Workload-Aware Scheduling for Heterogeneous-Storage Data Analytics

name1, name2, name3, and name4
State Key Lab. for Novel Software Technology, Nanjing University, CN
Email: email address, {email address}@nju.edu.cn

***Abstract*—A trend in nowadays data centers is that equipped with SSD, HDD, etc., heterogeneous-storage devices are widely deployed to meet different demands of various big data workloads. Since the read speeds of diverse storage devices are quite different, the unbalanced use on either specific type of storage or particular device would easily elongate the latency on reading data, worsening the overall performance for analytical tasks. In this paper, we formulate Workload-Aware Scheduling problem for Heterogeneous-storage (WASH), as well as show its NP-hardness. To solve this proposed problem, we design a randomized algorithm (WASH-rdm) which chooses source devices based on delicate calculated probabilities and can be proved concentrated on its optimum with high probability through our theoretical analysis. Extensive experiments show that the WASH-rdm can reduce the reading time for tasks by up to 55% over the state-of-the-art baseline algorithm.**

***Index Terms*—big data analytics, heterogeneous-storage, workload-aware scheduling**

I. Introduction

Nowadays, there are new trend that data centers have a variety of data analytical workloads, e.g., some data streaming applications like Storm [36] and other machine learning based applications like Grep [23]. The different workloads might have different requirements on storage performance [24] [25] [26] [27]. For example, Storm is a compute-intensive application whose computation relies significantly on the I/O performance while Grep is a I/O-intensive application which has a great throughput on I/O. In order to meet these demands, a lot of heterogeneous devices have been deployed in big data analytics frameworks, e.g., Hadoop [14] and Spark [15].

However, the heterogeneity of these devices would lead to the divergent read performance due to two aspects. Firstly, the different types of storage, e.g. SSD [28] and HDD [29], have different read performance, naturally. For example, as shown in Fig.1 reading the same amount of data from HDD takes almost twice longer than SSD. Secondly, when there are a large number of tasks reading data concurrently from specific device, read performance is also affected. As further shown in Fig.1, when the amount of tasks increases, the reading time for each task will also increase.

Due to the different read performance, traditional scheduling may result in unbalanced usage on storage device. For example, when tasks fetch data from those disks with high-performance, they would spend less time on I/O, compared with those low-performance disk. Furthermore, the I/O queries containing too many data analytical tasks may easily overload the storage devices. As a result, disks with poor read performance or heavy workload would elongate the I/O performance of analytical tasks.

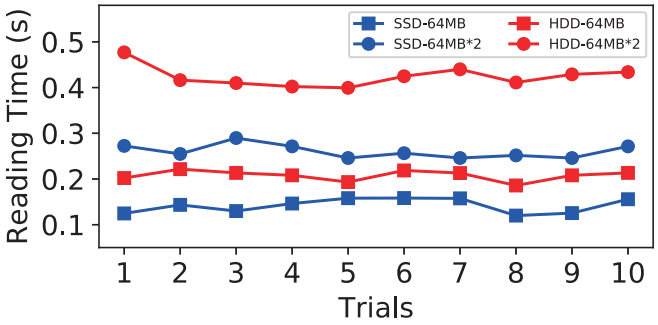


Fig. 1. Comparisons of (1) reading time for one piece of data with unified size, i.e., 64MB, on various storage. (2) reading time for various I/O workloads.

Thus, we propose to focus on dealing with unbalanced use of heterogeneous-storage devices in this paper. However, due to the multiple replicas of data, how to choose these feasible locations is a big challenge. Most of the researches focus on heterogeneous computing resources [21] [22] [31] [32], ignoring the heterogeneous-storage devices. Although some works have already considered heterogeneous-storage [6] [7], multiple types of storage are not considered. In contrast, we formulate the Workload-Aware Scheduling problem for Heterogeneous-storage as well as prove its NP-hardness. Afterwards, we propose a randomized algorithm (WASH-rdm) with guaranteed performance by carefully choosing source device based on delicate calculated probabilities. More concretely, our contributions are as follows:

- To balance the workload on heterogeneous-storage device, we propose Workload-Aware Scheduling problem for Heterogeneous-storage (WASH) which is proved to be NP-hard.
- We propose randomized algorithm with guaranteed-performance. The WASH-rdm algorithm can be proved concentrated around its optimum with high probability 1 - O(e^{-t^2}) where t is the concentration bound.
- The results of our extensive simulations show that the performance of our proposed WASH-rdm improves by up to 55% over the art-of-the-state algorithms.

