IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 5, 2020, accepted June 10, 2020, date of publication June 15, 2020, date of current version June 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002565

LSTPD: Least Slack Time-Based Preemptive Deadline Constraint Scheduler for Hadoop Clusters

IHSAN ULLAH[1], MUHAMMAD SAJJAD KHAN[2,3], MUHAMMAD AMIR[3], (Member, IEEE), JUNSU KIM[2], AND SU MIN KIM[2], (Member, IEEE)

[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea
[2]Department of Electronics Engineering, Korea Polytechnic University, Siheung 13557, South Korea
[3]Department of Electrical Engineering, International Islamic University at Islamabad, Islamabad 44000, Pakistan

Corresponding author: Su Min Kim (suminkim@kpu.ac.kr)

This work was supported in part by the Ministry of Science and ICT (MIST), South Korea, through the Information Technology Research Center (ITRC) Support Program supervised by the Institute for Information and Communication Technology Planning and Evaluation (IITP), under Grant IITP-2019-2018-0-01426, and in part by the National Research Foundation (NRF) funded by the Korea Government (MIST) under Grant 2019R1F1A1059125.

ABSTRACT Big data refers to numerous forms of complex and large datasets which need distinctive computational platforms in order to be analyzed. Hadoop is one of the popular frameworks for analytics of big data. In Hadoop, a big job is split into multiple small tasks and then they are distributed to worker nodes in a parallel way using MapReduce to speed up computational processes. In this aspect, it is important how to improve throughput performance. MapReduce jobs require quick responses from the worker nodes to complete them under their deadlines. The existing scheduling schemes for Hadoop such as FIFO, fair, and capacity schedulers cannot guarantee the quick response requirement satisfying a prior deadline. Thus, Hadoop system needs to improve response time and completion time for the heterogeneous MapReduce jobs. In this paper, we propose an efficient preemptive deadline constraint scheduler based on least slack time and data locality. In order for better allocation of tasks and load balancing, we first analyze the task scheduling behaviors of the Hadoop platform. Based on that, we propose a novel preemptive approach which considers the remaining execution time of the job being executed in deciding preemption. The experimental results show that the proposed scheme significantly reduces the job execution time and queue waiting time, compared to existing schemes.

INDEX TERMS Hadoop, MapReduce, distributed system, parallel computing, preemptive job scheduling, queuing theory.

I. INTRODUCTION

In recent years, cloud computing and big data have attracted the researchers' attention. It comprises distributed computing over a network where the applications are run in parallel and distributed manners [1]. Cloud service providers such as Amazon, Google, and Yahoo. developed their data centers in a distributed manner so that users can access anyplace. The main goal of distributed computing is to develop a distributed network of heterogeneous devices that work together and share their workload. Big data processing is one of the prominent research topics for future internet [2]. Google introduced an efficient programming model, called MapReduce, for the processing of large datasets [3]. It is regarded as the first massive model, in distributed computing and cloud computing where the data is processed in a parallel way. Nowadays, Hadoop MapReduce is an open-source platform, which widely employed for the development of a distributed and cloud computing system.

Hadoop is a distributed computing framework based on the MapReduce model that runs applications on a cluster of a large number of commodities and inexpensive computing nodes. It is developed by Google in 2004 to handle big data applications by parallel processing. In Hadoop system, all the jobs share the heterogeneous resources of clusters [4]–[6]. Hadoop requires proper scheduling policy and algorithm

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Asaduzzaman.

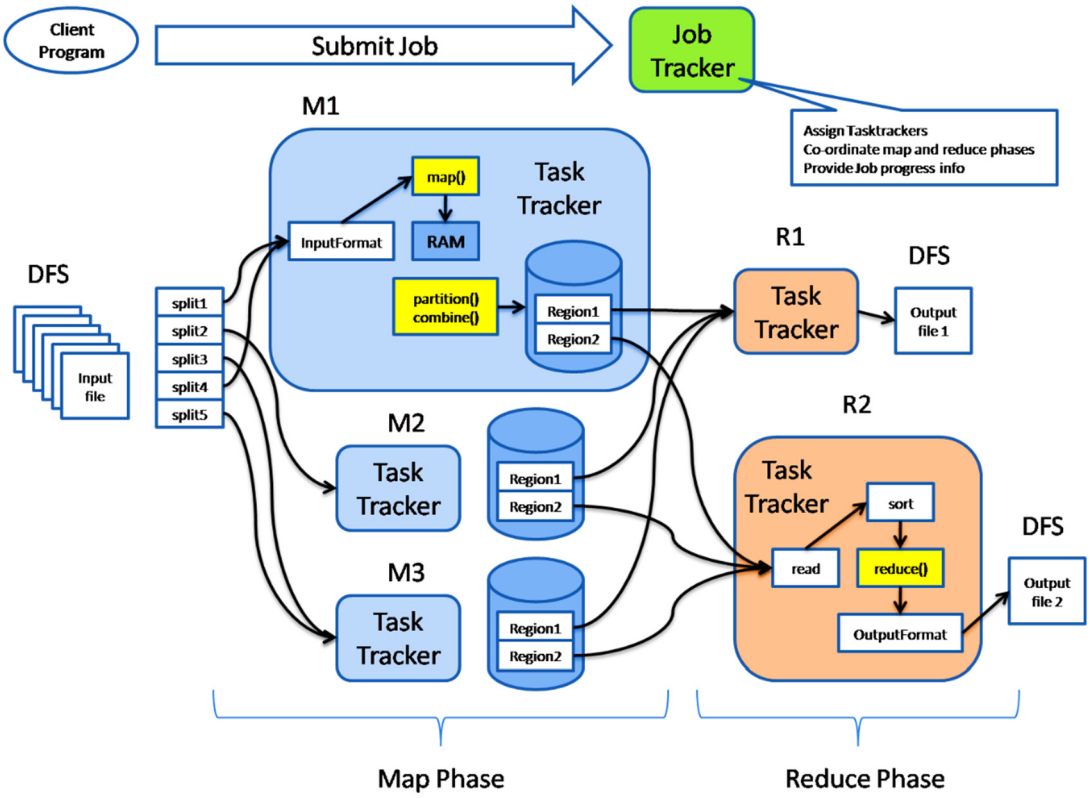

FIGURE 1. The structure of MapReduce [10].

to diverse workloads in terms of requirements efficiently. Hadoop utilizes a special and reliable shared storage system, called Hadoop Distributed File System (HDFS), to manage data access in a distributed approach. The HDFS makes ease to access to a data block for a MapReduce job to run the corresponding big data application in parallel [7]–[9]. It can provide a stable and reliable interface to the application, and build a distributed system with high reliability and scalability. As a result, HDFS implements a reliable shared storage system and MapReduce allows parallel processing based on the two core parts of Hadoop as shown in Fig. 1. [10].

Furthermore, a job scheduling policy gives a dominant impact on the performance of distributed computing systems [11]. Several scheduling policies were introduced to balance workload efficiently and minimize the waiting time of jobs in distributed systems [12]–[14]. However, few researchers, have managed efficient job scheduling algorithms under deadline constraints in cloud-based environments. Varga *et al.* [15] and Perret *et al.* [16] proposed a scheduling algorithm to manage the resource of a cloud computing system. The proposed algorithm is based on linear programming to maximize resource utilization and maximize the waiting time of the job. Buyya *et al.* [17] proposed a scheduling algorithm to reduce the response time of the cloud resource with minimum cost. Several scheduling schemes were proposed to balance the workload based on the amount of available resources [9], [18]–[21]. All these schemes are non-preemptive and do not consider priority based on the deadlines of the submitted jobs. On the contras, several constraints can be considered for job scheduling to improve the system performance, such as least slack time, deadline size of jobs, and so on.

Considering the preemption is an effective approach for avoiding the delay of high priority jobs while allowing the system to be shared among the other regular jobs. In most existing schemes, a high priority job preempts only based on deadline without any other considerations. This may increase the frequency of the preemption which yields to increase preemption and computation overheads. The proposed scheme in this paper attempts to solve these issues by focusing on meeting the deadlines of the jobs in a shared computing environment. This requires accurate estimation of the map and reduce task computation time. While minimization of the job completion time is an NP-hard problem, a least slack time-based scheduling approach allows good performance by utilizing the current status of the system and employing a job execution cost model.

