Response-Time Tail Mitigation in Cloud Systems

1st Asaad Althoubi
Department of Computer Science
Kent State University
Kent, OH USA
aalthoub@kent.edu

2nd Hassan Peyravi
Department of Computer Science
Kent State University
Kent, OH USA
hpeyravi@kent.edu

***Abstract*—Response time is one of the most important metrics in cloud data centers that measure application performance and availability. The response time depends on how the scheduler distributes tasks belonging to a job to various cores. Even with at a maximum level of parallelism, the slowest task (the straggler) dominates the response time, thereby increasing the average and extending the tail of the response time distribution.**

In this paper, we propose a distributed scheduling scheme where tasks are systematically assigned to randomly selected servers based on the expected response time of each server and the size of the tasks. Consequently, straggling tasks are less likely to become head-of-line blockers and are more likely to finish in a shorter period of time. In light of the bursty nature of task arrivals, the instantaneous response time is not an accurate measure of a server's load. Therefore, we developed a Truncated Exponentially Weighted Moving Average (*TEWMA*) for the selection of servers. To adapt to the new schemes, we augmented the Sparrow simulation and compared its performance to Sparrow's. Our approach probes fewer servers and selects most suitable ones based on a simple matching algorithm. The results have shown that the tail of response time distribution has been declined considerably, and consequently the average response time has been reduced accordingly.

***Keywords*—Cloud, Job Scheduling, Response Time, Tail Latency, TEWMA**

I. Introduction

Cloud computing is a seamless on-demand computing resource allocation platform that provides application services without direct active management by the user. Cloud applications and their business driven configurability often need different quality-of-service requirements.

Recent advancements in cloud computing have helped seamless integration of resources on-demand and the quick arrangement of inter-connected data centers that are geographically dispersed for offering high quality and dependable services [1]. Cloud computing has risen as another Internet-based model for empowering clients. It can organize access to a shared pool of configurable assets on-demand, which can be immediately deployed and discharged with very little administration or cloud provider cooperation [2].

The various environments where cloud computing is applicable, such as finance, medicine, industry, etc., made the the cloud computing resources and services having different features to meet the different user requirements. Thus, it is necessary to study cloud computing scheduling algorithms because cloud computing represents further development of parallel computing, distributed computation and grid computing [3]. Scheduling algorithms are generally categorized as static and dynamic. In static scheduling algorithms, all the required information such as the number of jobs, the number of resources, the processing times, etc must be known in advance. In contrast, in dynamic scheduling, resources are allocated as and when users require.

For cloud computing, task scheduling technology is one of the core technologies of cloud platform. There are some deficiencies in existing task scheduling algorithms, which restrict the application of cloud computing technology in the field of application. Thus, an excellent task scheduling algorithm should not only effectively reduce the total task completion time, but also better realize the load balancing of the system and improve the utilization of cloud computing resources [4].

Some of the current parallel cloud applications show a decrease in CPU utilization whenever there is an increase in parallelism [5]. The scheduling mechanism plays a crucial role in cloud computing performance. If arriving jobs are not scheduled in an efficient way, then the performance reduces as the cloud processes a huge amount of data. Moreover, any increasing in the number of clients using the cloud makes the scheduling more difficult and then an efficient scheduling algorithm needs to be utilized.

The users of cloud may utilize hundreds or thousands of virtualized resources and users cannot allocate each task manually. Thus, scheduling plays a vital role in cloud computing to assign the resources efficiently and effectively.

It can be difficult for service providers to maintain small tails of latency distribution, especially for large-scale interactive systems. Consistency in service delivery becomes more challenging as system size and complexity increase. When a request is handled in parallel, the long tail distribution of the parallel operation may become problematic right away and take over the total response time. Each response must be consistent and latency-free, or the operation's response time will increase.

Furthermore, load balancing, scalability, response time, reliability, dynamic re-allocation of resources to the computing machines are all the major conspicuous problems in task scheduling. Hence, an efficient task scheduling algorithm is needed in the cloud computing environment. Thus, the objective of this paper is to propose a new task scheduling

979-8-3503-2028-2/22/$31.00 ©2022 IEEE
DOI 10.1109/CSCI58124.2022.00290

algorithm for low latency applications in cloud computing environments.

