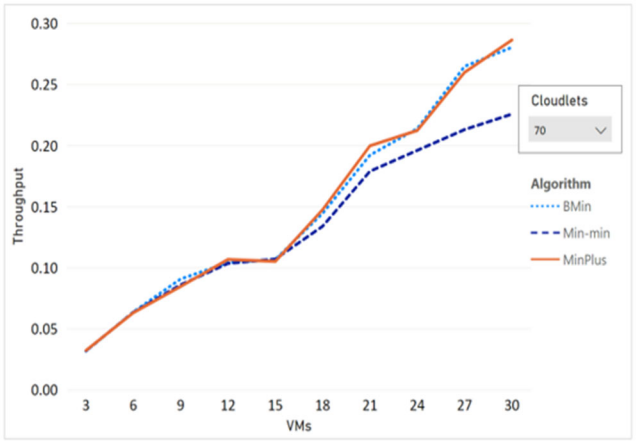

Fig. 8. Running result of BMin, Min-min and MinPlus (throughput, 70 tasks)

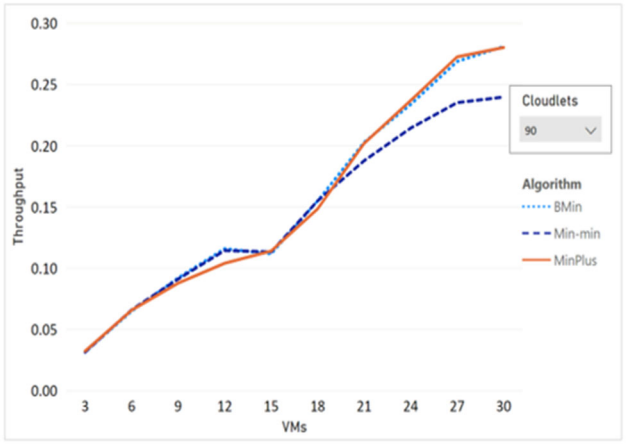

Fig. 9. Running result of BMin, Min-min and MinPlus (throughput, 90 tasks)

Figure 10, 11, and 12 show that MinPlus often times outperforms BMin and Min-Min in terms of turnaround time.

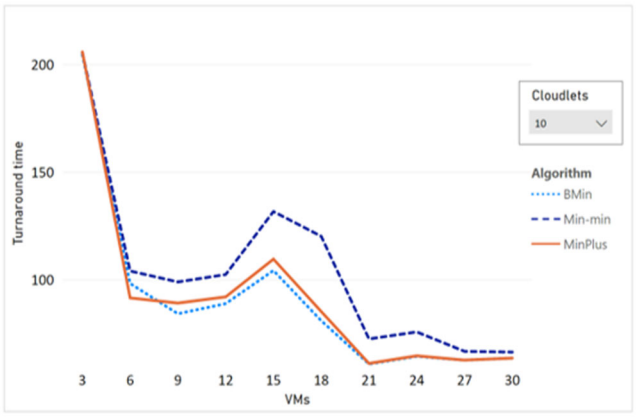

Fig. 10. Running result of BMin, Min-min and MinPlus (turnaround time, 10 tasks)

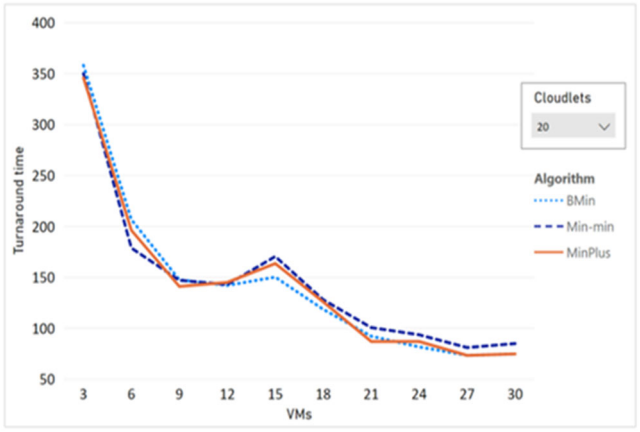

Fig. 11. Running result of BMin, Min-min and MinPlus (turnaround time, 20 tasks)