In this paper, the problem of least slack time-base scheduling on a MapReduce model is addressed first. We present a preemptive approach for effectively scheduling the jobs so that the total completion time of the jobs is reduced under given deadlines and least slack time. Then, we propose a least slack time-based preemptive deadline constrain scheduler (LSTPD) which attempts to improve the performance of Hadoop scheduling system. The proposed LSTPD scheme maximizes resource utilization with minimum waiting time

of the jobs in the queue under the preemption of low priority jobs if needed. To this end, a novel job scheduling policy is developed using a preemptive queueing model based on M/M/1 and M/M/C models [22]. The proposed model effectually deals with distributed resources to reduce the response time of the system with heterogeneous workloads. The main contributions of this paper are summarized as follows:

- Dynamic workloads scheduling with queue-wise preemption based on the priority of jobs to maximize the resource utilization of a Hadoop cluster
- Assigning a task to TaskTracker based on the deadline, data locality, and least slack time to minimize the average completion and waiting time for the task
- Heterogenous resource allocation for different types of workloads to achieve high efficiency in term of queueing delay and response time
- Developing a multi-server queuing model applicable to the proposed scheme to improve the schedulability process of MapReduce jobs under different constraints and requirements

The rest of the paper is organized as follows: in Section II, the previous work related to the scheduling of distributed systems is discussed. The proposed scheduling scheme for Hadoop is presented in Section III. In Section IV, the experimental results are shown, and the conclusion is drawn in Section V.

II. RELATED WORK

The primary goal of job scheduling is to maximize throughput and minimize the execution time of the jobs under various requirements and limited resources. Many researchers have contributed to the performance improvement of Hadoop schedulers by optimizing the amount of resources and job execution time. We have observed several research contributions on the scheduling policies for a distributed system. In the following, we discuss real-time task scheduling and Hadoop scheduler.

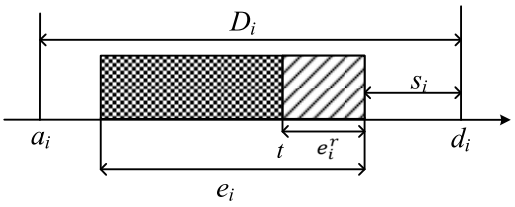

FIGURE 2. The timing diagram of a real-time task.

A. REAL-TIME TASK SCHEDULING

In this section, we describe the basics of real-time task scheduling. Real-time tasks have some constraints such as priority, deadline, and execution time as shown in Fig. 2 [23]–[25].

a_i : arrival time of the task
D_i : deadline for the task
e_i : execution time of the task
s_i : slack time of the task
e_i^r : remaining execution time of the running task

Earliest deadline first (EDF) is a dynamic scheduling algorithm used in real-time task environments to place processes in a priority queue. Whenever a scheduling event occurs, the closest process to its deadline is searched in the queue. The EDF is an optimal preemptive scheduling algorithm, which schedules the collection of jobs so that they are completed until their deadlines [25]. It is expressed as

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \tag{1}$$

where C_i represents the worst-case computation time for i-th process and T_i denotes the sojourn time period of the i-th process

The least slack time (LST) [26] is the scheduling algorithm policy that considers the selected slack time of the task for its priority. The slack times is the remaining time to the deadline $(d_i - e_i^r - t)$ at the current time epoch t. It is expressed as

$$U = \sum_{i=1}^{n} \frac{e_i}{D_i} \leq 1. \tag{2}$$

B. HADOOP SCHEDULERS

In this section, we present the existing schedulers for Hadoop to schedule the submitted MapReduce jobs based on their requirements and available resource in a computing cluster. fair scheduler [27] was proposed to assign average amount of resources to the jobs to be on equally shared all the jobs over time. If there is single job in the system then the job can use all the resources of the entire cluster till another job is arrived and submitted. When a new job arrives, the system will share the resources with a new job so that each job acquires the same amount of CPU time. The JobTracker fairly manages and shares the resources among the tasks in the system. It gives a higher priority to the tasks of which delay is long. Unlike the default Hadoop scheduler, the fair scheduler lets short jobs finish within a reasonable time, while long jobs are not starving. It can also take into account job priorities. The priorities are used as weights to determine the fraction of the total computation time that each job gets. The Fair scheduler organizes jobs into several pools and fairly divides resources fairly among these pools. Within each pool, jobs can be scheduled using either Fair scheduler or first-in-first-out (FIFO) scheduler. In addition to providing fair resource allocation, the fair scheduler allows assigning guaranteed minimum amount of resources to each pool, which is useful for ensuring that certain users, groups or production applications always obtain sufficient resources. In addition, a pool can also be allowed to preempt tasks if it is below half of its fair share under the given timeout. Finally, the Fair Scheduler can limit the number of concurrently running tasks per pool. This can be useful when multiple jobs have a dependency on an external service like database or web that can be overloaded if too many Map or Reduce tasks are run at once.

The Hadoop capacity scheduler (HCS) [28] is another popular scheduler. The main idea behind the HCS is equal resource allocation based on the queue length. It is implementable as multi-queue system and all the resources are equally allocated to each queue. Each job in the queues can obtain the resources that have been already allocated to the corresponding queue. If there exist unallocated resources, they can be used in any queue if its capacity is guaranteed. Inside the queue, all jobs follow the FIFO policy to obtain the required resource and optionally job priority. This algorithm does not support the preemption once the job is running. Therefore, some jobs with high priorities may suffer from a longer delay than its deadline. Hadoop on Demand (HOD) [29] is a scheduler for independently provisioning and managing the resources of the Hadoop system. It automatically de-allocates nodes or resources from a virtual cluster when there are no more running jobs in the cluster. HOD also provides great security with less sharing of the nodes and improves the performance thanks to a lack of contentions among the nodes for multiple users' jobs. However, it has some drawbacks such as in term of poor locality and resource utilization. Dynamic proportional scheduler [30] provides job sharing and prioritization capability in scheduling, with results in increasing share of cluster resources and degrees of differentiation in service levels of different jobs. Recently, many resources have been developing more effective and environment-specific schedulers. Time estimation and optimization for Hadoop jobs have also been investigated. Most of the efforts in scheduling are handling various priorities and the time estimation which is based on runtime running jobs. In [31], a data scheduling locality-based algorithm was proposed. It allocates the input data blocks to the proper nodes based on their processing capacity in order to enhance the performance of MapReduce in heterogeneous Hadoop clusters.

In [32], Liu *et al.* proposed preemptive deadline constraint scheduler (PDCS) for minimizing the completion time of jobs under their deadlines. It preempts the running jobs to fulfill the minimum requirement of new jobs, if it is necessary. The PDCS scheme employs the kill primitive process to allocate slots for the tasks that belong to the nearest deadline. Moreover, a new policy related to map and reduce slot allocations is proposed in the PDCS scheme, it transfers the free slot and data to the ''reduce task'' when a ''map task'' finishes minimizing the waiting time for reducing the slot. As compared to the previous algorithm where the minimum number of slots are allocated to a job to allow other jobs to be scheduled, this preemptive algorithm effectively uses the available resources to avoid job starvation. Qiu *et al.* [33] proposed a scheduling model for heterogeneous workloads in a Hadoop cluster to reduce task outsourcing costs. It is an online algorithm to schedule MapReduce workloads in the hybrid cloud to achieve a minimum cost within delay constraints. By an efficient scheduling mechanism, it improves resource utilization for real-time systems.

In [34], the authors proposed a generalized scheduling algorithm that considers the resource provisioning based on cloud computing based on queueing theory. They modeled a cloud computing system using the M/M/C/C model that allocates the resources to clients based on different service levels with different priority classes. The main performance measure in their analysis is the rejection probability for different customer classes, which can be analytically determined. The proposed solution supports cloud service providers in decision making on 1) defining realistic Service Level Agreements (SLAs), 2) dimensioning of data centers, 3) whether to accept new clients, and 4) the amount of resources to be reserved for high priority clients. In [35], a scheduler to minimize the energy consumption subject to deadlines was proposed for yet another resource negotiator (YARN). It approaches in two levels: job level and task level. At the job level, the scheduler tries to finish the jobs within deadline, while at the task level, it focuses on minimization on the energy consumption of the tasks.