The proposed algorithm in this paper focuses on providing a fine-grained task scheduling for low-latency applications. This scheduling approach assumes that there is already a long-running executor process running on each server, and it uses a modified truncated waiting moving average (*TEWMA*) as an indicator to the expected server response time. The mechanism of our proposed approach is explained in more details in Section III.

The rest of this paper is organized as follows. Section II shows a literature on the recent researches and contributions in task scheduling in cloud environments. Section III presents the design goals of developing task scheduling approach and illustrates how the proposed task scheduling approach handle job parallelism. Section IV shows the simulation results and the parameters used for the cloud computing environment. Section V provides a new method to efficiently compute the expected waiting moving average (*TEWMA*). Section VI provides a conclusion.

II. Related Work

In cloud computing, scheduling has been recently capturing attention of many researchers. Many scheduling algorithms and approaches have been developed, entailing concepts from the service oriented computing and aspects from the constantly changing distributed systems environments.

In [6], a scheduling algorithm for the cloud computing system based on the driver of dynamic essential path to solve the problem of the scheduling result affected by the scheduling order change of each task node in the scheduling process. In [7], a job scheduling algorithm for edge computing based on Modified Monte Carlo Tree Search (*MMCTS*) proposed to guide the jobs to the most suitable edge server to improve resources utilization. To minimize the overhead and the number of fail jobs which overstep their deadline in edge cloud, [8] proposed a parallel job scheduling algorithm to solve the scheduling problem based in edge cloud on the process mode that job scheduling on parallel batch processing machines (P-BPM), called P-MACO algorithm (Parallel-batch multi-object ant colony algorithm).

The heterogeneity of resources and the various execution periodicity of tasks in jobs significantly impact the efficiency of large scale job executions. Peng et al. [9] proposed a periodic task-oriented two-level job scheduling architecture which provides resource domains for the scheduler to shield the details of the underlying heterogeneous resources. Moreover, all tasks with the same resource requirements are assigned to the same sub-job scheduler to reduce the scheduling complexity.

To ensure service quality when improving the resource utilization rate, Cheng et al. [10] developed an online framework for task scheduling in large-scale warehouse data centers. Zhao et al. [11] combined q-learning with deep neural networks and used the maximum completion time and load balance as performance indicators to solve the task scheduling problem in the cloud environment. In [12], the whale optimization algorithm was used for cloud task scheduling with multiple objectives.

To reduce the task completion time and cost, Randa et al. [13] developed a heuristic scheduling algorithm that is based on perceptual mobility, and [14] proposed a Multi-objective Evolutionary Algorithm on the Cloud (MEAC), which provides some novel schemes including problem-specific encoding and also evolutionary operations, such as crossover and mutation. This algorithm shows a significant reduction in cost and time of task scheduling.

This paper explores a different scenario in the design space as it proposes a set of decentralized machines that operate autonomously. This decentralized scheduling system provides better scaling and availability properties. For example, users can send requests to an alternate scheduler if a scheduler fails. Moreover, given the fact that providing response times in decentralized systems comparable to those provided by centralized systems is challenging, this paper adopts the power of using the exponential moving weighted average ($EWMA$) concept in scheduling user requests to provide better decisions when placing jobs (better load balancing) on the servers with the minimum average waiting times.

III. Scheduling Algorithm For Parallel Jobs

There has been various types of scheduling algorithm exist in distributed computing system. For the best system throughput, a job scheduling algorithm should achieve a high performance computing. Traditional job scheduling algorithms may not be able to provide the same scheduling in the cloud environments.

Since batch workloads acquire resources for long periods of time and thus require infrequent task scheduling, low-latency workloads have more demanding scheduling requirements than batch workloads do. The goal of scheduling algorithm is to minimize the job completion time. The start time and finish time of each task within a job are essential and significantly influence the completion time, which basically entails the computation and communication time.

A task scheduler usually have a complete view of which tasks are running and on which servers. Then, it uses this view to schedule the arriving tasks to available servers. Sparrow [15] came up with a different approach in which many schedulers operate in parallel and do not maintain any state about cluster load. In Sparrow, schedulers rely on instantaneous load information acquired from servers. In this paper, a new approach is proposed to improve performance, introduces the idea of calculating the workload at the servers in terms of the expected waiting time of tasks arrived at servers. In this approach, the placement of jobs is done through assigning the tasks with the highest exponentially estimated processing time to the servers with the lowest expected average waiting time, so the problem of task starvation is minimized.

