IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 19, 2020, accepted March 10, 2020, date of publication March 20, 2020, date of current version April 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982320

Hawkeye: Adaptive Straggler Identification on Heterogeneous Spark Cluster With Reinforcement Learning

HAIZHOU DU[1,2] AND SHAOHUA ZHANG[2], (Member, IEEE)
[1]College of Electronic and Information Engineering, Tongji University, Shanghai 201804, China
[2]School of Computer Science and Technology, Shanghai University of Electric Power, Shanghai 200090, China

Corresponding author: Haizhou Du (haizhou.du@shiep.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672385.

ABSTRACT It is a common sense that people harbors the belief that stragglers exert huge influence upon the performance conducted by the analysis systems of big data for the reason of poor performance made by some computing nodes, data skew and so on. Accordingly, stragglers have been billed as an indispensable bottleneck in Map-Reduce framework processing. However, existing studies on stragglers identification are targeting coarse-grained detection, schedule level optimization and off-line log based cause analysis. Accuracy identifying the stragglers in time for each job, however, is an extremely tough because (1) Auite a number of root causes for stragglers in data analytics frameworks;(2) The number of key parameters affecting stragglers identification; and (3) The different clusters configurations, and their impact on the stragglers detection, vary among different job types and sizes. Either existing solutions adopt a ''tweak-and-pray'' manual tuning approach, which is complex, time-consuming and error-prone, or only most of them fix theirs eyes upon coarse-grained straggler detection. In this paper, we systematically conduct the exploration on the fundamental problem of automatic, adaptive straggler identification on big data analytics platform. Under the inspiration of the recent triumphs over implementing Reinforcement Learning (RL) techniques for solving complex online optimal problems, we conducted investigation that Reinforcement learning are reasonably employed to adaptively opt the optimal parameters to identify stragglers free of the intervention of human beings. Specifically, we propose Hawkeye, a general adaptive speculative execution system which identifies stragglers by reinforcement are learning to launch speculative tasks on heterogeneous cluster at runtime. In accordance with the experimental conclusions, Hawkeye manages to cut down the job completion time over the distinct type applications. An instance is that it reveals as many as nearly 37% decrease average job completion time based on an improvement of 23% on the preciseness of the present resolutions to the heterogeneous cluster.

INDEX TERMS Spark cluster, speculative execution, reinforcement learning, Pareto distribution, straggler identification.

I. INTRODUCTION

The last dozen years have witnessed that big data analysis platform has enjoyed unprecedented accumulating massive data processing, which exceeding the storage capability harbored by any of the single machine. Popularized frameworks to handle large volume of data have been applied are adopted by enterprises and research institutes, which helps draw out the applicable data. Apache Spark [1] is a typical example, which is known as a popular open-source Map-Reduce programming platforms. It brings into the conception of resilient distributed datasets (RDDs) [2] enabling the fast processing of large-scaled data leveraging distributed memory. Due to the high-performance variability, the execution time of tasks is able to change within a very large range in the same job. The execution time of large numbers of tasks are still nearby the average execution time, some of which might display a large deviation. Additionally, it is not essentially a routine to put some tasks under observation based on the longer execution time as many as. It is not unusual in practice to observe

The associate editor coordinating the review of this manuscript and approving it for publication was Paul D. Yoo.

some tasks with execution time up to 8 times compared to the average execution time [3]. The phenomenon is defined as heavy-tail distribution [4]. As a result, the difficulty is derived out of the reality that the job's execution time on Spark tends to reversely affected by a small number of slow tasks, which, consequently, will lead the tasks to the likely sufferings from the highly latent circumstances. The delayed tasks which conducted relatively slower performance than the sibling jobs, are referred to as stragglers. Service response time functions as a crucial indispensable factor in pay-by-the-hour conditions like Amazon EC2, as well as the systems requiring rapid response to the users. Nonetheless, a required factor of this kind is inclined to grow more and more a difficult job in large-scaled systems for the reasons of stragglers.

An huge work-body, which is devoted to avoiding the happening of stragglers [5], [6]. However, changeable performance gives birth to the unexpected stragglers. It has been proved that stragglers can exert a key role in affecting the performance. In consequence, it is a significant task to mitigate stragglers, as it can help better distributed cluster's performance. As a number of straggler mitigation techniques are being brought to put to use [7]–[9], speculative execution [10] has been evolved to be the most widely used technique. However, the current straggler mitigation strategies have been applied as imperfect solutions considering that they can merely deal with the symptoms based on the fixed judgment [11], [12]. For the purpose of solving the challenges, we bring forward an adaptive straggler mitigation approach which is able to adjust to the total job varieties, heterogeneous nodes, as well as volatile big data analysis cluster, in the meanwhile of attaining the operator-defined aims. In this paper, We contribute a framework- Hawkeye- to optimizing the performance under heterogeneous conditions in virtue of strengthening the preciseness of straggler identification. The approach takes advantage of below techniques: (i) Identifying stragglers which perceive resource limitations as well as job imbalances, (ii) Adjusting the parameters that harbors close relation with the preciseness of straggler identifying via reinforcement learning, (iii) Strengthening the preciseness of straggler processing in virtue of combining joint speculative execution technique with clone technique. Last but not least, Hawkeye takes data locality into considered, too.

The paper's major contributions are generalized in the following parts:

- We put forward a novel framework, Hawkeye, which takes consideration of both job and node to identify straggler by optimizing speculative execution.
- To the best of our knowledge, this is the first time that reinforcement learning (RL) is employed for identifying the stragglers of accuracy; meanwhile mitigating stragglers off effectiveness with no human instructions exceeding the specified high-level objective like minimizing average job completion time. Furthermore, it is a method with the properties of self-improving and online learning that has little impact on the performance of the cluster.
- We take speculative execution to its extreme and propose launching one clone task at the beginning whose hostname is in a list which has set down the nodes where stragglers frequently appear.
- Hawkeye which is known as a prototype on Spark, is under the evaluation against Spark-Default and Spark-Speculative. According to experimental results on an average, Hawkeye exceeds the solutions by 37% of CPU-intensive applications, with nearly 25% from I/O-intensive applications, 19% from Mixed-intensive applications, 18% from machine learning applications and 23% on accuracy improvements.