In light of the above discussion on the existing systems, we are motivated to extend the existing work using a preemptive queueing model based on M/M/C, queue model. Several constraints can be considered for job scheduling to improve the system performance, such as least slack time, deadline, size of jobs, and data locality. The proposed scheme is explained in the following section.

III. THE PROPOSED SCHEDULING SCHEME

In this section, the proposed LSTPD scheme for MapReduce job scheduling is presented. The LSTPD scheme attempts to maximize resource utilization and minimize the completion time, response time, and scheduling delay for the submitted jobs under their deadlines and available resources in the system. It determines the priority of the job by calculating the deadline and slack time of the submitted job to eliminate unnecessary preemption and complete it within the given deadlines.

A. DESIGN GOAL

The design goal of the proposed scheme is to improve the performance of Hadoop scheduler in terms of job completion time and resource utilization. The proposed scheme analyzes the submitted MapReduce job with different constraints such as least slack time and the deadline to complete the job with minimum cost and reduce the response time of the Hadoop system. To this end, we employ a multi-priority M/M/C model that schedules the job requests in a preemptive-resume manner [36], [37]. It is a multi-server queueing model where the jobs arrive according to a poison process. In the proposed scheme, the request with a higher priority can achieve reduced waiting time and minimize the missing rate of the deadlines, while the existing non-preemptive scheduler does not consider the priority. Furthermore, since the proposed LSTPD scheme takes into account data locality, the preempted job is resumed on the same TaskTracker or node. In order to improve the schedulability of processes, we proposed a more

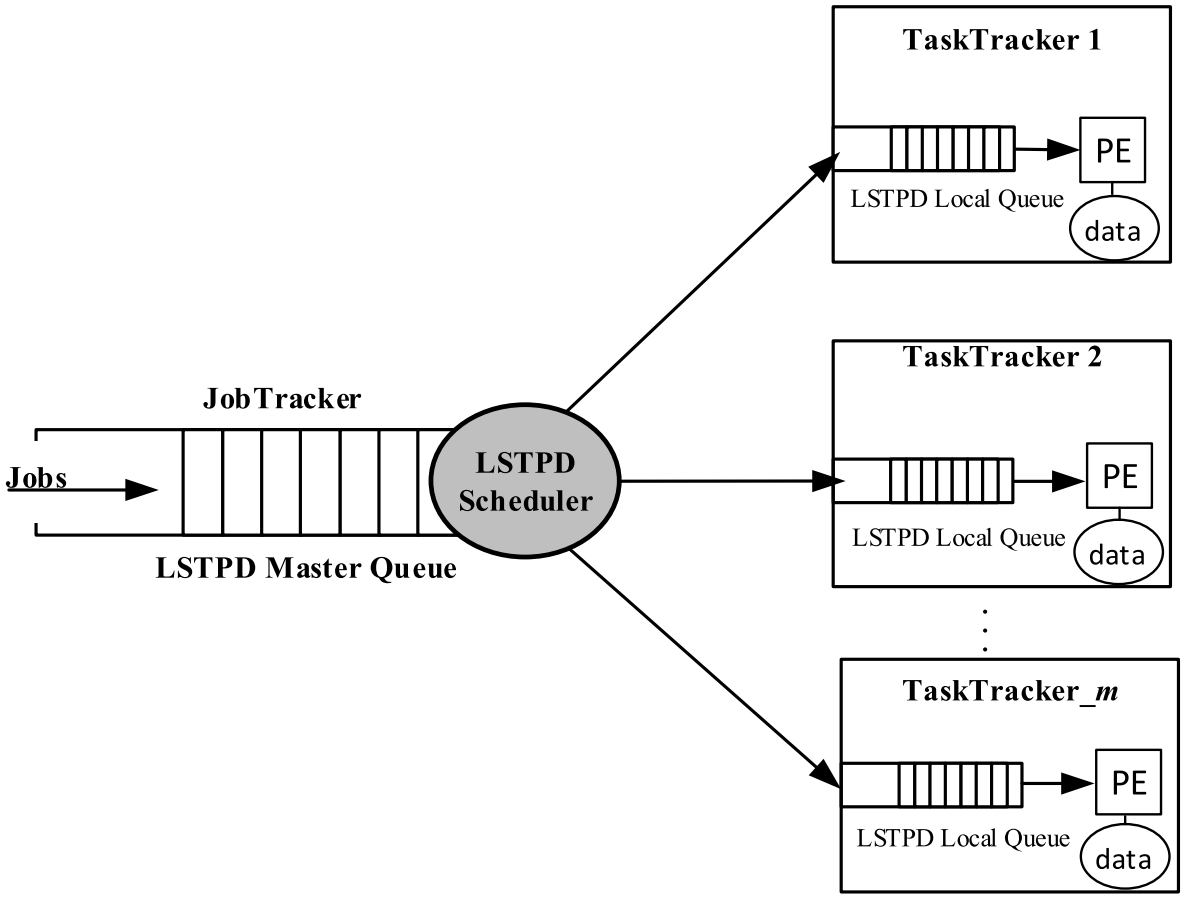

FIGURE 3. The architecture of the proposed $M/M/\beta_i/LSTPD$ queue system.

queueing model, $M/M/\beta_i/LSTPD$, by modifying a typical preemptive queuing model, M/M/C.

B. PROPOSED SCHEDULING

In order to effectively deal with the MapReduce jobs that run with different execution times, the proposed scheme adopts a least slack time-based scheduling for both queues of master and slave nodes. It assigns a priority based on the least slack time of the job. The JobTracker (master) node assigns the arrival tasks to the TaskTracker (salve) nodes based on the available amount of resources. If the JobTracker receives new jobs with no available resources, the new jobs wait in the queue for resources to become idle or to preempt a running job if they have a higher priority. Fig. 3 demonstrates the architecture of the proposed LSTPD scheme. Jobs are arrived at the JobTracker and then, they are transferred to one of TaskTrackers for processing. All the submitted tasks are sorted by each TaskTracker based on slack time and deadline. It is possible that the proposed scheme sends the task, T_{i-1}, to a busy TaskTracker to be waited if the waiting time is less expensive than data transfer cost. Then, the busy TaskTracker contains data, ∂_{i-1}, for the task, T_{i-1}, because of data locality. Consequently, the proposed LSTPD scheme mainly focuses on data locality management, slack time, deadline, and preemption for the tasks.

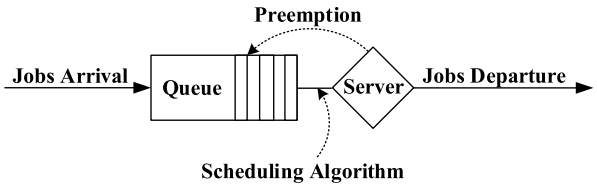

FIGURE 4. The process of job scheduling for services in $M/M/C$.

C. PERFORMANCE ANALYSIS

Queuing theory is an effective tool to deal with different types of queuing systems as shown in Fig. 4. The analytical model of the proposed scheme is based on an M/M/c queuing model [36], [37]. For the proposed LSTPD scheme, we modify it as a novel, $M/M/\beta_i/LSTPD$, model. Based on this model, we evaluated the performance of the proposed scheme in terms of job completion time, average waiting time, resource utilization, job deadline, the probability of missing deadline, and scheduling delay. The variables used in the model are listed in Table 1.

Let us assume, J_i (i = 1,2,3, ...n), is a queue for arrived jobs with an input data size of ∂_i and deadline D_i (i = 1,2,3, ...n). When the task, J_i, arrives at the queue, it is able to preempt the resources allocated to the earlier job, J_{i-1}. Suppose that J_i is the currently running task and the priority of the newly arriving task is higher than that of J_i. If the new task

TABLE 1. The notation used in the model.