A. Simulation Model

In our simulation, a cluster composed of servers that execute tasks and schedulers that assign tasks to server is used. A job

consists of k tasks that are each allocated to a server and that job can be handled by any scheduler. If a server currently doesn't have enough resources for the new arriving tasks to be executed, it queues those new arriving tasks until existing tasks release enough resources.

Additionally, the time from when a task is submitted to the scheduler until when the task begins executing is called *wait time*. The time the task spends executing on a server is called *service time*, and the time the job spends from when the job is submitted to the scheduler until the last task finishes executing is called *job response time*. The time difference between the job response time using a given scheduling technique and the job response time if all of the job's tasks are scheduled with zero wait time is called *delay* (the total delay within a job is due to both scheduling and queuing).

B. Job Scheduling and Task Placement

The mechanism of the proposed scheduling technique is to aggregate load information in terms of expected average waiting time for tasks within server queues rather than queue lengths.

For each arriving job consists of several tasks with different durations, a task must be placed in a server queue with the lowest expected queuing delay. This expected delay is estimated using the concept of *late binding*, which was proposed by Sparrow [15]. In *late binding*, the scheduler sends a reservation request to a server. Once the request reaches the front of the server queue, the server sends a *RPC* back to the scheduler indicating the queuing delay. However, *late binding* unnecessarily increases the delay for the scheduler. The delay exasperates with the server request getting behind a head-of-line blocking task (or long task). Consequently, this will negatively impact the performance of the scheduler in terms of its throughput.

The control chart *EWAM* has been widely used to smooth data variations and reduce system fluctuation or oscillation. Given the most recent *RPC*'s are more representative of a server response time, we have used the truncated EWAM (*TEWMA*) with a limited sliding window of *RPC* samples. Equation (1) describes how an exponentially weighted moving average is computed, where $\overline{W}_t$ is the moving average at time t and α is a smoothing factor $0 < \alpha \leq 1$.

The mechanism of our proposed algorithm adopts two important aspects. First, schedulers send probes to servers, and the probed servers instantaneously responds back to the schedulers with their expected average waiting times for each sent probe. Let W_t be the sample waiting time observed from a turnaround time of a job being executed. Then, the moving average waiting time ($EWMA$) is computed as

$$\overline{W}_t = \begin{cases} W_0, & \text{if } t = 0 \\ \alpha W_t + (1-\alpha)\overline{W}_{t-1}, & \text{if } t > 0 \end{cases} \quad (1)$$

Second, a scheduler assigns job tasks to the set of servers in such away that the task with the highest duration time is placed on the server with the least expected average waiting time, the second task with the second highest duration time is placed on the server with the second least average expected waiting time and so on. This new scheduling approach is shown in Algorithm 1.

Algorithm 1 A Job Scheduling Algorithm for Cloud

1: $M = \{m_1, m_2, \cdots, m_n\}$: Set of servers
2: p: probe factor, $p \geq 1$
3: $\overline{W}(m_i)$: represents the mean waiting time of server m_i
4: $\overline{E}(t_j)$: represents the expected finish time of task t_j
5: **for** each arriving job J at the scheduler **do**
6: $P = k \times p$ Probe size
7: $S = \{m'_1, m'_2, \cdots, m'_P\}$: Set of P probed servers, such that $\overline{W}(m'_i) \leq \overline{W}(m'_j)$ and $m'_i, m'_j \in M, i \leq j$
8: $T = \{m'_1, m'_2, \cdots, m'_k\}$: Set of the first k servers in S
9: $J' = \{t'_1, t'_2, \cdots, t'_k\}$: Set of k tasks in job J, such that $\overline{E}(t'_i) \leq \overline{E}(t'_j), i \leq j$
10: **for** $t'_j \in J'$ **do**
11: $m'_j \leftarrow t'_j$ $j = 1, 2, \cdots, k$, assign task t'_j to server m'_j
12: $T = T \setminus m'_j$
13: **end for**
14: **end for**

IV. Experiments and Results

Experiments are executed and explained in this section to verify the effectiveness of the proposed Algorithm 1. The parameters of our algorithm are shown in Table 1.