The following part presents the rest part of the thesis. Section II illustrates the backdrop of the research. Section III presents our system model. Implementation is revealed in Section IV, and an evaluation of it in Section V. In Section VI, previous undertakings of correlation with our research is under discussion. Finally, Section VII offers the conclusion of the paper.

II. BACKGROUND AND MOTIVATION

This part illustrates our introduction of the speculative mechanism of Spark(Spark-Speculative); more than that, some limitations of Spark-Speculative is also put forward capable part of being promoted. The whole aspects can motivate Hawkeye.

A. SPECULATIVE MECHANISM IN SPARK

The distributed cloud computing frameworks, like Hadoop, Spark, have been billed to be sensitive to heavy tails in response time. The challenge comes out of the reality that execution time of Map-Reduce jobs the execution time of Map-Reduce jobs tends to be affected by a few slow tasks reversely, thus rendering the jobs to be likely to suffer from high latent factors. Those slow tasks, recognized as stragglers, are able to dramatically affect the whole performance of clusters. Spark, which harbors low latency, universality, scalability and highly fault tolerance distributed computing framework, is featured by achievable speculation mode (named Spark-Speculative) to lessen the stragglers. According to this mode, Spark is able to make additional attempts for the jobs after the completion of at least one of the identical task. Regularly, Spark put under calculation the distinction betwixt the execution time of every running task as well as the median completion time successful tasks are calling for. It embarks an additional extra attempted copy for the task with the biggest distinction. Thereby, provided that Spark embarks extra attempted copies for tasks in great number, they will bring about the waste in cluster resources. For another, if Spark embarks additional attempted copies in a small number, some stragglers tend to go on with straggling. Spark [2] provides four parameters to control Spark-Speculative. The parameter ''spark.speculaton'' (False by default) contributes an able default speculative mechanism. Speculative execution for the jobs proves to be a health-check process which checks

tasks for the speculation who is the one run slower at a stage, when comparing the threshold which the median of all that succeeded in completing the tasks multiplying the parameter of "spark.speculation.multiplier" (1.5 by default). Such slow tasks are to be re-submitted to another Work Node. Moreover, instead of going on with the slow task, it will launch a new copy at meantime. In addition, speculation starts with the parameter "spark.speculation.quantile"(0.75 by default) by executing each "spark.speculation.interva" (100ms by default) on a regularly basis. Obviously, Only "spark.speculation.quantile" and "spark.speculation.multiplier" are closely related to mitigate.

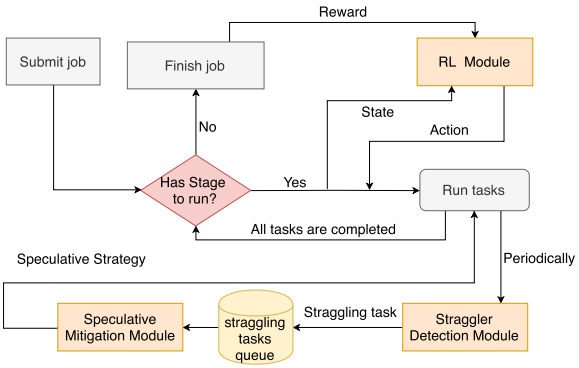

FIGURE 1. Hawkeye workflow.

B. LIMITATIONS OF SPARK SPECULATION

Despite the fact that Spark-Speculative are able to mitigate the straggler from affecting heterogeneous clusters in some ways, the system has to be faced with the following two weaknesses:

- Spark-Speculative is featured by the restricted consideration of task execution speed in the whole job duration; it failed to cover the processing capacity harbored by the different nodes. Hence, it is less accurate to merely compare the execution time to the parameter.
- Speculative execution launches clones of speculative tasks across the cluster and directly adds the duplicates to task queue and submit them. It may lead to superfluous clones because of the imprecision of stragglers identification.

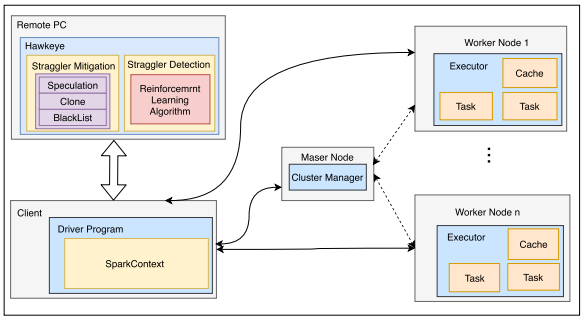

FIGURE 2. The architecture of Hawkeye.

III. HAWKEYE DESIGN

We put forward the identification points where tasks have no capability of making progress at the normal ratio and meanwhile implementing the targeted solutions. The following two items presents the guiding principles distinguishing Hawkeye from the previous mitigation designs:

1) Identifying stragglers in the quickest way to minimize their impact on cluster performance.
2) Improving the preciseness of stragglers identification as much as possible to improve the utilization of cluster resources.
3) Unifying clone, speculative strategies and enable minimize the job completion time.

A. ARCHITECTURE OF HAWKEYE

Figure 1 and 3 portray the process of Hawkeye and system architecture which extends Spark, severally. Our design is structured with three major ingredients.

1) **Straggler Identification Module**: Firstly, collecting the tasks logs out of the worker nodes on a regular basis. Less accurate time can be utilized on the tasks manage to complete for presuming the execution time of the present task. Accordingly, we make the prediction that provided a task would be a straggler in virtue of the expected execution time of the present task, and comparing with all finished tasks at the present stage of the same job severally(as Fig.3 shown).
2) **Reinforcement Learning Module**: There are two parameters "spark.speculation.quantile" and "spark.speculation.multiplier" in all spark configuration properties that are closely related to the identification of stragglers. They have close connection with the preciseness of straggler identification. We employ Reinforcement Learning to pinpoint optimal parameters adaptively, depending on designating a high-degrade objective like minimizing average job completion time.
3) **Speculative Mitigation Module**: We adopt speculative execution to mitigate stragglers. Besides, it is obvious that tasks are more likely to be a straggler in poor performance nodes. Hence, in Hawkeye, we adopt speculative execution to its extreme and propose launching a parallel attempt of tasks at the beginning belonging to a hostname list which records the nodes where stragglers frequently appear by merely employing the result of the one that completes first.