The rest of the paper is organized as follows. In Section II,

we present the related works of the WASH problem. Then we propose our system model and our algorithm in Section III and Section IV. At last, we conclude this paper in Section VI by summarizing our main contributions.

II. RELATED WORKS

The related works consist of two parts. For the first part, due to limited link bandwidth, some works focus on data locality to improve data analytics performance. However, in heterogeneous environments, it is not enough to consider data locality alone. Thus, some other studies have been carried out to accelerate data analytical tasks by studying hardware heterogeneity [1].

Due to the limited network bandwidth resources in data center, while a large amount of data is transmitted between nodes before task execution, it will greatly affect the data analytics. Therefore, processing data locally as much as possible can improve performance, e.g., placing map task to the node which stores the input data. Matei et al. [2] proposes delay scheduling to assure data locality. Delay scheduling considers that when a node has idle slots, priority should be given to the tasks whose input data is stored at that node, and delay to schedule tasks whose related data is not stored at that nodes. Ganesh [3] makes multiple replicas for high-frequency data by analyzing the access frequency of data, improving the data locality in this way. Cristina [4] proposes a distributed adaptive data replication algorithm DARE, which can help the scheduler show better data locality. Jalaparti [5] believes that most of the production work is repetitive and predictable, by which the scheduler could make scheduling plan ahead. Coordinating the tasks and data placement of these jobs can effectively reduce bandwidth usage, further, improving data locality. All of these work improves the performance of data analytics by improving data locality.

However, in heterogeneous data centers, due to the different performance of data storage devices, task reading time is often different. At present, some of the existing work is based on the research of storage heterogeneity, which accelerates the speed of data processing. Xu et [6] considers the current usability of underlying hardware (CPU, memory, I/O), but does not consider the performance differences of different storage hardware. In Hadoop 2.8, HDFS [19] considers heterogeneous-storage and supports multiple storage strategies. One of the storage strategies is ONE_SSD, that is, one replicas is stored in SSD, and the rest is stored in HDD. However, the task scheduling strategy is not aware of the disk's performance. Based on ONE_SSD, Pan [7] proposes H-Scheduler which takes into account the performance between HDD and SSD. Facing the trend of increasing heterogeneous-storage, it is one-sided to divide the storage devices into two categories. Wang B [8] uses Markov model to describe the use of nodes in the cluster to deploy data reasonably. However, the scheduling problem of tasks is not considered, and the heterogeneity of different storage media is also not considered. These work does not accurately define the differences between storage hardwares, and there is still a situation where a large number of tasks read low-speed devices, causing bottlenecks.

The difference between those tasks and ours is that our work specificly defines the difference in reading speed of disk. Then, the scheduler deploys tasks according to different reading speed of disk to avoid bottlenecks.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first introduce the background of heterogeneous-storage used on data analytics and then build the system model. After that, we formulate the Workload-Aware Scheduling problem and prove its NP-hardness.

A. Background and motivation

In data center, Hadoop and Spark wildly use heterogeneous-storage to support different workloads. For example, in MapReduce framework, Shuffle [38] [37] stage usually has a large requirement on I/O . In order to meet this demand, Crail [33], which is a high-performance I/O architecture for data processing, use NVMeSSD as supplementary storage to speed up Shuffle.