TABLE I
Table of Notations

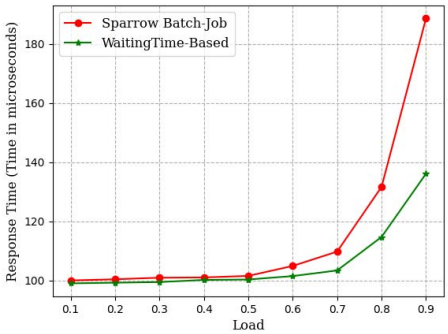
Parameter	
M	servers
p	probe factor
$\overline{W}$	average waiting time
$\overline{E}$	expected finish time
J	job
P	probe size
t	task

Our proposed algorithm is implemented using Python 3.8. We compare our algorithm with another scheduling algorithm, Sparrow, and within the same environment and under the same conditions. Sparrow is a stateless decentralized scheduler that provides near optimal performance using two key techniques, batch sampling and late binding. All experiments were conducted on a real VM-based cloud platform consisting of 10000 machines.

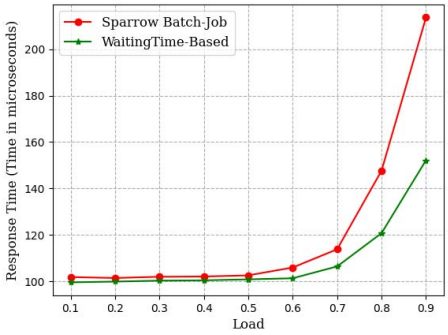
(a)

(b)

Fig. 1. (a) Shows the performance when using 1000 machines of 4-core and probe factor of 2 (b) Shows the performance when using 1000 machines of 4-core and probe factor of 3



(a)



(b)

Fig. 2. (a) Shows the performance when using 1000 machines of 4-core and α value of 0.2 (b) Shows the performance when using 1000 machines of 4-core and α value of 0.3

A. Response Time Performance

In Figure 1, it can be seen that changing the probe factor impact the performance, especially at higher cluster loads. Figure 1 depicts the response time for both Sparrow Batch-Job and our proposed approach. Figure 1 shows that our proposed approach provides a better performance (less response time) when probe factor increases from 2 to 3, especially when load goes above 70% in a 1000-machine cluster of 4-core machines running the synthetic workload.

The probe factor is one of the major factors in our proposed approach as changing its value impacts the performance. A small probe factor negatively impacts the performance because schedulers do not sample enough machines to find least light loaded machines. *However, over-sampling may also negatively impact the performance due to the increasing in the communication messaging*. Overall, using machines with higher cores and a better probe factor chosen carefully/properly improves the performance as shown in Figure 3. The effect is most apparent, especially when load goes above 70%.

Another important factor in our scheduling approach is α. Changing the α value in computing the expected average waiting time formula plays a crucial role and heavily impacts the performance. Figure 2 depicts the response time for both Sparrow Batch-Job and our proposed approach when changing the α value.

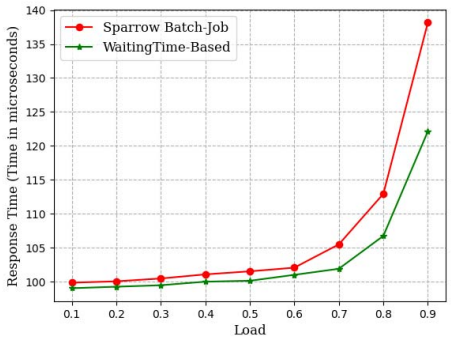
To further see how our proposed approach performs when increasing size of cluster, we considered a 10000-machine cluster of 6-core and 8-core machines. The proposed approach still gives a better performance when increasing probe factor from 2 to 3, alpha value from 0.2 to 0.3, and average job size from $\cong$ 50 to $\cong$ 150 in comparison to Sparrow as shown in Figure 3. Overall, our experiment tests show that for small jobs of less than 100 tasks, the α value should be small too, 0.2 in our case. For large jobs, jobs of more than 100 tasks, α value of 0.3 gives the best performance.

B. Performance Evaluation

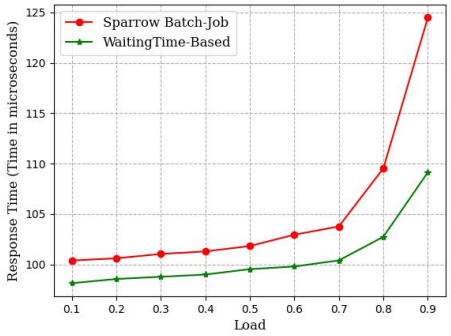
As shown in Figure 4, response time exponentially increases with increasing load, because schedulers have less success finding free machines on which to schedule tasks. In this paper, the percentage of performance gain is computed as

$$\textit{Performance Gain (\%)} = 1 - \frac{\overline{T}_W}{\overline{T}_S},$$

where $\overline{T}_S$ and $\overline{T}_W$ represent the average response times in Sparrow and our approach respectively.