B. PROBLEM FORMULATION

Consider a job that is submitted to a datacenter, where the job consists of N tasks. It is associated with a expected execution time and consists of N independent tasks. Spark-Speculative mitigating stragglers often involve speculative execution, i.e., launching a parallel attempt of each straggler. Let r_i(0 or 1) be the number of speculative attempts that are launched for task i. Thus, the task is completed if one of its $r_i + 1$ attempts (one original attempt plus r_i speculative

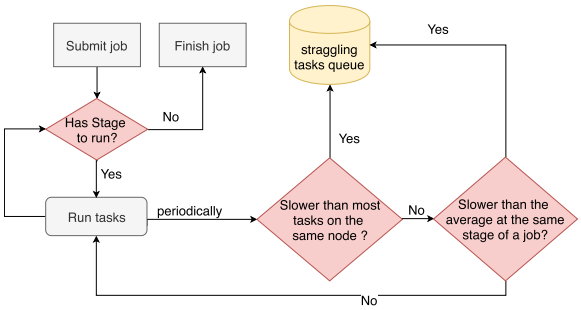

FIGURE 3. Execution process of stragglers identification.

attempts) is finished. We denote the execution time of attempt j of task i as $T_{i,j}$. Thus, task completion time T_i equal:

$$T_i = \min_{j=1,2} T_{i,j}, \quad \forall j \tag{1}$$

To optimize the speculative execution strategies using a model-based approach, Pareto distribution is often selected to model the execution time of tasks in [13], and is used in [8], [14], [15] to analyze the straggler problem. Following these papers, we assume the execution time $T_{i,j}$ of each attempt follows a Pareto distribution with parameters t_{min} and β. The probability density function of $T_{i,j}$ is

$$f_{T_{i,j}}(t) = \begin{cases} \dfrac{\beta \cdot t_{min}^{\beta}}{t^{\beta+1}} & \text{if } t \geqslant t_{min} \\ 0 & \text{if } t < t_{min} \end{cases} \tag{2}$$

where t_{min} is the minimum execution time of the task and β is the exponent, while different attempts are assumed to have independent execution time distributions. We now formally define PoF(the possible of normal completion) under the assumption that tasks execution time is a Pareto distribution, is presented in Theorem 1. We also analyze the expected execution time of a task and present expressions for these in Theorem 2.

Definition: PoF is the probability that a running task completes before the target time T which is the expected execution time of the task.

Theorem 1: The PoF

$$PoF = 1 - \left(\frac{t_{min}}{T}\right)^{\beta} \tag{3}$$

Proof: Firstly, let us denote the probability that a single attempt of a task fails to finish before T by 1 - PoF. Then, we have

$$1 - PoF = \int_{T}^{\infty} \frac{\beta \cdot t_{min}^{\beta}}{t^{\beta+1}} dt = \left(\frac{t_{min}}{T}\right)^{\beta} \tag{4}$$

Theorem 2: T follows the Pareto distribution with parameters t_{min} and β. Then, E(T) can be computed as:

$$E(T) = \frac{t_{min} \cdot \beta}{\beta - 1} \tag{5}$$

Proof: We introduce

$$T - t_{min} = \int_{T}^{t_{min}} dt = \int_{T}^{t_{min}} I(t)dt, \tag{6}$$

where I(t) is the indicator function for event $T > t$:

$$I(t) = I\{T > t\} \overset{def}{=} \begin{cases} 1, & \text{if } T > t \\ 0, & \text{if } T \leq t \end{cases} \tag{7}$$

Taking expectations we have that

$$E(T) = \int_{t_{min}}^{\infty} E(I(t))dt + t_{min} = \int_{t_{min}}^{\infty} P(T > t)dt + t_{min}. \tag{8}$$

Since $P(T > t) = P(T_{j,i} > t)$, we have:

$$E(T) = \int_{t_{min}}^{\infty} (\frac{t_{min}}{t})^{\beta} dt + t_{min} = \frac{t_{min} \cdot \beta}{\beta - 1} \tag{9}$$

IV. IMPLEMENTATION

Having implemented Hawkeye as a pluggable service which parallel data processing platforms (e.g., Spark, or YARN) can communicate with over an RPC interface. We will reveal the implementation details of Hawkeye in this section.

A. STRAGGLER IDENTIFICATION MODULE

Spark performs tasks in three steps:

- TaskDeserialize (TS1): aiming to achieve a deserialized task object and data.
- TaskRun (TS2): aiming to run a task.
- TaskResultSerialization (TS3): aiming to serialize the result.

Going forward, we are going to conduct an elaborate interpretation about stragglers identification, and give the pseudo code presented our Algorithm 1.

We have mentioned that for the reason of the distinct the performance among worker nodes of a heterogeneous cluster, it's improper to forecast the execution time- a task has to be spend only by applying statistical data based on the identical assignment. As we referred to, our advice undertakes to identify stragglers from two distinct perspectives.

1) MUCH SLOWER SPEED COMPARED TO THE EXPECTED TASK EXECUTION TIME UPON THE IDENTICAL WORKER NODE

We gathered the execution information of tasks from every worker node, as well as catching the average execution time of each step as $TS1_{host}$, $TS2_{host}$; $TS3_{host}$. Then we compute the expected execution time of each worker node as ET_{host}.

Specifically speaking, we achieve every step's average execution time in accordance with the designated worker node it runs on. Next, we fix the location of the current step, meanwhile obtaining the executing time at the current step. Afterwards, we calculate *SR*'s value(The rate of straggling). Ultimately, we make the comparison of *SR* with parameter "spark.speculation.multiplier" (*MT*), in order for judging whether or not the task is working as a straggler. The calculation of *SR* is conducted based on the formula as below:

$$SR = \frac{RT + \sum_{i=j}^{3} TSj_{host}}{ET_{host}} \tag{10}$$

where RT is the running time of the task, j is the step id, host refers to the IP address in which the task submitted on, ET_{host} is the expected execution time of the task in regard to the work node.