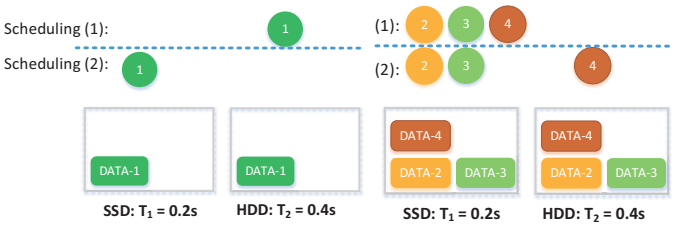
Data and replicas. In data analytics system, large volume of data is generated, such as GPS information [34], system logs [35], etc. Due to their massive sizes, a large file is often divided into multiple fixed-size pieces, i.e., one $data$ block, stored in distributed file system such as HDFS [19], whose uniform size is often 64MB or 128MB. However, the disk is usually unreliability, e.g., about 5 to 6 disks are damaged per day in a cluster of 5,000 machines [28]. In order to avoid data loss, traditional approach is to keep multiple $replicas$ of one data in storage, such as three backups across two racks. Then, one data will be stored in multiple disks with various I/O performance.

Job and tasks. A data analytics workload, named a job, includes a lot of parallel $tasks$. Since each task must read its related input data before execution from the corresponding disk, the scheduler in data analytics system needs to decide the fetching source for each task among multiple replicas. Note that the completion of a job depends on the straggler task.

However, the traditional schedulers in data analytics is usually unaware of disks' types and workload, which often lead to straggler task. In contrast, our scheduler is then designed to avoid the straggler task by balancing workload of heterogeneous devices.

Motivation and example. We will use an example to illustrate importances on choosing heterogeneous-storage. As shown in Fig.2, there are two disks, i.e., SSD and HDD. The reading time of a replica from SSD and HDD is T_1 = 0.2 and T_2 = 0.4, respectively.

- A traditional scheduler may place tasks such as scheduling 1 of Fig.2(a), and the reading time of scheduling 1 is $0.4s$ while the reading time of delicate method, i.e., scheduling 2, is $0.2s$. Obviously, the tasks that read data from SSD has a shorter reading time than those reading from HDD, i.e., scheduling 2 is better than scheduling 1.



(1) = $max\{0,\ 0.4s\}$,
(2) = $max\{0.2s,\ 0\}$.
(a) Type of storage.

(1) = $max\{0.2s * 3,\ 0\}$,
(2) = $max\{0.2s * 2,\ 0.4s\}$.
(b) Load of storage.

Fig. 2. Two aspects that affect the task reading time: (a) type of storage, (b) Load of storage. Delicate method, i.e., Scheduling 2, improves 50%, 33% over scheduling 1, respectively.

- In Fig.2(b), the reading time of scheduling 1 is $0.2s*3 = 0.6s$ while the reading time of the delicate method, such as scheduling 2, is $max\{0.2s * 2, 0.4s\} = 0.4s$. Apparently, tasks that read from lower workload storage have a shorter reading time, i.e., scheduling 2 is also better than scheduling 1.

This simple case reveals two important finding: (1) the types of heterogeneous-storage affect task reading time. (2) the workloads of storage also have effects on task reading time. In order to shorten the task reading time, we should take both of the these two findings into consideration.

B. System Model

The major notations used are shown in table I. The data center is equipped with heterogeneous disks, and we denote by $\mathcal{D}$ the heterogeneous disks set, $\mathcal{D}$ = {1, 2, ...,$|\mathcal{D}|$}. Each disk-i corresponds to a different reading time for one task, denoted by T_i, $i \in \mathcal{D}$. $\mathcal{M}$ is defined as the set of data in data center, $\mathcal{M}$ = {1, 2, ..., $|\mathcal{M}|$}. Each data-l has C replicas whose size τ, e.g., 64MB or 128MB, stored on C different disks. π_l^i is a binary variable indicating whether data-l is stored in disk-i or not. More concretely, we define π_l^i as

$$\pi_l^i = \begin{cases} 1, & \text{if the disk-i stores a replica of the data-l,} \\ 0, & \text{otherwise,} \end{cases}$$

When the job arrives, the job will be divided into parallel tasks, denoted by $\mathcal{T}$ = {1, 2, ..., $|\mathcal{T}|$ }. Each task-j corresponds to one data ϕ_j, where ϕ_j represents the data that task-j need to read. Since there are C replicas for each data deployed in C disks, task-j can read ϕ_j among the C disks, denoted by $\{d_1^j, d_2^j, ... d_C^j\}$.