(a)



(b)

Fig. 3. (a) Shows the performance when using 10000 machines of 6-core and probe factor of 2, α of 0.2 and job size of 50 tasks (b) Shows the performance when using 10000 machines of 6-core and probe factor of 3, α of 0.3 and job size of 150 tasks

At 70% load, WaitingTime-Based sampling improves performance by 14% compared to Sparrow Batch-Job placement, and this performance improvement increases from 19% to 24% when load increases from 80% to 90%.

It is obvious that the curves of our proposed approach and Sparrow are largely offset when load goes above 70% as shown in Figure 1 and 2. Figure 3 shows a remarkable insight that the performance gain is increased with the increasing in number of cores on machine servers and the increasing in the carefully-chosen probe factor and alpha. In a cluster of 10000 of 8-core machines and a probe factor of 3, the performance gain of our proposed approach nearly reaches 15% when load is 70%, and it increases to 24% when loads is 90%. The performance is slightly decreased when reducing number of cores to only 6 on the servers so the performance nearly reaches 10% when load is 70%, and it increases to 20% when loads is 90%. Overall, the performance is linearly reduced with the decreasing in the cluster size, probe factor, alpha and number of cores.

Based on the experimental results, the following suggestions seems considerable

- For heavy load (above 70%) and small number of virtual machines available, the proposed approach gives a better performance
- For a load (above 70%), a large number of virtual machines available and a small probe factor, the proposed approach gives a better performance
- For virtual machines with higher cores, the proposed approach gives a better performance, especially when load goes above 70%

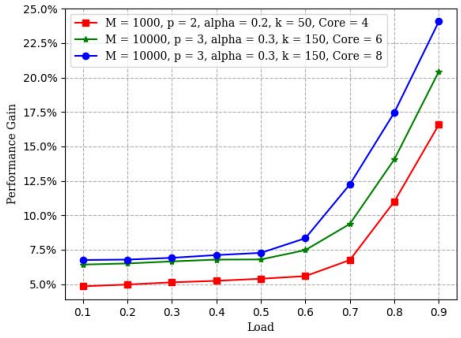


Fig. 4. Shows the performance gain percentage in the proposed approach when changing the values of the five key parameters, M, p, α, k, and $Cores$

V. Expected Server Response time

As a result of the time-invariant of server response times, the arithmetic mean of consecutive response times is not an ideal indicator of the average load on a server. Thus, the recent results (server response times) should be regarded as reliable to consider the required accuracy of the coming results. This is sometimes called a 'history window' or 'spin-up interval'. The effect of prior server response times on the resultant moving average depends on α; smaller α values make the choice of $\bar{T'}_w$ relatively more important than larger α values, since a higher α discounts older observations faster.

Therefor, we consider a modified Exponentially Weighted Moving Average (*TEWMA*), which is a sliding-window-based [1]. In a typical *TEWMA*, a weight α is given to the most recent response time and $\alpha(1-\alpha)^j$ is given to the previous w response times, respectively, where $j = 1, 2 \cdots$.

However, given the bursty nature of the traffic in data centers, the average response time depends only on a few recent w terms. Therefore, a relationship between w and α is required. To establish such a relationship, we first need to expand the recursive relationship in *EWMA* as:

[1] See https://ieeexplore.ieee.org/abstract/document/4341677 for more details about the truncated weighted moving average

$$
\begin{aligned}
\bar{T}_i &= \alpha T_i + (1-\alpha)\bar{T}_{i-1} \\
&= \alpha T_i + (1-\alpha)[\alpha T_{i-1} + (1-\alpha)\bar{T}_{i-2}] \\
&= \alpha T_i + (1-\alpha)[\alpha T_{i-1} + (1-\alpha)[\alpha T_{i-2} + (1-\alpha)\bar{T}_{i-3}]] \\
&\vdots \\
&= \alpha[T_i + (1-\alpha)T_{i-1} + (1-\alpha)^2 T_{i-2} + (1-\alpha)^3 T_{i-3} \\
&\quad + \cdots (1-\alpha)^k T_{i-w)}] + 1 - \alpha)^{w+1} T_{i-(w+1)} + \cdots \\
&= \frac{T_1 + (1-\alpha)T_2 + (1-\alpha)^2 T_3 + (1-\alpha)^3 T_4 + \cdots}{1 + (1-\alpha) + (1-\alpha)^2 + (1-\alpha)^3 + \cdots}
\end{aligned}
$$