Algorithm 1 Straggler Identification

Input: *taskInfos*: all tasks information collected; $TS1$: mapping of host and the average deserialization time; $TS2$: mapping of host and the average computing time; $TS3$: mapping of host and the average serialization time; MT: the parameter of speculation_multiplier; $PrOB$: the threshold of probability; SR: straggling ratio; PoF: the probability that a running task completes before DT;

Output: speculatableTasks;

for each $task_i$ in *taskInfos* **do**
  if $task_i$ has not finished **then**
    $SR \leftarrow 0$;
    $TS1_{host} \leftarrow 0$;
    $TS2_{host} \leftarrow 0$;
    if $PoF.get(host) \leq PrOB$ **then**
      $speculatableTasks += task$;
      continue;
    end if
    if *task* has entered TS1 **then**
      $avg\_time_{TS1} = TS1.get(host)$;
      $SR \leftarrow \frac{runtime_i + TS1_{host}}{ET_{host}}$;
    else if *task* has entered TS2 **then**
      $avg\_time_{TS2} = TS2.get(host)$;
      $SR \leftarrow \frac{runtime_i + TS1_{host} + TS2_{host}}{ET_{host}}$;
    else if *task* has entered TS3 **then**
      $avg\_time_{TS3} = TS3.get(host)$;
      $SR = \frac{RT + \sum_{i=1}^{3} TSj_{host}}{ET_{host}}$;
    end if
    if $SR \geq MT$ **then**
      $speculatableTasks += task$;
    end if
  end if
end for

2) SLOWER THAN THE AVERAGE AT THE SAME STAGE OF A JOB

All of that said, it is possible that a task's execution time harbors no significance which is different from the expected execution time of the work node. However, as a result of heterogeneity of the cluster, it falls behind the most of tasks at the stage of a job. Obviously, we are required to consider it. We calculate the probability that tasks complete before the deadline time(DT) and obtain the identification result.

$$DT = MT \cdot AT \tag{11}$$

where AT is the average execution time of the whole finished at the preset tasks at the preset stage of a job. Finding the optimal values of "spark.speculation.quantile" and "spark.speculation.multiplier" in distributed heterogeneous clusters works as complicate and time-costing effort. For the purpose of solving the difficulty, we employ Reinforcement Learning to dynamically choose workload-specific parameters without any human instructions more than designating a higher-degree target like bringing down average job completion time to the least value.

Reinforcement Learning is billed as a kind of machine learning algorithms which contains the learning of the agent on the way of behaving under some circumstances in virtue of the positively and negatively rewards it has obtained. The rewards are not achieved following every action under the condition. However, it attains some interest points. In virtue of multiple iterations, the agent realizes the exact action leading to the particular compensation. At first, it has no idea of the outcome of each action, the agent is required to investigate the environment in order for a better understood projection. Choosing action step from the provided situation is one of the crucial element of reinforcement learning.

B. REINFORCEMENT LEARNING MODULE

1) DETAILED EXPLANATION OF REINFORCEMENT LEARNING ALGORITHM

We use standard policy gradient methods that have recently been applied to multiple domains [16]–[18]. The main idea in policy gradient is to learn by performing gradient descent on the policy parameters using empirically observed rewards. Consider a sequence of interactions with the environment of length T, where the agent collects (state, action, reward) experiences,i.e., (s_t, a_t, r_t), at each step t. It then updates the parameter θ of its policy $\pi_\theta(s_t, a_t)$ (defined as the probability of taking action a_t in state s_t) using the well-known reinforcement algorithm [19].

In reinforcement learning algorithm, we set the state space to the job type, job size, the value of "spark.speculation.quantile" and "spark.speculation.multiplier" (0.75 and 1.5 by default). The reward function and utility value update function are defined as following formulas, respectively:

$$r = \frac{-(t_k - t_{k-1})}{t_k} \tag{12}$$

where t_k and t_{k-1} are the running time of a job.

$$\theta \leftarrow \theta + \alpha \sum_{t=1}^{T} \nabla_\theta log\pi_\theta(s_t, a_t)(\sum_{t'=t}^{T} r_{t'} - b_t) \tag{13}$$

where: α - represents the learning rate and b_t is a baseline used to reduce the variance of the estimated gradient. An example of a baseline is a "time-based" baseline [20], [21], which sets b_t to the cumulative reward from time t onwards, averaged over all training episodes. Intuitively, $\sum_{t'}(r_{t'} - b_t)$ estimates how much better (or worse) the total reward is in a particular episode compared to the average case; $\nabla_\theta log\pi_\theta(s_t, a_t)$ provides a direction to increase the probability of choosing action at at state s_t.

2) TRAINING OF REINFORCEMENT LEARNING ALGORITHM

Training Hawkeye for the cluster creates two challenges. First, system-based reward calculation is not a good fit. For a set of N jobs $Job_1, Job_2, \ldots, Job_N$ sampled from a training set, the standard objective minimizes $E[\sum_{i=1}^{N} T(J_i)]$, where $T(\cdot)$ denotes the completion time of a job. We originally attempted to train using system-based-reward objective. But we found that it does not converge when training a finite set of jobs, as this results in too large of state space. Fortunately, there is an alternative method that calculate reward based on job directly.

In addition, with continuous job arrivals, our real objective is to minimize the average job completion time over a large time horizon, i.e., to minimize $E[\lim_{N\to\infty} \sum_{i=1}^{N} T(J_i)/N]$. Operationally, we replace the standard reward with a differential reward: at every step t, the agent receives the reward r_t - $\hat{r}$, where r_t is the standard reward at time t and $\hat{r}$ is a moving average of the rewards r_t across a large number of previous time steps (across many training iterations). Intuitively, such a reward signal ''normalizes'' the sum completion times of jobs in an RL episode by the length of the episode to counter any bias arising due to varying episode lengths. It is an objective that closely correlates to the average number of concurrent jobs in the system.

The learning process consists of simulating multiple runs of four types of jobs. Each run (or ''episode'') begins with a sequence of jobs, and ends with completion of all jobs. We use standard policy gradient methods that have recently been applied to multiple domains for our reinforcement learning algorithm.

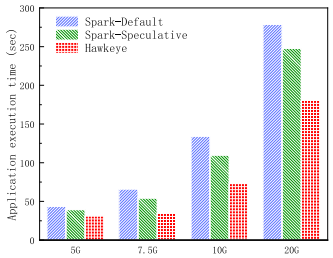

FIGURE 4. The JCT of WC (5GB,7.5GB,10GB,20GB).