I_i^j is a decision variable indicating whether task-j choose to read the replica stored in disk-i or not. Specificly, we define π_l^i as

$$I_i^j = \begin{cases} 1, & \text{if the task-j chooses to read the replica} \\ & \text{stored in disk-i,} \\ 0, & \text{otherwise,} \end{cases}$$

Then, we denote by $load_i$ the reading time of all tasks for the disk-i, $load_i = \sum_j I_i^j * T_i$. In order to balance the use of heterogeneous and speed up the reading time of data analytical tasks, our objective is to minimize the maximum disk's $load$.

TABLE I
THE MAJOR NOTATIONS USED IN THIS PAPER.

Variable	Description
T_i	The time of reading a replica from disk-i
ϕ_j	The index of data that task-j needs, $\phi_j \in \mathcal{M}$
d_r^j	The index of disk which stores data ϕ_j's r-th replica
π_l^i	A binary variable indicating if data m_l are stored at disk-i or not
ϕ_j	The data that task-j needs
N_i	Number of tasks which reads data from disk-i
τ	Size of each data replica in data center
C	The number of replicas for each data.
Decision variable	**Description**
I_i^j	A binary variable indicating if task t_i choose to read the replica stored in d_i or not

C. Workload-Aware Scheduling problem for Heterogeneous-storage (WASH)

A large number of tasks are running in the cluster of data centers. If the schedulers are unaware of the heterogeneity of disks, it is easily causing the unbalanced use on disks, resulting in elongating the latency on analytical tasks. To avoid such situation, we propose Workload-Aware Scheduling problem for Heterogeneous-storage (WASH), as shown below. The optimization goal is to minimize the maximum disk load. Finally, by determining the value of the decision variable I_i^j, the appropriate disk is selected for each task to read the corresponding data. Detailed description is as follows:

$$Min: \quad \max_i\{\sum_j I_i^j * T_i\} \qquad \text{[WASH]}$$

$$s.t. \quad \sum_i I_i^j = 1, \ \ \forall j, \tag{1}$$

$$I_i^j \le \pi_{\phi_j}^i, \ \ \forall i, j, \tag{2}$$

$$I_i^j \in \{0, 1\}, \ \ \forall i, j. \tag{3}$$

Constraint (1) guarantees that task-j can only fetch data ϕ_j from one source disk. Constraint (2) guarantees that task-j can only select from those disks containing its input data. If one replica of data ϕ_j is stored in disk-i, then $\pi_{\phi_j}^i$ = 1, otherwise $\pi_{\phi_j}^i$ = 0. Constraint 3 denotes the domain of decision variables, which can only be 0 or 1. I_i^j = 1 indicates that task-j selects the replica stored on disk-i as input, while I_i^j = 0 is on the contrary.The key to solve WASH problem is to determine the value of decision variables I_i^j.

Next, we show the NP-hardness of this problem. WASH problem can be reduced from the integer linear programming (ILP) problem. The ILP problem is NP-hard, and it follows that WASH problem is NP-hard. The specific proof is as follows.

Theorem 1: The WASH problem is NP-hard.

Proof: See Appendix.

IV. DESIGN OF ALGORITHMS FOR WASH PROBLEM

Our aim is to find the optimal assignment of each reading task among the heterogeneous disks. However, due to the hardness of WASH problem, the optimal assignment can not be obtained in polynomial time. In order to solve WASH, we first design a heuristic algorithm based on greedy idea, named WASH-greedy. However, extensive experiments shows that WASH-greedy has a little improvement over the baseline, i.e., Hadoop-default which is storage-unaware. Furthermore, In order to solve WASH better, we design a randomized algorithm (WASH-rdm) which chooses source devices based on delicate calculated probabilities and can be proved concentrated on its optimum with high probability, i.e., 1- O(e^{-t^2}), through our theoretical analysis.

A. Heuristic Alogrithm

In this subsection, we first introduce our heuristic algorithm, WASH-greedy, specificly. Based on a greedy idea, while finding the assignment for each task, the disk which has the minimum reading time of one replica, i.e. T_i, will be selected by WASH-greedy. Note that the selected disk must store the replica that the task need. Then, the scheduler will use the assignments to schedule each reading task. To be more specific, WASH-greedy is as follows:

Algorithm 1 WASH-greedy

Require: Find an assignment of each reading task-j, $j \in \mathcal{T}$, among the disks which store one replica of data ϕ_j.