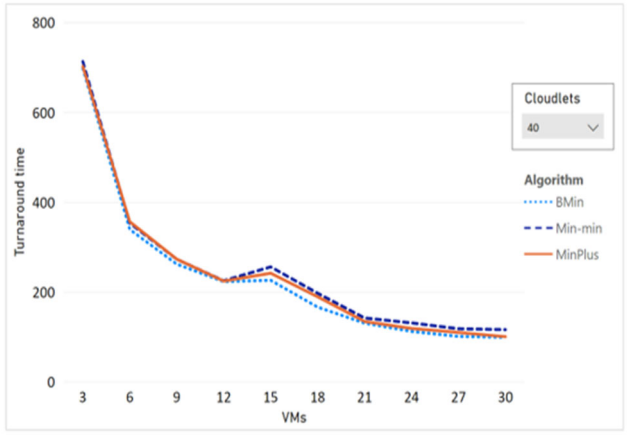

Fig. 12. Running result of BMin, Min-min and MinPlus (turnaround time, 40 tasks)

We also compare MinPlus with BMin and Min-Min in terms of virtual machine workload standard deviation. Figure 13, 14, 15, and 16 show that a lot of cases, MinPlus outperform BMin and Min-Min in this category as well.

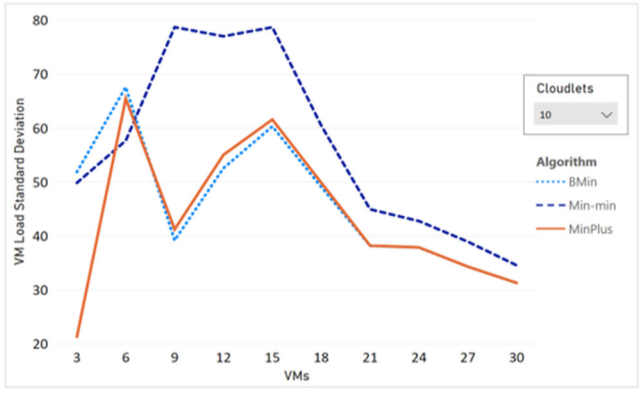

Fig. 13. Running result of BMin, Min-min and MinPlus (VM load standard deviation, 10 tasks)

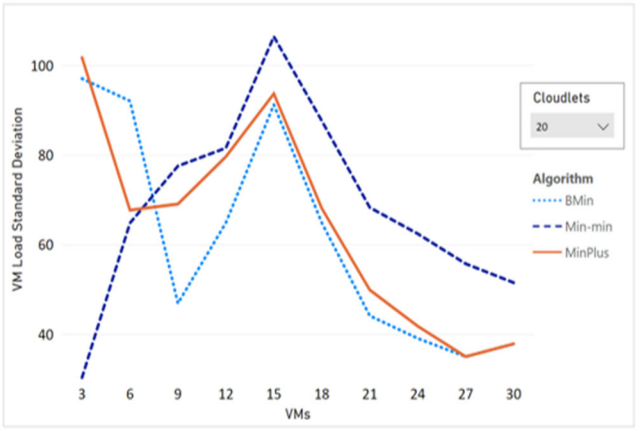

Fig. 14. Running result of BMin, Min-min and MinPlus (VM load standard deviation, 20 tasks)

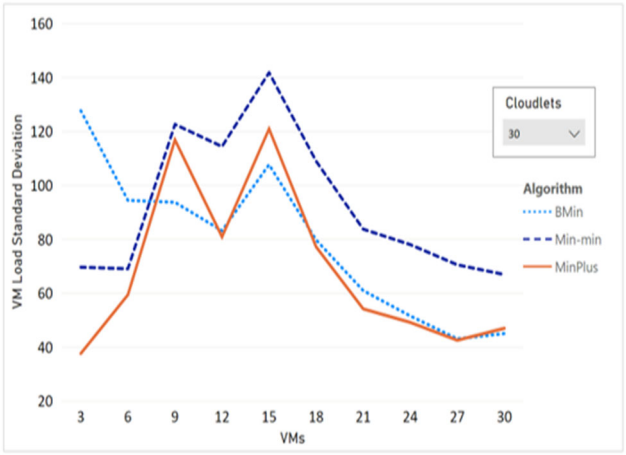

Fig. 15. Running result of BMin, Min-min and MinPlus (VM load standard deviation, 30 tasks)