C. SPECULATIVE MITIGATION MODULE

Speculative execution has been employed in a large scale for the purpose of maintaining the output correctness as well as enhancing the applications performance. In detail, the speculative execution is characterized by the biggest difficulty in seeking for precise straggler to the fullest. Here are the three major factors which affect the speculative execution to be carefully thought about:

- Straggling rate.
- Execution step of a task. It is of vital importance to take into consideration the present execution step for a task to realize the effective replication creation. The moment a task manages to make the entry into the late step, commit earlier completeness provides a good chance compared to the duplication.
- The performance of nodes which tasks running on. Obviously, tasks are more likely to be a straggler in poor performance nodes.

With straggler identification preciseness upgrades dramatically, the earlier are the resources which stragglers consume issued, the more degree will be the utilization ratio of the cluster. Thus, it is of importance to look carefully at the present execution step delivered for a task for the efficient attempt generation. An attempt ought to be composed only it is possible to bring it to the end in priority to the aboriginal task. Accordingly, we apply formula (14) for further reducing the likeliness of attempts failure.

$$FP = (1 - \alpha) * SR + \frac{S - 1}{3} * \alpha \tag{14}$$

where SR stands for the task's straggling rate, $S \in \{1, 2, 3\}$ is which step it is in.

Besides, we have a list in record of the nodes where stragglers frequently appear. We take speculative execution to its extreme and propose launching a parallel attempt at the beginning of tasks whose hostname are in the list.

Last but not least, having achieved straggling tasks, we will conduct speculative tasks on the idle nodes. It is most crucial for the speculative tasks to guarantee that the duplication to be completed previous to the origin one. It is proved to be an practical option to neglect the nodes which have high straggling ratio [22]. Hence, We add the Delay Scheduler [23] to the system for furthering straggler mitigation.

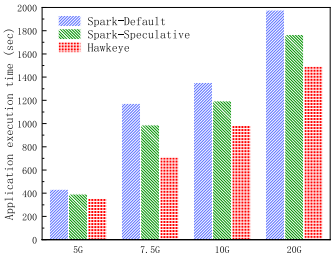

FIGURE 5. The JCT of Sort(5GB,7.5GB,10GB,20GB).

V. EVALUATION

We implemented Hawkeye in Apache Spark and spent approximately 2 days training on it. In this section, the performance of Hawkeye is put under evaluation on a heterogeneous spark cluster in virtue of having it compared with Spark-Default, Spark-Speculative in terms of job completion time as well as the preciseness stragglers identification.

A. EXPERIMENTAL ENVIRONMENT

For the purpose of analyzing Hawkeye's performance, we deploy it on a real environment. Our ten-servers-based experimental clusters are made up of a master and nine workers. It is comprised of three categories in accordance with the node performance. Table 1 shows the further information on the whole nodes. Next, the operation system of each node

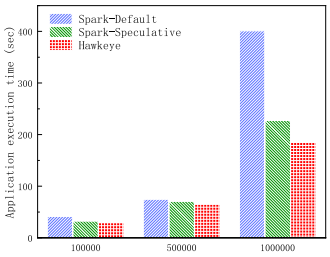

FIGURE 6. The JCT of PageRank (1E5,5E5,1E6 pages).

TABLE 1. Detailed information of each node.

NodeId	Memory	Core processors	Disk	Node type
Master	4GB	4	1TB	-
Worker 1	4GB	4	500GB	A
Worker 2	4GB	4	500GB	A
Worker 3	4GB	4	500GB	A
Worker 4	4GB	4	500GB	A
Worker 5	4GB	4	500GB	A
Worker 6	4GB	2	200GB	B
Worker 7	4GB	2	200GB	B
Worker 8	2GB	4	500GB	C
Worker 9	2GB	4	500GB	C

separately refers to Ubuntu 14.04.3 LTS, Java version for 1.8.0_191, and Scala version for 2.11.4. The whole ten nodes are located at the identical Local Area Network. Further, the Hadoop and Spark version of the experiment individually refers to Apache Hadoop 2.7.2 and Apache Spark 2.2.0. Standalone is chosen to be the cluster manager. Hadoop and Spark configuration are both reserved as default configuration. To evaluate the performance of Hawkeye, we use WordCount, Sort, PageRank and Kmeans which respectively represent CPU-intensive, I/O-intensive, Hybrid-intensive and machine learning applications. All of them are applied as common applications used in Spark for evaluating performance. We use Hibench [24], which is a big data benchmark suite, which aims to provide these applications with workload and help evaluate different applications in terms of speed and the throughput.

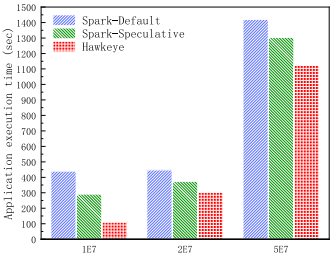

FIGURE 7. The JCT of Kmeans(1E7,2E7,5E7 samples).

B. METRICS FOR THE MEASUREMENT

We make comparison on Hawkeye with Spark-Default, Spark-Speculative in our cluster, aiming to express the progress by employing the job completion time as well as the accuracy of stragglers identification performing the duty of measurement metrics. The two metrics of the measurement are explained in the part below:

1) Job completion time: the average elapsed time of every job, to evaluate the public presentation of Spark.
2) Accuracy of stragglers identification: we assign the location of tasks which are marked as stragglers and calculate the precision rate.

Furthermore, to construct the computing capability variances among each of the nodes, background applications are run in the host OS of the nodes. These execution computation tasks from combination of the optional times with the inject background noise. The assessment approach is rationally revealed as the Eq.(15). It stands for the distinction betwixt Hawkeye(HK) and Spark-Speculative(SS) isolated by Spark-Speculative.

$$Improvement\ Rate = \frac{SS - HK}{SS} \tag{15}$$

C. PERFORMANCE EVALUATION OF REINFORCEMENT LEARNING

Speculative execution, the creator of task replicas at runtime, is touted as a representative approach which is planned in grandly distributed systems for tolerating stragglers. The project interprets stragglers based on the specified static parameters in order to summarize the distinction betwixt the execution time of running task and the median completion time for all of the successful tasks. Nevertheless, specifying static parameters deteriorate the speculation efficiency since it cannot take into consideration the natural diversity of job timing constraints in the restricted modern day cloud computing systems. Furthermore, parameters also cannot cover system environmental limitation in light of replication overheads as well as optimal system resource usage. Therefore, Bayesian Optimization algorithm and Reinforcement Learning are used to compare the results of parameters tuning.