Notation	Description
N_s	The total number of Map/Reduce jobs in the system (queue and servers)
W_s	The total time spent by the jobs in the Hadoop system
W_Q	The total waiting time of Map/Reduce jobs waiting in the queue
P_n	The probability of a TaskTracker to be busy for serving on tasks
T_i	Identification of Task i,
C_i	Worst case computation time of Job_i,
C_i'	Remaining computation time of Job_i,
t_i	Current time epoch since the cycle start of the job
D_i	Deadline of Job_i,
R_i	The time spent for the execution of Job_i,
β	Number of TaskTrackers
λ_{mp}	The arrival rate of the MapReduce jobs in the JobTracker system
ρ	Utilization of the queue in the Hadoop system
μ_{mp}	Service time of the MapReduce Jobs in Hadoop system
$W_{U/L}$	The upper/lower bounds of waiting time in the queue
H_i/L_i	High priority and low priority jobs

does not preempt J_i, all the jobs with higher priorities than J_i verify if they can meet their deadlines based on the following Eq. (3) with which defines the condition of preemption.

$$\left(C_i - R_i + W_i + \sum_{m=1}^{n} C_m\right) \leq D_i | i \in Q, \tag{3}$$

where Q is the ready queue, C_i denotes the computation time, and R_i denotes the execution time spent for job, i. Slack time, s, is the difference between the deadline and waiting and execution time. For the i-th job, it is defined as

$$s_i = (D_i - t_i) - C_i^{'}. \tag{4}$$

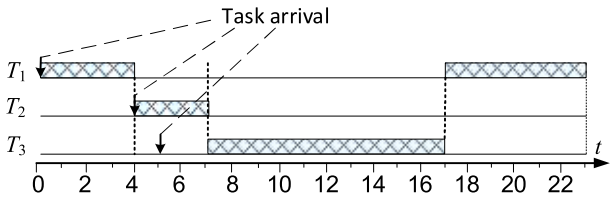

FIGURE 5. A timing diagram of the jobs described in Table 2.

Table 2 lists an example of task executions and Fig. 5 shows the corresponding timing diagram. Note that J_2 has the highest priority while J_1 is the lowest priority based on the duration of the jobs. Assume that, D_i and D_{i-1}, are the deadlines for the running job and the newly arriving job, respectively. Then, we check the following condition to decide preemption.

TABLE 2. An example of task executions.

Task	Arrival	Duration	Deadline
J_1	0	10	33
J_2	4	3	28
J_3	5	10	29

If $D_i < D_{i-1}$ like J_1 and J_2 in the example, no preemption occurs. Otherwise, a preemption occurs like J_2 and J_3 in the example.

In Fig. 5, there are two preemption points. One is the arrival of J_2 at 4. At this point, J_1 is preempted because J_2 has an earlier deadline. The other one is the arrival of J_3 at 5 where J_2 keep keeps running without a preemption since J_3 has a later deadline than J_2.

Suppose that there are β_i servers ($1 \leq i \leq \beta_i$), in which the service rate is μ_i. Then, the total arrival rate is of λ and the total service rate is, $\mu = \prod_{k=1}^{n} \mu\_i, (k = 1, 2, 3, \ldots n)$. It is obvious that the system is stable when $\lambda/\mu < 1$. The jobs arrive in the queue according to a Poisson process and are served by an available server that fulfills the data locality requirements. Otherwise, the jobs preempt any running job based on the deadlines. To evaluate the waiting and slack time of the jobs, the $M/M/\beta_i/LSTPD$, queueing model. M/M/1 model [38] represents a single server that has queue with unlimited capacity and infinite requests, in which inter-arrival and service time follows exponential distributions. A series of different performance measures can be calculated based on the arrival rate, λ, and service rate, μ. As well-known, in the M/M/1 model, the average number of jobs in the system, N, can be calculated as follows.

$$N = \frac{\lambda^2}{\mu\,(\mu - \lambda)}. \tag{5}$$

The probability that an arriving job needs to be waited for service is expressed as

$$\rho = \frac{\lambda}{\mu}, \tag{6}$$

where P_n denotes the server utilization factor or task intensity when n jobs are in the system. The probability that no job/task is waiting in the system is expressed as

$$P_o = 1 - \frac{\lambda}{\mu}. \tag{7}$$

From Eq. (6) and (7), we can generalize the formula to compute the probability of n waiting for jobs in the queue as

$$P_n = \left(\frac{\lambda}{\mu}\right)^n P_o. \tag{8}$$

With multiple servers, each arriving request waits in a single queue, and then it is moved to the first available server to be served. The utilization of the queue, ρ, is written by

$$\rho = \frac{\lambda_{mp}}{\beta_i \mu_{mp}}, \tag{9}$$

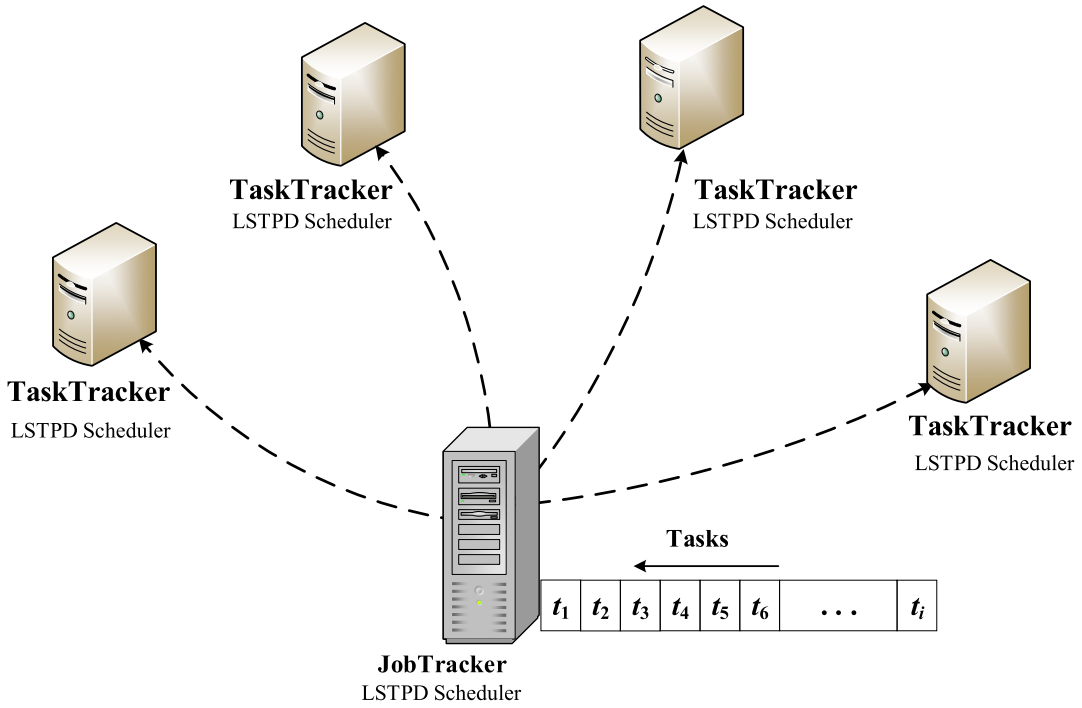

FIGURE 6. Hadoop simulation environments for LSTPD.

where λ_{mp}, is the MapReduce job, μ_{mp}, is the service rate, and β_i denotes TaskTrackers (servers). As the requests of a cloud system come from all over the world, the pool of the jobs and the size of the queue are deemed to be unlimited. Let us denote a set of states of the system is, $E = \{0, 1, 2, \ldots\}$, where each state denotes the number of MapReduce jobs in the Hadoop system. When the number of jobs is less than β_i, only n out of β_i nodes are busy and the mean service rate is equal to $\beta_i\mu$. In this case, the probability that there are n jobs in the system is expressed as

$$P_n = p_0 \left(\frac{\lambda_{mp}}{\mu_{mp}}\right)^n \frac{1}{n!}, \quad \text{if } 1 \leq n \leq \beta_i.\mathrm{p_n} \tag{10}$$

If the number of MapReduce jobs is greater than or equal to β_i, i.e., n $\geq \beta_i$, all the nodes are busy and the effective service rate is equal to $\beta_i\mu$. Thus, Eq. (10) can be rewritten by

$$P_n = p_0 \left(\frac{\lambda_{mp}}{\mu_{mp}}\right)^n \frac{1}{\beta_i!} \left(\frac{1}{\beta_i!}\right)^{n-\beta_i} \quad \text{if } n \geq \beta_i. \tag{11}$$