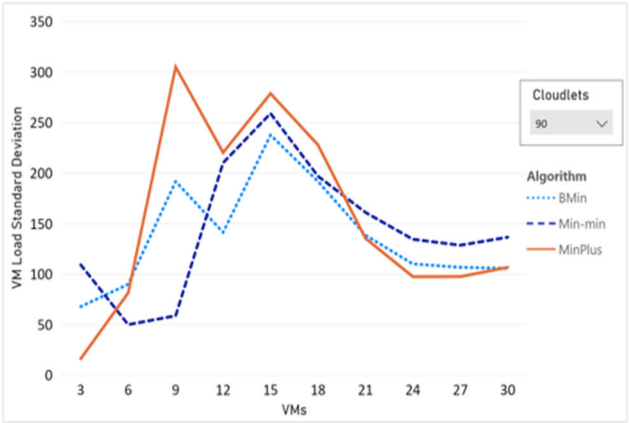

Fig. 16. Running result of BMin, Min-min and MinPlus (VM load standard deviation, 90 tasks)

V. Conclusion

We propose MinPlus, a new task scheduling algorithm designed to gain better throughput, load balancing, and turnaround time. Experiments that use the CloudSim framework and compare MinPlus' performance to that of its predecessors demonstrate the algorithm's advantages. In the future, we will continue revising our algorithm to achieve better performance for resource allocation.

ACKNOWLEDGEMENT

This research was supported in part by US NSF grants CNS-2103405, AMPS-2229073, CPS-2103459, and SHF-2210744.

References

[1] P. Salot, “A survey of various scheduling algorithm in cloud computing environment”, M.E, computer engineering, India.

[2] R. M. Singh, S. Paul and A. Kumar, “Task scheduling in cloud computing”, International Journal of Computer Science and Information Technologies, Vol. 5 (6), 2014.

[3] R. P. Padhy, P. G. P. Rao, “Load Balancing in Cloud Computing Systems”, National Institute of Technology, Rourkela, May,2011.

[4] The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne. http://www.cloudbus.org/intro.html

[5] E. J. Ghomi, A. M. Rahmani, and & N. N. Qader, (2017). Load-balancing algorithms in cloud computing: a survey. Journal of Network and Computer Applications, 88, 50-71.

[6] Q. Xu, R. V. Arumugam, K. L. Yong, Y. Wen, Y. S. Ong, and W. Xi (2015). Adaptive and scalable load balancing for metadata server cluster in cloud-scale file systems. Frontiers of Computer Science, 9(6), 904-918.

[7] J. Bushra, I. Humaira, S. Mohammad, M. Kashif, and B. Rajkumar, "Resource Allocation and Task Scheduling in Fog Computing and Internet of Everything Environments: A Taxonomy, Review, and Future Directions," ACM Comput. Surv. 54, 11s, Article 233 (January 2022), 38 pages. https://doi.org/10.1145/3513002

[8] MH. Kim, JY. Lee, S.A. Raza Shah *et al.* "Min-max exclusive virtual machine placement in cloud computing for scientific data environment," *J Cloud Comp* **10**, 2 (2021). https://doi.org/10.1186/s13677-020-00221-7

[9] V. Gajera, Shubham, R. Gupta and P. K. Jana, "An effective Multi-Objective task scheduling algorithm using Min-Max normalization in cloud computing," 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), 2016, pp. 812-816, doi: 10.1109/ICATCCT.2016.7912111.

[10] E. K. Tabak, B. B. Cambazoglu and C. Aykanat, "Improving the Performance of Independent Task Assignment Heuristics MinMin,MaxMin and Sufferage," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 5, pp. 1244-1256, May 2014. doi: 10.1109/TPDS.2013.107

[11] M. Maheswaran, S. Ali, H. J. Siegal, D. Hensgen and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99), San Juan, Puerto Rico, USA, 1999, pp. 30-44. doi: 10.1109/HCW.1999.765094

[12] Y. Shi, K. Suo, S. Kemp and J. Hodge, "A Task Scheduling Approach for Cloud Resource Management," 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), 2020, pp. 131-136, doi: 10.1109/WorldS450073.2020.9