We demonstrate the effectiveness of our adaptive parameters tuning in systems tolerating straggler in two aspects:

1) Analysis on performance.
2) Analysis on parameter sensitivity.

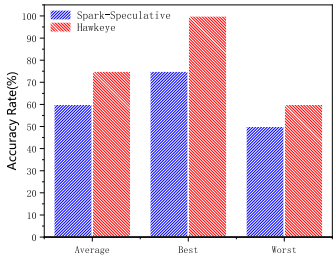

FIGURE 8. The Accuracy of WordCount(5GB).

Part One explores job timing improvement in virtue of primary-oriented the job response time as well as replica number. Part Two focuses on algorithms effectiveness in distinct types and sizes of jobs.

Our experiment demonstrates the following: The adaptive parameters tuning can improve job completion and decrease timing failure occurrence. As shown in Table 2-4, the Reinforcement Learning can reduce job completion time on

TABLE 2. Cluster workload factor experiments results(WordCount 3.2 GB).

Workload ID	Bayesian Optimization		Performance(JCT)			RL	Performance(JCT)		
	multiplier	quantile	Worst Case	Best Case	Average	Adaptive tuning	Worst Case	Best Case	Average
Workload 1	1.3	0.5	58.9	34.3	40.2	\	53.5	35.5	44.8
Workload 2	1.4	0.5	61.7	52.7	56.8		59.8	46.4	54.3
Workload 3	1.2	0.6	66.3	59.4	62.1		67.0	43.8	58.9
Workload 4	1.7	0.5	81.4	57.1	68.5		76.8	58.6	67.7

TABLE 3. Job type factor experiments results.

JOB TYPE	Bayesian Optimization		Performance(JCT)			RL	Performance(JCT)		
	multiplier	quantile	Worst Case	Best Case	Average	Adaptive tuning	Worst Case	Best Case	Average
WordCount(1.6 GB)	1.5	0.7	36.1	28.9	32.7	\	38.4	27.6	30.6
Sort(1.6 GB)	1.4	0.5	124.3	82.0	114.8	\	127.4	80.4	112.4

TABLE 4. Job size factor experiments results.

JOB SIZE	Bayesian Optimization		Performance(JCT)			RL	Performance(JCT)		
	multiplier	quantile	Worst Case	Best Case	Average	Adaptive tuning	Worst Case	Best Case	Average
1.6GB	1.5	0.7	36.1	28.9	31.7	\	38.4	24	30.6
3.2GB	1.4	0.5	61.7	52.7	56.8		59.8	46.4	54.3
4.8GB	1.2	0.6	74.9	54.6	65.3		75.5	49.3	58.6
6.4GB	1.6	0.6	95.2	59.1	75.3		95.6	53.4	70.7

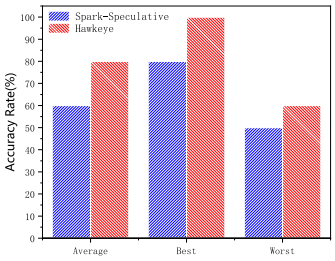

FIGURE 9. The Accuracy of WordCount(7.5GB).

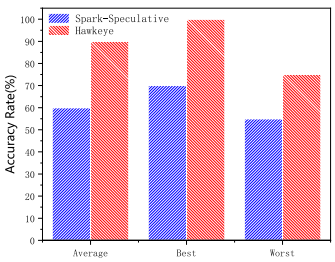

FIGURE 10. The Accuracy of Sort(5GB).

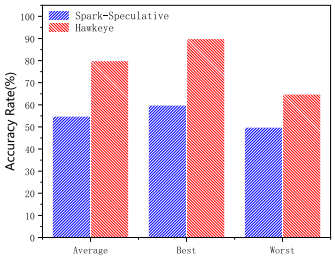

FIGURE 11. The Accuracy of Sort(7.5GB).

average across all experiment cases compared to Bayesian Optimization algorithm that chooses static optimal para meters. Table 2 shows that the values of optimal parameters vary under different workloads which we run different types and sizes of background applications. It is obvious that reinforcement learning can choose optimal parameters adaptively according to the workload of the cluster. Table 3 and Table 4 show that the values of the optimal parameters are different for different job sizes and types. In general, The use of reinforcement learning in this scenario is interesting and effective.

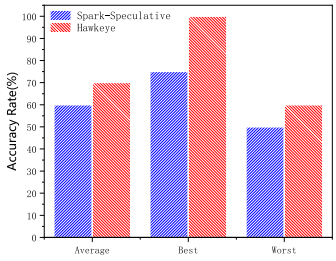

FIGURE 12. The Accuracy of PageRank(1E5 pages).

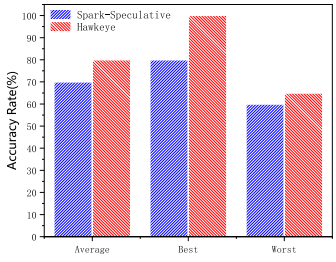

FIGURE 13. The Accuracy of PageRank(5E5 pages).

D. PERFORMANCE EVALUATION IN HETEROGENEOUS ENVIRONMENT

1) JOB COMPLETION TIME

Aiming to validate the efficiency of Hawkeye, we ran the jobs 20 times in the circumstances of several Spark-Default, Spark-Speculative and Hawkeye. And then, we gathered each job's information along with the calculated average consuming time. Fig. [4 - 7] show our terminal results. 4 explains that compared to Spark-Speculative, Hawkeye was

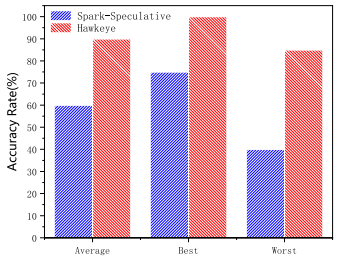

FIGURE 14. The Accuracy of Kmeans(1E7 samples).