Here, $\rho = \lambda_{mp}/\beta_i\mu_{mp}$, must be less than 1 for the system to be stable. Note that the expected number of busy nodes is equal to $\beta_i\rho$, which is, λ_{mp}/μ_{mp}. Returning to Eqns. (10) and (11) by substituting $\beta_i\rho$ for λ/μ, we obtain

$$P_n = P_0 \frac{(\beta_i\rho)^n}{n!}, \quad \text{for } n \leq \beta_i, \tag{12}$$

$$P_0 = \frac{(\beta_i\rho)^n}{\beta_i^{n-\beta_i}\beta_i!}, \quad \text{for } n > \beta_i. \tag{13}$$

To obtain P_0, both sides of Eqns. (12) and (13) are summed up, and since $\sum_{n=0}^{\infty} \rho_n = 1, P_0$ is obtained as

$$P_o = \frac{1}{\sum_{n=0}^{k-1} \frac{(\lambda_{mp}/\mu_{mp})^n}{n!} + \frac{(\lambda_{mp}/\mu_{mp})^k}{k!}\left(\frac{k\mu_{mp}}{k\mu_{mp}-\lambda_{mp}}\right)}. \tag{14}$$

In addition, the average number of jobs in the queue is computed as.

$$N_q = \frac{(\lambda_{mp}/\mu_{mp})^n \mu_{mp}\lambda_{mp}}{(k-1)!\left(k\mu_{mp}-\lambda_{mp}\right)^2}. \tag{15}$$

Assuming that the maximum number of jobs in the queue is k and the maximum queue length is (k−1), the upper bound of waiting time in the queue is given by follows.

$$W_U = \sum_{n=0}^{k-1} \frac{(\lambda_{mp}/\mu_{mp})^n}{k} + \frac{(\lambda_{mp}/\mu_{mp})^k}{n!}\left(\frac{k\mu_{mp}}{k\mu_{mp}-\lambda_{mp}}\right). \tag{16}$$

Moreover, the lower bound of waiting time also given by

$$W_L = \sum_{n=0}^{k-1} \left(\frac{(\lambda_{mp}/\mu_{mp})^n}{k} + \frac{k\mu_{mp}}{k\mu_{mp}-\lambda_{mp}}\right). \tag{17}$$

Eqns. (16) and (17) can be generalized to compute the probability that there exist n waiting jobs in the queue as

$$P_n = \frac{(\lambda_{mp}/\mu_{mp})^n}{n!} P_o \quad \text{for } n \leq k, \tag{18}$$

$$P_n = \frac{(\lambda_{mp}/\mu_{mp})^n}{kk!^{(n-k)}} P_o \quad \text{for } n > k. \tag{19}$$

For a preemptive, $M/M/\beta_i/LSTPD$ queue, the job with a higher priority, H_i, and a lower priority L_i arrive at the system according to the Poisson process as with the arrival rates of λ_{H_i} and λ_{L_i}, respectively. Their service time and arrival time are independent of each other. According to Eq. (6),

$$\rho_{H_i} + \rho_{L_i} < 1. \tag{20}$$

Let, $\rho_k = \lambda_k/\mu_k$, the queue utilization of jobs, J_k, with the preemptive-resume policy, where the lower priority job, L_i, is interrupted and the server proceeds with the higher priority job, H_i. Once there is no higher priority job in the system, the server resumes the service of the lower priority job from the moment where it was interrupted. The priority here is decided by the deadline. As mentioned earlier, the jobs are resumed in the same TaskTracker where they were interrupted in order to support data locality.

Let N_{H_i} and W_{H_i} denote the number of jobs of the i-th priority and the sojourn time in the system, respectively. The average values of them, $E[S_{H_i}]$ and $E[N_{H_i}]$, are derived as

$$E[S_{H_i}] = \frac{1/\mu_{H_i}}{1-\rho_{H_i}}. \tag{21}$$

$$E\left[N_{H_i}\right] = \frac{\rho_{H_i}}{1-\rho_{H_i}}. \tag{22}$$

Since the remaining service time of the jobs is exponentially distributed with the same mean, the total number of jobs in the system does not depend on the order that the jobs are served. Therefore, the number is the same as the case where all the jobs are sequentially served in the order of arrival. Hence,

$$E\left[N_{H_i}\right] + E\left[N_{H_L}\right] = \frac{\rho_{H_i} + \rho_{L_i}}{1-\rho_{H_i}-\rho_{L_i}}. \tag{23}$$

By the Little's law,

$$E\left[S_{H_L}\right] = \frac{1/\mu_{H_i}}{\left(1-\rho_{H_i}\right)\left(1-\rho_{H_i}-\rho_{L_i}\right)}, \tag{24}$$

$$W_Q = \frac{\sum_{H_i=1}^{k} \lambda_{L_i}\overline{\sigma}_j^2}{2\left(1-\sum_{L_i=1}^{k-1}\rho_{L_i}\right)\left(1-\sum_{L_i=1}^{k}\rho_{L_i}\right)}, \tag{25}$$

where W_Q represents the average waiting time in the queue. As we mentioned in the previous section, the main goal of the proposed LSPD scheme is to improve the performance by preemption based on the deadline in the local scheduling of the queues. To this end, the deadline, residual time, waiting time, sojourn time of the preempted job, and the total number of jobs needs to be estimated. The schedulability analysis [39] requires worst-case execution time of the jobs as well as that of operating system functions in order to check if a given set of jobs can be successfully executed without missing the deadline. To evaluate the waiting and slack time of the jobs, the proposed, $M/M/\beta_i/LSTPD$, queueing model is used. The proposed LSTPD scheduling algorithm for both JobTracker and TaskTracker are describe in Algorithm1 and Algorithm2, respectively.

Algorithm 1 JobsTracker_Schedular

Initialize the values:
Q_{Job}: Queue of jobs in the JobTracker
Q_{Node}: Queue of node in cluster
N_{status}: status of node in cluster (idle/busy)
D_i: Deadline for jobs.
Procedure:
Check status of TaskTracker (heartbeats)
$N_{status} \leftarrow Idle$
$N_{status} \leftarrow busy$
While *JobQueue is not empty* **do**
if N_{status} *is idle* **then**
if $(D_i < D_{i-1})$ **then**
$N_{idle} \leftarrow Q_{Job}[i]D_i$
$N_{status} \leftarrow busy$
end if
else
check and update the status of the node
$N_{status} \leftarrow Idle$
$N_{status} \leftarrow busy$
end if
end while

IV. PERFORMANCES EVALUATION

In this section, we describe our experimental setup for evaluating the proposed LSTPD scheduling scheme by using Hadoop 2.7.2 (stable), Linux operating system, and CentOS. The proposed LSTPD scheduler is compared with four existing scheduling schemes developed for Hadoop: FIFO, Fair, Capacity, and Proactive and Reactive Scheduling (PRS) [27], [28], [40]. The efficiency of the proposed scheduling scheme is evaluated in terms of job completion time, slot utilization, and deadline under least slack time and preemption techniques. Two data-intensive applications, TeraSort and WordCount [41]–[43], are used to evaluate the performance of the proposed scheme in a heterogeneous Hadoop cluster. TeraSort and WordCount are MapReduce applications running on Hadoop clusters. WordCount is a benchmark that measures the performance of MapReduce by counting words in text files. It splits the input text into words, shuffles every word in the map phase, and counts its number of occupations in reduce phase. It counts the occurrences of each word in the input file. WordCount runs on different size of data, e.g., 1∼5 GB. TeraSort is a popular benchmark that measures MapReduce performance and the elapsed time to sort randomly distributed data. TeraSort runs on 1 GB of data. A full TeraSort benchmark run consists of the following three steps [43]:

1. Generating the input data via TeraGen.
2. Running the actual TeraSort on the input data.
3. Validating the sorted output data via TeraValidate.

As shown in Fig. 6, the tested cluster consists of a single JobTracker (Master) and ten TaskTrackers (Salves) nodes, and twenty heterogeneous nodes, whose parameters are summarized in Table 3.

TABLE 3. Node setup and Hadoop configuration.