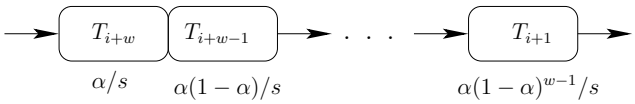
Since $1/\alpha = 1 + (1-\alpha) + (1-\alpha)^2 + (1-\alpha)^3 + \cdots$ and to determine how far to go back for an initial weight value, the weight is omitted by stopping after w terms:

$$
\begin{aligned}
\bar{T'}_w &= \alpha[(1-\alpha)^w + (1-\alpha)^{w+1} + (1-\alpha)^{w+2} + \cdots] \\
&= \alpha((1-\alpha)^w[1(1-\alpha)^2 + (1-\alpha)^3 + \cdots] = (1-\alpha)^w
\end{aligned}
$$

$$
w = \left\lfloor \frac{\log(0.001)}{\log(1-\alpha)} \right\rfloor \tag{2}
$$

A. Implementation

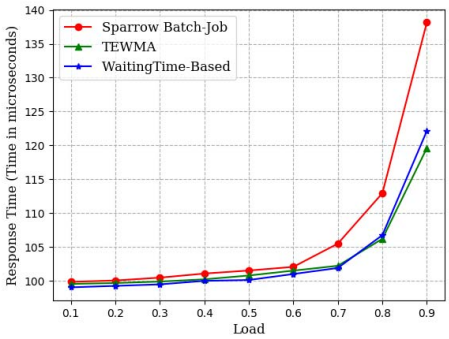
Once a task of job is completed by a server, its turnaround time, which is composed of the server response time plus the network delay, is recorded in the lth most recent position of a sliding window of size w first and then shift the other $w-1$ response times to the right. The sliding window acts as an truncated exponentially weighted sliding window with weights distributed as below. In our simulation, the network is set to 0.5 milliseconds.



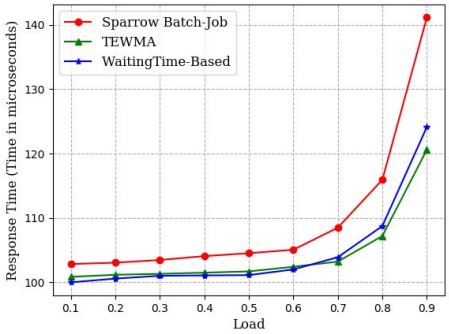
where $s = \alpha \sum_{i=1}^{w-1}(1-\alpha)^i$.

Job tasks are scheduled on the first k servers in S with the minimum expected average waiting times. The expected average waiting time for each of the probed servers in S is computed as the average of the recent w waiting time observations. This gives a better scheduling decision due to the fact that the concurrently operating schedulers may make conflicting scheduling decisions when considering instantaneous response times.

As shown in Figure 5, using the 'history sliding window' and considering w terms related to α improves the performance by 10% when heavy loads (above 70%) . To verify our



(a)



(b)

Fig. 5. (a) Shows the performance when using 10000 machines of 6-core and probe factor of 2, α of 0.2 and job size of less than 100 tasks (b) Shows the performance when using 10000 machines of 6-core and probe factor of 3, α of 0.2 and job size of more that 100 tasks

results, we considered the best α value, which is 0.2, based on the results obtained in our previous experiments.

Our proposed approach shows performance improvements through mitigating the the tail latency by 15% as shown in Figure 6. It shows that 99% of jobs takes less than 400 ms, while 99% of jobs in Sparrow takes 500 ms or less. Our approach also gives better average response times by about 12%. We ran our simulation for 10000 time units.