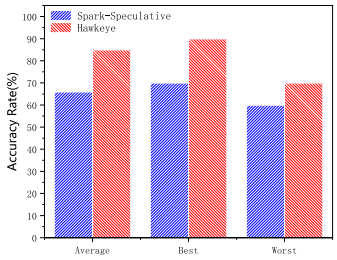

FIGURE 15. The Accuracy of Kmeans(2E7 samples).

able to WordCount applications by 37% faster and compared to Spark-Default by 48% faster on the average, on the prerequisite that 10GB as the input size. When the input size reaches 10 GB, on an average, Hawkeye was by 33% faster compared to Spark-Speculative and by 45.5% faster compared to Spark-Default. Moreover, 5 displays the completion time of Sort Applications on the 20 times. Despite that the completion time of all methods presents irregular fluctuation, the performance of Hawkeye conducts evidently better performance compared to the time of Spark-Default and Spark-Speculative. Fig. 6 shows that Hawkeye finishes PageRank applications at a faster rate by 18.5% in contrast of Spark-Speculative and a faster rate of 53.9% compared to Spark-Default on average when the input size is 1,000,000 pages. Furthermore, as the input size reaches 5,000,000 pages, Hawkeye on average at the rate of 9.5% compared to Spark-Speculative and 16.9% faster than Spark-Default. Similarly, from Figure. 7, it is obvious that Hawkeye gives better performance compared to the one of Spark-Default and Spark-Speculative.

On the whole, it is an undenied reality that Hawkeye improvement is of great significance.

2) THE PRECISENESS OF STRAGGLER IDENTIFICATION

Provided that speculative tasks terminates earlier than straggler tasks, we titled the speculation a success, and a failure otherwise. Hawkeye can reduce job completion time more effectively and steady than Spark-Speculative. In order to evaluate Hawkeye could lessen job completion time in a more effective and steady way compared to Spark-Speculative. Aiming to give Hawkeye a more accurate and effective assessment, we made comparison on the preciseness of Spark-Speculative. Specifically, conducted 20 times experiment on Spark-Speculative and Hawkeye. Thus, calculating the formula-based results that successful handling tasks number is separated by marked numbers as straggler tasks. As [8 - 15] show, compared to Spark-Speculative from all cases, Hawkeye expressed higher identification preciseness. Therefore, evidently, with nearly preserved at 85%, the preciseness rate was upgraded around 23% at least in regards to Spark-Speculative.

In accordance with the experimental results, Hawkeye manages to reduce the job completion time on the different type application in the meanwhile of attaining achieving superior performance and enhancing the preciseness of stragglers identification dramatically.

VI. RELATED WORK

According to the previous work, stragglers is capable of running as many as 8 times slower compared to the median task [3], [11], [25]. They are able to affect dramatically the whole conduct process performed by cloud applications. Stragglers are inevitable in big data analysis system due to a number of reasons. Firstly, cluster nodes are typically comprise commodity ingredients and that can be of diversity by nature, thus causing many kinds of nodes to conduct in different way. Secondly, the reality that all kinds of sources of hardware/software errors are present in cluster can give rise to node failures which interrupts task execution upon the failed nodes. Last but not the least, along with source sharing in the meanwhile, co-scheduled tasks running operated on the same physical machines tend to result in resource contention as well as network congestion. It exemplifies contribution significantly to the happenings of the stragglers.

Straggling task brings about longer service response times as well as lessening cluster utilization. The distributed systems have to be commonly faced with these problems, the promoted methods in other such distributed systems as Map-Reduce contributed instructions and guidance to the research on the mitigation of the stragglers, influence on Spark. The last decade has seen great numbers of studies on improving to stragglers identification [26]–[29]. For example, Dean and Ghemawat [25] proffered a straggler identification approach on the basis of progress score, which is explained a $0 - to - 1$ number and processed data rate for the total task input data. Additionally, that has been exhibited to bring out the dramatic performance promotion. Zaharia *et al.* [26] announced that the progress score could not precisely tell the way in which a task is operated as distinct tasks on the specific occasion. Thereby, those researches put forward a new scheme titled as Longest Approximate Time to End(LATE). It took into consideration not only the progress score but also the running time. Nonetheless, several reasons are also existing, which conduces to less accurate s tragglers identification. A typical instance is that reducing tasks of the same task might take on distinct input data sizes. Accordingly, it is required that a task based on larger input data size should have more time for the execution in comparison to another jobs. Ananthanarayanan *et al.* [3] revealed a cause-aware mechanism, Mantri, which recognizes stragglers in virtue of the anticipated execution task time($ET_{expected}$). The $ET_{expected}$

of a task belongs to the sum of the task's elapsed time as well as the estimated time remaining. To apply speculative execution as soon as possible, Dolly [11] adopts an active way to handle stragglers. For each task, Dolly launches clones at the beginning of the job. Apparently, there is no need to clone all tasks. Thus, Dolly causes a serious waste of resources.

As a result of the Spark characteristics, these algorithms fail to be applied to it in a direct way. However, they are still meaningful for Spark.

In addition, there are some researches expressing concern on how to avoid stragglers. Avoidance-oriented with approaches have been occurring during the procedure of task scheduling phase [30], [31]. The planner might try not to take scheduling tasks for the known poor nodes in virtue of conducting blacklisting techniques [30], [32], [33]. In the procedure, the tasks are never to be distributed to these nodes before the removal of the list. These methods are primarily performing as an optimized duty for the average stragglers which teams up with the speculative execution design and that helps forecast node performance by avoiding distributing tasks/speculations for the slow servers. The methods features the effectiveness which relies upon properly detecting faulty nodes to blacklist, or the system capacity is to be worsened as a result of the false positives.

The research focuses on identifying a larger number of stragglers, therefore, shorten the long-term execution tail brought about by stragglers. Nevertheless, a small number of these researches regard the preciseness of stragglers identification [34]–[36]. Meanwhile, there is little prior work on applying reinforcement learning techniques [16], [37] to straggler mitigation. Reinforcement learning represents a class of machine learning algorithms in which the agent learns how to behave in a environment through the positive and negative rewards that it obtains. Despite that the runtime stragglers are ever recognized by the previous work, we prove to bring forward in virtue of reinforcement learning before another in time to promote the preciseness of stragglers identification. Hawkeye can help reduce job completion time than prior methods that duplicate tasks to the ending of a phase.