1: $Result \leftarrow \{\}$
2: **for** each task-j, ($j \in \mathcal{T}$) **do**
3: $\{d_1^j, d_2^j, \ldots d_C^j\} = f(\phi_j)$
//Function f is a mapping from data to disks which stores the replicas of the data.
4: $d_{j_{min}} \leftarrow \min_{d \in \{d_1^j, d_2^j, \ldots d_C^j\}} \{T_{d_{j_l}}\}$
5: $Result \leftarrow Result \cup \{\langle j, d_{j_{min}} \rangle\}$
6: **end for**
7: Use $Result$ to schedule each task-j, $j \in \mathcal{T}$.

Algorithm 1 shows the detailed of WASH-greedy. Line 1-4 initializes $Result = \{\}$. Line 2-6 is to select a source disk for each task-j. In line 3, function f is used for task-j to find all disks that store replicas of data ϕ_j, denoted by $\{d_1^j, d_2^j, \ldots d_C^j\}$. Next, in line 4, the disk with minimum reading time is selected from the C disks. After putting the task-j and disk d_{j_l} into the set $Result$, the algorithm completes one assignment of a task. After $|\mathcal{T}|$ iterations, WASH-greedy come to an end.

Next, we use an example to illustrate WAHS-greedy. In data center, there exits a set of heterogeneous disks $\mathcal{D}$= {1, 2, 3, 4} with T_1 = 0.2, T_2 = 0.25, T_3 =0.4 and T_4 =0.6, respectively. Data set $\mathcal{M}$ = {1, 2, 3, 4} are stored as Fig.3. Obviously, each data has two replicas. When query tasks $\mathcal{T}$= {1, 2, 3, 4} (ϕ_j = j, 1≤j≤4) comes, the algorithm WASH-greedy runs as follows:

Initial: Reslut = {}

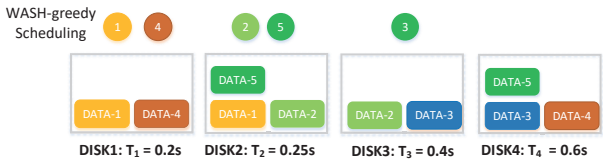


Fig. 3. An execution example of greedy algorithm

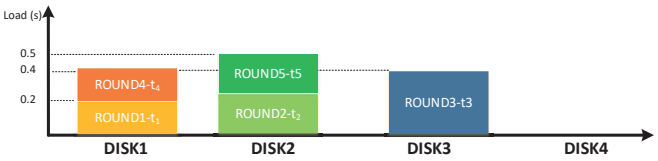


Fig. 4. The process of WASH-greedy algorithm execution for the data stored as Fig.3.

- **Round 1**:for task-1: f(ϕ_1) = {1, 2} = {DISK1, DISK2}
  1 = min{T_1 = 0.2, T_2 = 0.25}
  Result = Result ∪ $\langle 1, 1 \rangle$ = {$\langle 1, 1 \rangle$}
- **Round 2**:for task-2: f(ϕ_2) = {2, 3}
  2 = min{T_2 = 0.25, T_3 = 0.4}
  Result = Result ∪ $\langle 2, 2 \rangle$ = {$\langle 1, 1 \rangle$, $\langle 2, 2 \rangle$}
- **Round 3**:for task-3: f(ϕ_3) = {3, 4}
  3 = min{T_3 = 0.4, T_4 = 0.6}
  Result = Result ∪ $\langle 3, 3 \rangle$ = {$\langle 1, 1 \rangle$, $\langle 2, 2 \rangle$, $\langle 3, 3 \rangle$}
- **Round 4**:for task 4: f(ϕ_4) = {1, 4}
  1 = min{T_1 = 0.2, T_4 = 0.6}
  Result = Result ∪ $\langle 4, 1 \rangle$ = {$\langle 1, 1 \rangle$, $\langle 2, 2 \rangle$, $\langle 3, 3 \rangle$, $\langle 4, 1 \rangle$}
- **Round 5**:for task-5: f($\phi(5)$) = {2, 4}
  2 = min{T_2 = 0.25, T_4 = 0.6}
  Result = Result ∪ $\langle 5, 2 \rangle$ = {$\langle 1, 1 \rangle$, $\langle 2, 2 \rangle$, $\langle 3, 3 \rangle$, $\langle 4, 1 \rangle$, $\langle 5, 2 \rangle$}