Nodes	Qty	Processor (x64)	HDisk	RAM	Map slots	Reduce slots	HDFS block size	Benchmarks (550 jobs)
JobTracker	1	Intel-core-i5, 3.2GHz	1 TB	8GB			64 MB	WordCount & TeraSort
TaskTrackers	5	Intel-core-i3, 3.1GHz	500GB/node	4GB	8/node	4/node		
TaskTrackers	5	Intel-dual-core, 3.3 GHz	500GB/node	2GB	4/node	2/node		

Algorithm 2 TaskTracker_Schedular

Initialize the values:
Q_{Task}: Queue of tasks in TaskTracker
TT_{status}: The state of each (idle/busy) on the TaskTracker
ST_i: slack time of job i
Procedure
Check status of resources
$TT_{status} \leftarrow Idle$
$TT_{status} \leftarrow busy$
While *TaskQueue is not empty* **do**
if TT_{status} *is idle* **then**
Schedule the next job from $Q_{Task}[i]$ *to the TT*
Remove the task from $Q_{Task}[i]$ & set $Q_{TT}[i]$ as busy
if(*TaskQueue is not empty*) &
(*All resources are busy*) **then**
if ($ST\_Q_{Task}[i] < ST\_runnTask$) **then**
(*Preempt running task and assign PE to new task*)
$Q_{Task}[i+1] \leftarrow ST\_runnTask$
$Q_{PE}[i] \leftarrow ST\_Q_{Task}[i]$
end if
end if
end if
end while

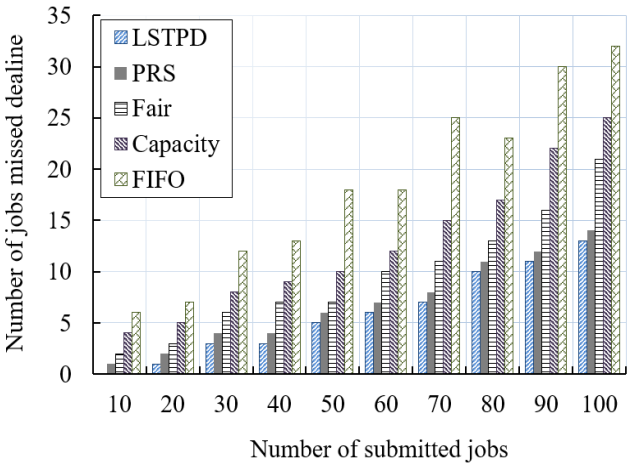

FIGURE 7. Comparison of missing deadline.

Fig. 7 shows the performance comparison of the conventional schemes with the proposed LSTPD scheme in terms of jobs missed deadline. This figure shows the result for varying workloads as the number of submitted jobs (10 to 100). All these jobs are heterogeneous in terms of requirements. The proposed LSTPD scheme schedule the jobs based on the deadlines. Therefore, the proposed LSTPD scheme achieves 40% $\sim$ 50% better performance compared to the other three schemes: Fair, Capacity, and FIFO schedulers in terms of the number of jobs missed deadline. The proposed LSTPD scheme maximizes the resource utilization and by minimizing the number of deadline missing.

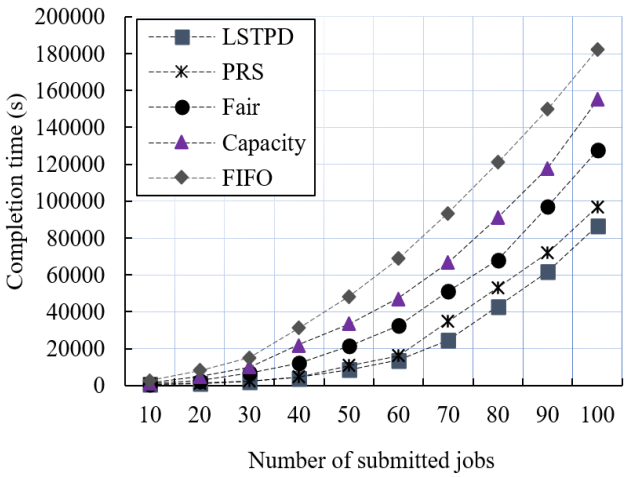

FIGURE 8. Comparision of total completion time.

Fig. 8 shows the total completion time of the proposed LSTPD scheme for varying workloads, compared with the conventional schemes. As shown in the figure the difference among the total completion time of the 10-100 jobs for each scheduling scheme gradually increases as the number of submitted jobs increases. Among the schemes, the proposed LSTPD scheme achieves the best performance by finishing the submitted jobs in minimum time shows better performance to finished the submitted jobs in minimum time.

Fig. 9 shows the result of average completion time by running 550 number of jobs with deference groups from 10 to 100. For each group of jobs, the average completion time is computed. As shown in the figure, the proposed LSTPD scheme outperforms the other schemes by 30%$\sim$50% on average.

Fig. 10 shows the average waiting time of the submitted jobs. It is shown that the waiting time of the proposed LSTPD scheme is basically smaller than the other conventional schemes. As the number of jobs increase the proposed

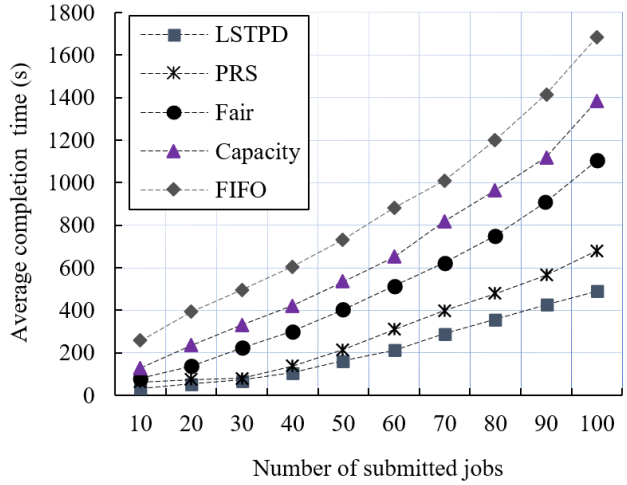

FIGURE 9. Comparision of average completion time.

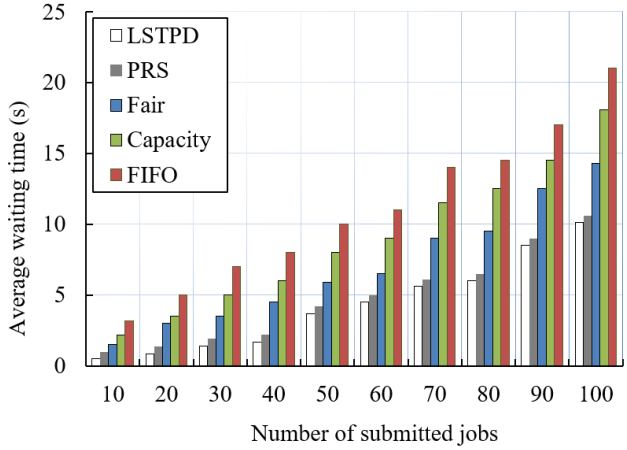

FIGURE 10. Comparision of average waiting time.

LSTPD scheme significantly outperforms the other schemes. The reason is that the proposed LSTPD scheme considers deadlines and slack time in every scheduling epochs, while the other schemes do not consider the priorities which can yield long waiting time due to lower priority jobs.

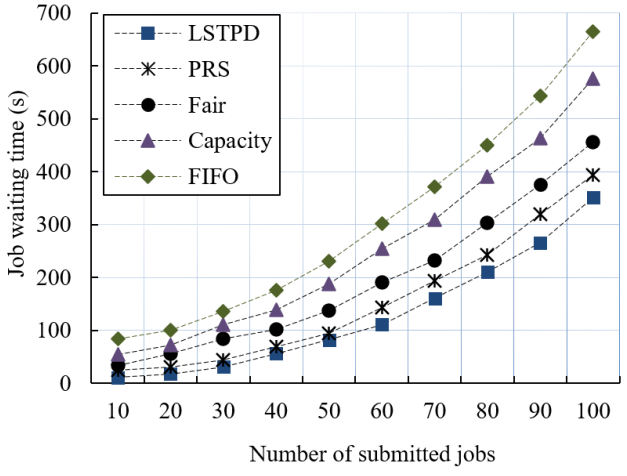

FIGURE 11. Comparison of job waiting time.

Fig. 11 shows the total waiting time of proposed and conventional schemes for varying the number of submitted jobs.schemes. It is observed that the total waiting time of the proposed LSTPD scheme is always smaller than the conventional schemes regardless decrease of workloads. This implies that the proposed LSTPD scheme is effective in average sense in terms of total waiting time.