VII. CONCLUSION

To sum up, this paper includes our introduction on Hawkeye, the first adaptive straggler identification framework on Spark cluster with reinforcement learning. The novel framework provides an adaptive straggler identification model by embedding the reinforcement learning to search for the optimal parameters at runtime. Using techniques to further address the efficiency and accuracy straggler issues without further jobs information. Our investigation deploys Hawkeye on a heterogeneous Spark cluster. The experimental results show Hawkeye can effectively identified that straggler on the average are exceeding the present solutions at the rate of 37% from CPU-intensive applications, with up to 25% from I/O-intensive applications, 19% from Mixed-intensive applications, coupled with 18% from machine learning applications and 23% on accuracy improvements.

With speculative execution being a common approach for straggler mitigation, scheduler is encountered with a decision on extra attempts versus the original tasks. For future work, we will concentrate on scheduling decision which is also a chance to make dramatic improved performance.

ACKNOWLEDGMENT

The authors would like to thank Q. Xiang and F. Le who help to review our earlier model and helped with the design of a new solution. This research was supported in part by NSFC 61672385.

REFERENCES

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. Usenix Conf. Hot Topics Cloud Comput.*, 2010, pp. 1–7.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.* Berkeley, CA, USA: USENIX Association, 2012, p. 2.

[3] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using Mantri," in *Proc. OSDI*, Oct. 2010, vol. 10, no. 1, p. 24.

[4] S. I. Resnick, A.-L. Fougères, and P. Soulier, "Heavy-tail phenomena: Probabilistic and statistical modeling," *SIAM Rev.*, vol. 50, no. 2, pp. 391–393, 2008.

[5] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *Proc. 12th USENI Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 99–115.

[6] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Phoenix: A constraint-aware scheduler for heterogeneous datacenters," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 977–987.

[7] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Maestro: Replica-aware map scheduling for MapReduce," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2012, pp. 435–442.

[8] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, Nov. 2015.

[9] D. Cheng, C. Jiang, and X. Zhou, "Heterogeneity-aware workload placement and migration in distributed sustainable datacenters," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 307–316.

[10] P. Zheng and B. C. Lee, "Hound: Causal learning for datacenter-scale straggler diagnosis," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 1, pp. 1–36, Apr. 2018.

[11] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 185–198.

[12] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating skew in MapReduce applications," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2012, pp. 25–36.

[13] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "GRASS: Trimming stragglers in approximation analytics," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 289–302.

[14] H. Xu and W. C. Lau, "Optimization for speculative execution in big data processing clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 530–545, May 2016.

[15] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, 2015, vol. 45, no. 4, pp. 379–392.

[16] H. Mao, M. Schwarzkopf, S. Bojja Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," 2018, *arXiv:1810.01963*. [Online]. Available: http://arxiv.org/abs/1810.01963

[17] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–12, Jul. 2016.
[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.
[20] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1471–1530, Nov. 2004.
[21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.
[22] S. Deshmukh, J. Aghav, K. T. Rao, and B. T. Rao, "Avoiding slow running nodes in distributed systems," in *Computer Communication, Networking and Internet Security*. Springer, 2017, pp. 411–420.
[23] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
[24] S. Huang, H. Jie, J. Dai, and X. Tao, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Proc. IEEE Int. Conf. Data Eng. Workshops*, Mar. 2010, pp. 41–51.
[25] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[26] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. OSDI*, 2008, vol. 8, no. 4, p. 7.
[27] C.-Y. Lin, T.-H. Chen, and Y.-N. Cheng, "On improving fault tolerance for heterogeneous Hadoop MapReduce clusters," in *Proc. Int. Conf. Cloud Comput. Big Data*, Dec. 2013, pp. 38–43.
[28] H. Xu and W. Cheong Lau, "Resource optimization for speculative execution in a MapReduce cluster," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, pp. 1–3.
[29] H. Xu and W. C. Lau, "Task-cloning algorithms in a MapReduce cluster with competitive performance bounds," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jun. 2015, pp. 339–348.
[30] N. J. Yadwadkar and W. Choi, "Proactive straggler avoidance using machine learning," Univ. California, Berkeley, Berkeley, CA, USA, White Paper, 2012.
[31] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding long tails in the cloud," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 329–341.
[32] X. Ouyang, P. Garraghan, C. Wang, P. Townend, and J. Xu, "An approach for modeling and ranking node-level stragglers in cloud datacenters," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 673–680.
[33] X. Ouyang, C. Wang, and J. Xu, "Mitigating stragglers to avoid QoS violation for time-critical applications through dynamic server blacklisting," *Future Gener. Comput. Syst.*, vol. 101, pp. 831–842, Dec. 2019.
[34] H. Yang, X. Liu, S. Chen, Z. Lei, H. Du, and C. Zhu, "Improving spark performance with MPTE in heterogeneous environments," in *Proc. Int. Conf. Audio, Lang. Image Process. (ICALIP)*, Jul. 2016, pp. 28–33.
[35] P. Zhang and Z. Guo, "An improved speculative strategy for heterogeneous spark cluster," in *Proc. MATEC Web Conf.*, vol. 173, 2018, p. 01018.
[36] X. Ouyang, P. Garraghan, D. McKee, P. Townend, and J. Xu, "Straggler detection in parallel computing systems through dynamic threshold calculation," in *Proc. IEEE 30th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2016, pp. 414–421.
[37] H. Du, S. Zhang, P. Han, K. Zhang, and B. Xu, "Cheetah: A dynamic performance optimization approach on heterogeneous big data analytics cluster," in *Proc. 5th Int. Conf. Big Data Comput. Commun. (BIGCOM)*, Aug. 2019, pp. 169–177.

HAIZHOU DU received the Ph.D. degree in computer science and technology from Tongji University, Shanghai, China. His research interests include machine learning, data management, and distributed systems.

SHAOHUA ZHANG (Member, IEEE) received the B.S. degree in computer science and technology from the Shanghai University of Electric Power, Shanghai, China, in 2016, where he is currently pursuing the M.S. degree in electric power information technology with the School of Computer Science and Technology.

His research interests include fault tolerance and optimal scheduling for distributed computing systems, cloud computing, and big data computing.