Then, the four tasks read data following the assignment of {$\langle 1, 3 \rangle$, $\langle 2, 1 \rangle$, $\langle 3, 2 \rangle$, $\langle 4, 1 \rangle$, $\langle 5, 2 \rangle$, $\langle 6, 1 \rangle$} to read data.The specific steps of WASH-greedy algorithm is shown in Fig.4.

B. Randomized Alogrithm

In the previous subsection IV-A, we design a greedy heuristic algorithm to solve WASH. However, our extensive experiments prove low-performance of WASH-greedy. Therefore, in this subsection, we design a randomized algorithm, named WASH-rdm, which chooses source devices based on delicate calculated probabilities. Specificly, by relaxing the WASH problem to WASH-relaxation problem, then our algorithm uses linear programming tools to solve WASH-relaxation. The decomposition obtained by linear programming is a fraction rather than an integer, which represents the preference when selecting source disks. Subsequently, we prove WASH-rdm is concentrated on its optimum with high probability, i.e., 1- O(e^{-t^2}), through our theoretical analysis where t is the concentration bound.

*a) **Relaxation of WASH Problem**:* According to the previous analysis III-C, this problem is NP-hard, whose optimal assignment of each task cannot be obtained in polynomial time. But linear programming (LP) problem is solvable in

polynomial time, a natural point of view is to relax the WASH problem to LP. The method is to change the domain of decision variables from integer domain {0, 1} to real domain [0, 1], i.e., WASH-relaxation. Such process is named **relaxation**. The solution of WASH-relaxation provides a lower bound for the original WASH problem (WASH is a minimization problem. For the maximization problem is on the contrary). Next, based on the solution of linear programming, the fractional solution is mapped back to integer in some way, which is named **rounding**.

$$
\begin{aligned}
Min: \quad & \max_i\{\sum_j p_i^j * T_i\} \qquad [\text{WASH} - \text{relaxtion}] \\
s.t. \quad & \sum_i p_i^j = 1, \ \ \forall j, \\
& p_i^j \le \pi_{\phi_j}^i, \ \ \forall i, j, \\
& p_i^j \in [0, 1], \ \ \forall i, j.
\end{aligned}
$$

Based on relaxation-rounding, we propose WASH-rdm algorithm. More concretely, WASH-rdm is described as follows:

Algorithm 2 WASH-rdm

Require: Find an assignment of each reading task-j, $j \in \mathcal{T}$, among the disks which store one replica of data ϕ_j.

1: $Result \leftarrow \{\}$
2: $\{p_i^j\}$ = WASH-relaxation
   // $\{p_i^j\}$ is the solution of WASH-relaxation problem.
3: According to $\{p_i^j\}$, use rounding strategy and get $\{I_i^j\}$
4: **for** $\forall$ i, j, ($i \in \mathcal{D}$, $j \in \mathcal{T}$) **do**
5: **if** I_i^j == 1 **then**
6: $Result \leftarrow Result \cup \{\langle j, i\rangle\}$
7: **end if**
8: **end for**
9: Use $Result$ to schedule each task-j, $j \in \mathcal{T}$.

The detailed of WASH-rdm is shown in Algorithm 2. Line 1 initializes the set of $Result$, which stores the final results. In the second line, WASH-relaxation problem is solved by linear programming tools. The third line uses rounding strategy to convert fractional solution to integer solution. Line 4-9 organize the results. Then the tasks will read the disk according to $Result$. The specific rounding strategies are as follows:

The value of $\{p_i^j\} \in$ [0, 1] which shows the correlation between task-j and disk-i. Then, we can select a disk-i for task-j with probability $\{p_i^j\}$. The method is to use the parameter q_j, which is randomly selected in (0,1]. If the $q_j \in (\sum p_{r-1}^j, \sum p_r^j]$, then $I_r^j = 1$, otherwise, I_i^j = 0 ($\forall$ i, $i \neq r$). This approach ensures only one disk can be selected for each task.