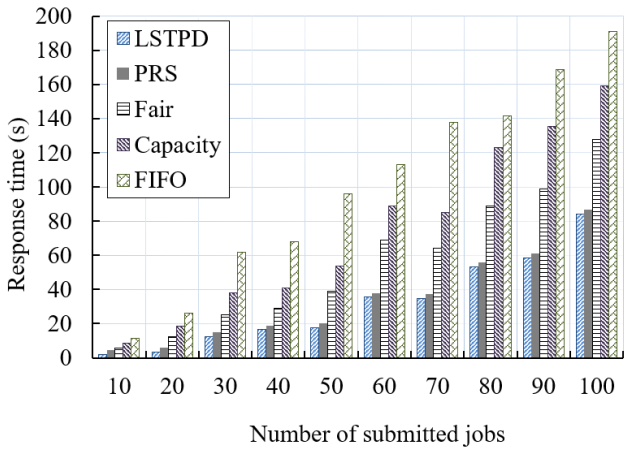

FIGURE 12. Comparison of response time.

Fig. 12 shows the response time of the schedulers for varying. For each batch of jobs, we calculate the response time of jobs and present the results in the figure. For all of the ten batches of heterogeneous jobs, the response time of the proposed LSTPD scheme is 40 %∼50 % smaller than the other three scheduling schemes, while the other scheme gets significantly longer response time because they basically use almost the same policy in the queue. They never consider the deadline and priority of jobs and thus, the overall results of these schemes in terms of response time are very poor.

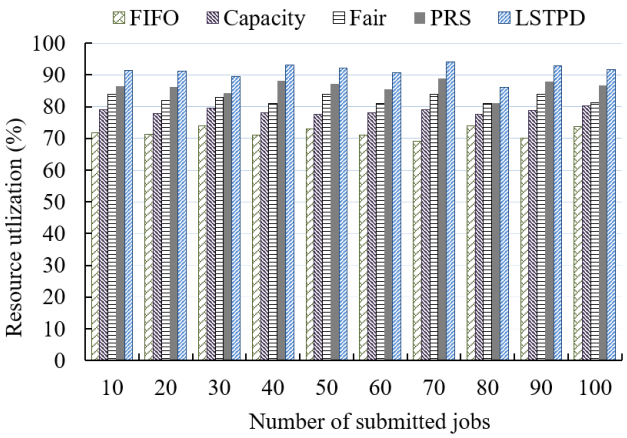

FIGURE 13. Comparision of resource utilization.

Fig. 13 shows the resource utilization for varying the number of submitted jobs. The resource utilization for the proposed LSTPD scheme is better than the other three schemes. This is because for the Poisson distribution, at the tail of the queue, the size of jobs is reduced, and these jobs can

be completed before the big sized-jobs arrive. For the other three scheme high priority jobs can be delayed until low parity or big sized-jobs are completed. The average resource utilization of the proposed LSTPD scheme is 93% which seems sufficiently good, compared to the other schemes.

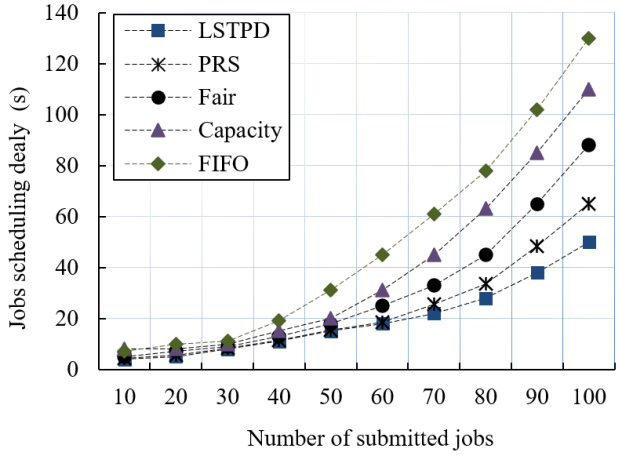

FIGURE 14. Comparison of scheduling delay.

In order to observe the delay of the scheduling process, we submitted the 550 MapReduce jobs for WordCount and TeraSort benchmarks. For real-time heterogeneous jobs in term of the deadline and slack time, the proposed LSTPD scheme achieves shorter scheduling delay, compare to the other three schemes as shown in Fig. 14. The proposed LSTPD scheme considers the priority for the jobs with respect to deadline and slack time. Hence, the proposed LSTPDscheme shows better performance and attempts to process all the high priority jobs with minimum delay. Furthermore, the proposed LSTPD scheme considers high data locality for the jobs with respect to heterogeneous demands in resources request.

V. CONCLUSION

MapReduce jobs scheduling and resource allocation have a great impact on Hadoop clusters, especially in heterogeneous environments. Big data analytics and workload management become more complex if the job and resources are heterogeneous. In this paper, we proposed a preemptive scheduler based on deadline and slack time for MapReduce to improve the performance of the Hadoop. When the slack time for the jobs is diverse with respect to the deadline and data locality, the different number of jobs can be assigned to TaskTracker. In addition, in this paper, we also quantitatively describes the worst-case waiting time in the ready queue. The proposed least slack time preemptive deadline constraint scheduler (LSTPD) scheme avoids unnecessary preemption if a certain condition is satisfied based on the slack time of the jobs. It also improves schedulability regarding available resources. We did not include constraint scheduling such as runtime estimation of Map or Reduce task, filter ratio estimation, data distribution and dynamic data replication base on job scheduling. We plan to address these issues as future work. In the future, we will consider the aspects that may have effects on the performance of scheduling including data distributions and replication in heterogeneous MapReduce system, and the iterative MapReduce jobs under deadline.

REFERENCES

[1] M. A. Belarbi, S. A. Mahmoudi, S. Mahmoudi, and G. Belalem, "A new parallel and distributed approach for large scale images retrieval," in *Proc. Int. Conf. Cloud Comput. Technol. Appl.*, 2017, pp. 185–201.

[2] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014.

[3] J.-M. Shih, C.-S. Liao, and R.-S. Chang, "Simplifying MapReduce data processing," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2011, pp. 366–370.

[4] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 739–750, Mar. 2013.

[5] H. Wang and Y. Cao, "An energy efficiency optimization and control model for Hadoop clusters," *IEEE Access*, vol. 7, pp. 40534–40549, 2019.

[6] T. Revathi, K. Muneeswaran, and M. B. B. Pepsi, *Big Data Processing With Hadoop*. Hershey, PA, USA: IGI Global, 2018.

[7] I. A. T. Hashem, N. B. Anuar, A. Gani, I. Yaqoob, F. Xia, and S. U. Khan, "MapReduce: Review and open challenges," *Scientometrics*, vol. 109, no. 1, pp. 389–422, Oct. 2016.

[8] D. Glushkova, P. Jovanovic, and A. Abelló, "Mapreduce performance model for Hadoop 2. x," *Inf. Syst.*, vol. 79, pp. 32–43, Jan. 2019.

[9] N. S. Dey and T. Gunasekhar, "A comprehensive survey of load balancing strategies using Hadoop queue scheduling and virtual machine migration," *IEEE Access*, vol. 7, pp. 92259–92284, 2019.

[10] S. User. *Hadoop Map Reduce Architecture and Example*. Accessed: Mar. 31, 2020. [Online]. Available: http://a4academics.com/tutorials/83-hadoop/840-map-reduce-architecture

[11] N. Deshai, B. Sekhar, S. Venkataramana, K. Srinivas, and G. Varma, "Big data Hadoop MapReduce job scheduling: A short survey," in *Information Systems Design and Intelligent Applications*. Springer, 2019, pp. 349–365.

[12] A. Sharma and G. Singh, "A review of scheduling algorithms in Hadoop," in *Proceedings of ICRIC 2019*. Springer, 2020, pp. 125–135.

[13] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic MapReduce scheduler for heterogeneous workloads," in *Proc. 8th Int. Conf. Grid Cooperat. Comput.*, Aug. 2009, pp. 218–224.

[14] A. Rasooli and D. G. Down, "COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems," *Future Gener. Comput. Syst.*, vol. 36, pp. 1–15, Jul. 2014.

[15] M. Varga, A. Petrescu-Nita, and F. Pop, "Deadline scheduling algorithm for sustainable computing in Hadoop environment," *Comput. Secur.*, vol. 76, pp. 354–366, Jul. 2018.