VI. Conclusion

In this paper, a stochastic distributed scheduling scheme was proposed to minimize the average response times so as to mitigate the tail latency. The proposed approach uses a truncated exponential waiting moving average (*TEWMA*) in computing the expected server response time on the probed machine workers due to the fact that queue length is not an ideal indicator when it comes to load balancing and execution times. The proposed approach uses a sliding window of recent response time observations in making decisions when task placements. Tasks are systematically scheduled on randomly chosen servers based on the expected server response times. It also provides the ability to schedule the least duration

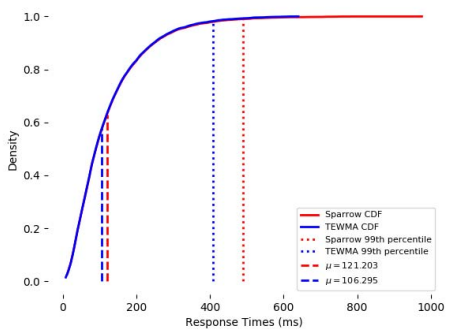


Fig. 6. Shows the tail latency comparison between the proposed approach (*TEWMA*) and Sparrow using 10000 machines of 6-core, probe factor of 2, α of 0.2 and job size of 100 tasks.

time tasks on the highest expected waiting time servers so as to avoid the problem of task starvation and to improve the load balancing. Our approach showed a better performance when high loads. Experiments using a synthetic workload demonstrate that the proposed approach is resilient to different values of probe factor and provides a quite better performance compared to the sampling approach, Sparrow.

REFERENCES

[1] K. Wu, P. Lu, and Z. Zhu, "Distributed online scheduling and routing of multicast-oriented tasks for profit-driven cloud computing," *IEEE Communications Letters*, vol. 20, no. 4, pp. 684 – 687, Feb. 2016.

[2] B. Keshanchi, A. Souri, and N. J. Navimipour, "An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing," *Journal of Systems and Software*, vol. 124, pp. 1–21, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121216301066

[3] B. P. Rimal and E. Choi, "A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing," *Int. J. Commun. Syst.*, vol. 25, no. 6, p. 796–819, jun 2012. [Online]. Available: https://doi.org/10.1002/dac.1279

[4] Y. Yu and Y. Su, "Cloud task scheduling algorithm based on three queues and dynamic priority," in *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, 2019, pp. 278–282.

[5] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Gener. Comput. Syst.*, vol. 91, no. C, p. 407–415, feb 2019. [Online]. Available: https://doi.org/10.1016/j.future.2018.09.014

[6] Z. Xie, X. Shao, and Y. Xin, "A scheduling algorithm for cloud computing system based on the driver of dynamic essential path," *PLOS ONE*, vol. 11, p. e0159932, 08 2016.

[7] H. Wang, C. Li, T. H. Wang, and F. Huang, "A job scheduling algorithm for edge computing based on modified monte carlo tree search," *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pp. 5–9, 2021.

[8] X. Zhao, X. Guo, Y. Zhang, and W. Li, "A parallel-batch multi-objective job scheduling algorithm in edge computing," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 510–516.

[9] P. Zhang, Y. Li, H. Lin, J. Wang, and C. Zhang, "A periodic task-oriented scheduling architecture in cloud computing," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 2018, pp. 788–794.

[10] M. Cheng, J. Li, P. Bogdan, and S. Nazarian, "H_2O-Cloud: a resource and quality of service-aware task scheduling framework for warehouse-scale data centers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 2925–2937, 2020.

[11] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep q-learning," *Inf. Sci.*, vol. 512, no. C, p. 1170–1191, feb 2020. [Online]. Available: https://doi.org/10.1016/j.ins.2019.10.035

[12] J. Xuan, L. Chengyang, and X. Jiang, "Research on multi-objective fuzzy flexible job-shop scheduling based on cloud computinger," in *2020 IEEE 8th International Conference on Computer Science and Network Technology (ICCSNT)*, 2020, pp. 7–10.

[13] R. M. Abdelmoneem, A. Benslimane, and E. Shaaban, "Mobility-aware task scheduling in cloud-fog iot-based healthcare architectures," *Comput. Networks*, vol. 179, p. 107348, 2020.

[14] X. Zhou, G. Zhang, T. Wang, M. Zhang, X. Wang, and W. Zhang, "Makespan–cost–reliability-optimized workflow scheduling using evolutionary techniques in clouds," *Journal of Circuits, Systems and Computers*, vol. 29, no. 10, p. 2050167, 2020. [Online]. Available: https://doi.org/10.1142/S0218126620501674

[15] K. Ousterhout, P. Wendell, M. A. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.