[16] Q. Perret, G. Charlemagne, S. Sotiriadis, and N. Bessis, "A deadline scheduler for jobs in distributed systems," in *Proc. 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Mar. 2013, pp. 757–764.

[17] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency Comput., Pract. Exper.*, vol. 14, nos. 13–15, pp. 1507–1542, Nov. 2002.

[18] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *J. King Saud Univ.-Comput. Inf. Sci.*, early access, Feb. 1, 2020, doi: 10.1016/j.jksuci.2020.01.012.

[19] P. Hamandawana, R. Mativenga, S. J. Kwon, and T.-S. Chung, "Towards an energy efficient computing with coordinated performance-aware scheduling in large scale data clusters," *IEEE Access*, vol. 7, pp. 140261–140277, 2019.

[20] Y. Wu, X. Song, and G. Gong, "Real-time load balancing scheduling algorithm for periodic simulation models," *Simul. Model. Pract. Theory*, vol. 52, no. 1, pp. 123–134, Mar. 2015.

[21] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Trans. Services Comput.*, early access, Aug. 21, 2018, doi: 10.1109/TSC.2018.2866421.

[22] E. Gelenbe, G. Pujolle, and J. Nelson, *Introduction to Queueing Networks*. Princeton NJ, USA: Citeseer, 1998.

[23] L. Yang, X. Liu, J. Cao, and Z. Wang, "Joint scheduling of tasks and network flows in big data clusters," *IEEE Access*, vol. 6, pp. 66600–66611, 2018.
[24] H. Alhussian, N. Zakaria, A. Patel, A. Jaradat, S. J. Abdulkadi, A. Y. Ahmed, H. T. Bahbouh, S. O. Fageeri, A. A. Elsheikh, and J. Watada, "Investigating the schedulability of periodic real-time tasks in virtualized cloud environment," *IEEE Access*, vol. 7, pp. 29533–29542, 2019.
[25] Wikipedia. (Mar. 8, 2017). *Earliest Deadline First Scheduling*. Accessed: Jan. 17, 2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Earliest_deadline_first_scheduling&oldid=769243721.
[26] M. Hwang, D. Choi, and P. Kim, "Least slack time rate first: An efficient scheduling algorithm for pervasive computing environment," *J. UCS*, vol. 17, no. 6, pp. 912–925, 2011.
[27] *Fair Scheduler*. Accessed: Apr. 15, 2017. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html
[28] *CapacityScheduler Guide*. Accessed: Apr. 15, 2015. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html
[29] *HOD Scheduler*. Accessed: Apr. 15, 2015. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hod_scheduler.html#Introduction
[30] T. Sandholm and K. Lai, "Dynamic proportional share scheduling in Hadoop," in *Proc. Workshop Job Scheduling Strategies Parallel Process.*, 2010, pp. 110–131.
[31] N. S. Naik, A. Negi, T. B. Br, and R. Anitha, "A data locality based scheduler to enhance MapReduce performance in heterogeneous environments," *Future Gener. Comput. Syst.*, vol. 90, pp. 423–434, Jan. 2019.
[32] L. Liu, Y. Zhou, M. Liu, G. Xu, X. Chen, D. Fan, and Q. Wang, "Preemptive Hadoop jobs scheduling under a deadline," in *Proc. 8th Int. Conf. Semantics, Knowl. Grids*, Oct. 2012, pp. 72–79.
[33] X. Qiu, W. Leong Yeow, C. Wu, and F. C. M. Lau, "Cost-minimizing preemptive scheduling of MapReduce workloads on hybrid clouds," in *Proc. IEEE/ACM 21st Int. Symp. Qual. Service (IWQoS)*, Jun. 2013, pp. 1–6.
[34] W. Ellens, M. Ivkovic, J. Akkerboom, R. Litjens, and H. van den Berg, "Performance of cloud computing centers with multiple priority classes," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 245–252.
[35] X. Cai, F. Li, P. Li, L. Ju, and Z. Jia, "SLA-aware energy-efficient scheduling scheme for Hadoop YARN," *J. Supercomput.*, vol. 73, no. 8, pp. 3526–3546, Aug. 2017.
[36] H. S. Bakouch, "Probability, Markov chains, queues, and simulation," *J. Appl. Statist.*, vol. 38, no. 8, p. 1746, 2011.
[37] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, 4th ed. Hoboken, NJ, USA: Wiley, 2016. [Online]. Available: http://as.wiley.com/WileyCDA/WileyTitle/productCd-047179127X.html
[38] G. Balbo and M. G. Vigliotti, "On the analysis of a M/M/1 queue with bulk services," *Comput. J.*, vol. 58, no. 1, pp. 57–74, Jan. 2015.
[39] S. Baruah, M. Bertogna, and G. Buttazzo, "Global fixed-task-priority scheduling," in *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2015, pp. 165–172.
[40] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *J. Syst. Softw.*, vol. 99, pp. 20–35, Jan. 2015.
[41] *Apache Hadoop 3.2.1—MapReduce Tutorial*. Accessed: May 19, 2020. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
[42] O. O'Malley. (May 2008). Terabyte sort on Apache Hadoop. Yahoo. [Online]. Available: http://sortbenchmark.org/Yahoo-Hadoop.pdf
[43] *Benchmarking and Stress Testing an Hadoop Cluster with TeraSort, TestDFSIO & Co*. Accessed: May 19, 2020. [Online]. Available: https://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/

IHSAN ULLAH received the B.S. and M.S. degrees in computer science from the University of Peshawar, Pakistan, in 2001 and 2004, respectively, and the Ph.D. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2019. He is currently a Research Fellow with UTRI, Sungkyunkwan University. His research interests include data aggregation, data fusion, the Internet of Things (IoT), intelligent system artificial intelligence, machine learning, distributed computing, and wireless sensor networks.

MUHAMMAD SAJJAD KHAN received the B.Sc. degree in computer information systems engineering from the University of Engineering and Technology (UET) at Peshawar, Pakistan, in 2004, the M.Eng. degree from Mehran UET, Jamshoro, Pakistan, in 2007, and the Ph.D. degree in electrical engineering from the University of Ulsan, South Korea, in 2016. Since 2011, he has been an Assistant Professor with the Department of Electrical Engineering, International Islamic University, Islamabad, Pakistan. He is currently working as a Research Professor with Korea Polytechnic University, Siheung, South Korea, since December 2018. His research interests include the Internet of Things (IoT), machine learning, distributed computing, and next generation wireless communication systems.

MUHAMMAD AMIR (Member, IEEE) received the B.Sc. degree from UET at Peshawar, the M.Sc. degree from UET at Taxila, and the Ph.D. degree from International Islamic University Islamabad, Islamabad, Pakistan, all in electrical engineering. He is currently serving as a Professor and the Dean Faculty of engineering and technology with International Islamic University Islamabad. He has more than 40 publications in reputed journals and conferences. His research interest includes signals and image processing.

JUNSU KIM received the B.S., M.S., and Ph.D. degrees in electric engineering and computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2001, 2003, and 2009, respectively. From March 2009 to August 2009, he was a Postdoctoral Research Fellow supported by BK21. From October 2009 to August 2011, he was a Postdoctoral Fellow with The University of British Columbia, Canada. Since September 2011, he has been with Korea Polytechnic University (KPU), Siheung, South Korea, where he is currently an Associate Professor. His research interests include cognitive radio resource management, wireless scheduling algorithm, cognitive radio systems, cooperative diversity techniques, and physical layer.

SU MIN KIM (Member, IEEE) received the B.S. degree in electronics engineering from Inha University, South Korea, in 2005, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2007 and 2012, respectively. He was a Postdoctoral Research Fellow with the Department of Electrical Engineering, KAIST, in 2012, and the School of Electrical Engineering, KTH Royal Institute of Technology, Sweden, from 2012 to 2014. Between 2014 and 2015, he was an Experienced Researcher in radio access technology at Ericsson Research, Kista, Sweden. He has been an Assistant Professor with the Department of Electronics Engineering, Korea Polytechnic University, since March 2015. His research interests include next generation mobile communication systems, hybrid automatic repeat request (HARQ) protocol, radio resource management, interference management, cooperative and buffer-aided relaying communications, cognitive radio communications, physical layer security, statistical signal processing, the Internet of Things (IoT), and ambient backscatter communications.