*b) **Analysis of WASH-rdm Algorithm**:* In this section, we will prove that our WASH-rdm can be proved concentrated on its optimum with high probability, i.e., 1- O(e^{-t^2}). Firstly, we prove that the difference between reading time of disk-i contributed by any task-j and expectation is martingales sequence [12]. Secondly, Based on the martingales sequence, we use Azuma's Inequality to illustrate the bound between the feasible solution and the optimal solution.

Theorem 2: $Pr[SOL(\text{WASH-rdm}) - OPT(\text{WASH}) \le t] \ge 1 - O(e^{-t^2})$.

Proof:

SOL(WASH-rdm) denotes the feasible solution found by WASH-rdm and OPT(WASH) denotes the optimum solution of WASH problem.

Firstly, task-j's contribution to disk-i's load is expressed as:

$$Z_i^j = I_i^j * T_i \tag{4}$$

From the rounding strategy in Algorithm 2, we can get that:

$$Pr[I_i^j = 1] = p_i^j, \ \ \forall i, j$$

The expectation of Z_i^j is

$$
\begin{aligned}
E[Z_i^j] \ &= E[I_i^j] * T_i \\
&= (Pr[I_i^j = 1] * 1 + Pr[I_i^j = 0] * 0) * T_i \\
&= p_i^j * T_i
\end{aligned} \tag{5}
$$

The difference between Z_i^j and $E[Z_i^j]$ defines as

$$Q_i^j = Z_i^j - E[Z_i^j] \tag{6}$$

For the all $|\mathbb{T}|$ tasks, $L_i^{|\mathbb{T}|}$ denotes the sum of Q_i^j in disk i:

$$L_i^{|\mathbb{T}|} = \sum_{j=1}^{|\mathbb{T}|} Q_i^j = \sum_{j=1}^{|\mathbb{T}|-1} L_i^j + Q_i^{|\mathbb{T}|} \tag{7}$$

Then, on the condition L_i^1, L_i^2, ..., L_i^{r-1}, the expectation of L_i^r is:

$$
\begin{aligned}
& E[L_i^r | L_i^1, L_i^2, ..., L_i^{r-1}] \\
& \overset{(7)}{=} E[L_i^{r-1} + Q_i^r | L_i^1, L_i^2, ..., L_i^{r-1}] \\
& = E[L_i^{r-1} | L_i^1, L_i^2, ..., L_i^{r-1}] + E[Q_i^r | L_i^1, L_i^2, ..., L_i^{r-1}] \\
& \overset{(6)}{=} L_i^{r-1} + E[Z_i^r - E[Z_i^r] | L_i^1, L_i^2, ..., L_i^{r-1}] \\
& = L_i^{r-1} + E[Z_i^r | L_i^1, L_i^2, ..., L_i^{r-1}] - E[E[Z_i^r] | L_i^1, L_i^2, ..., L_i^{r-1}] \\
& = L_i^{r-1} + E[Z_i^r] - E[Z_i^r] \\
& = L_i^{r-1}
\end{aligned} \tag{8}
$$

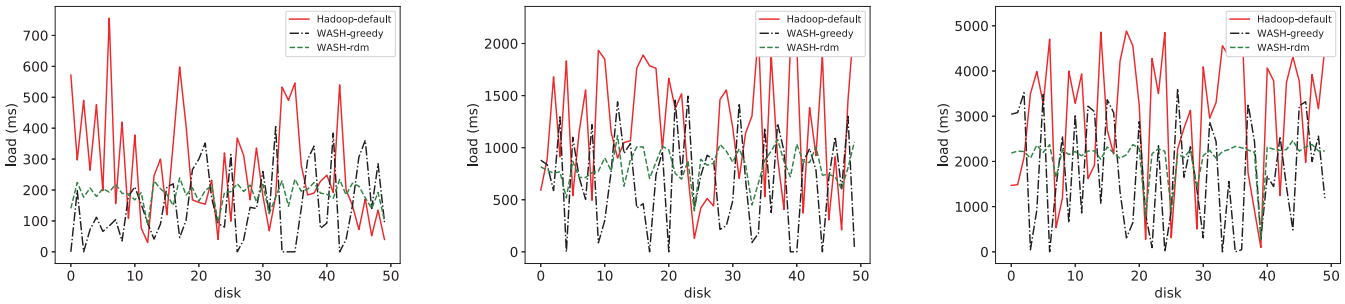
Therefore, L_i^1, L_i^2, ..., $L_i^{|\mathcal{T}|}$ are martingales sequence [13]. For completeness, we let L_i^0 = 0. And $\forall r \ge 1$, we have:

$$|L_i^r - L_i^{r-1}| \overset{(9a)}{=} |Q_i^r| \overset{(9b)}{=} |Z_i^j - E[Z_i^j]| \le g_i^r \tag{9}$$

$$g_i^r = \max\{T_i - E[Z_i^j], E[Z_i^j]\} \tag{10}$$

In (9), the existences of (9a) and (9b) are due to 7 and 6, respectively. From above (10), any two adjacent values, i.e., L_i^r, L_i^{r-1}, in the martingales sequence have constant bounds g_i^r. Based on (8) and (10), we can use Azuma's Inequality. Then,

$$Pr\{L_i^{|\mathcal{T}|} - L_i^0 \ge t\} \le exp\{-\frac{t^2}{2\sum_{i=1}^{|\mathcal{T}|}(g_i^k)^2}\} \tag{11}$$



(a) Small Workload (b) Medium Workload (c) Large Workload

Fig. 5. Comparison of WASH-rdm and other Algorithm on different Workloads. 5(a) denotes the results on Small Workload which has about 500 tasks. 5(b) denotes the results on Medium Workload which has about 2000 tasks. 5(c) denotes the results on Large Workload which has 5000 tasks. X-axis denotes the disk. Y-axis denotes the load of each disk after running the three algorithms.

Substitute equations (6) and (7) into the upper equation (11). Then,

$$Pr\{\sum_{j=1}^{|\mathcal{T}|} Z_i^j - \sum_{j=1}^{|\mathcal{T}|} E[Z_i^j] \geq t\} \leq exp\{-\frac{t^2}{2\sum_{i=1}^{|\mathcal{T}|}(g_i^k)^2}\} \quad (12)$$

The (12) is equal to:

$$Pr\{\sum_{j=1}^{|\mathcal{T}|} Z_i^j \leq \sum_{j=1}^{|\mathcal{T}|} E[Z_i^j] + t\} \geq 1 - exp\{-\frac{t^2}{2\sum_{i=1}^{|\mathcal{T}|}(g_i^k)^2}\} = 1 - O(e^{-t^2}) \quad (13)$$

Let $S_i = \sum_{j=1}^{|\mathcal{T}|} Z_i^j$, $E_i = \sum_{j=1}^{|\mathcal{T}|} E[Z_i^j] \overset{(a)}{=} \sum_{j=1}^{|\mathcal{T}|} p_i^j * T_i$, where the existence of (a) is due to 5. Then,

$$Pr\{S_i \leq U_i + t\} \geq 1 - O(e^{-t^2}) \quad (14)$$

S_i represents the load of disk-i which is the result of ILP. And E_i denotes the expectation which is the result of LP. Because LP provides a lower bound for the optimal solution of ILP problem (WASH is a minimization problem. For the maximization problem is on the contrary), we have:

$$E_i \leq OPT(\text{WASH}) \quad (15)$$

Take S_u, E_v as

$$S_u = S_{max} = \max_i S_i \quad (16)$$

$$E_v = E_{max} = \max_i E_i \quad (17)$$

Then, we have the following inequalities,

$$SOL(\text{WASH-rdm}) = S_u \overset{(18a)}{\leq} E_u + t \overset{(18b)}{\leq} E_v + t \overset{(18c)}{\leq} OPT(\text{WASH}) + t \quad (18)$$

In (18), the existences of (18a), (18b) and (18c) are due to (13), (17) and (15), respectively. From (14) and (18), we can conclude that,

$$Pr\{SOL(\text{WASH-rdm}) < OPT(\text{WASH}) + t\} \geq 1 - O(e^{-t^2}) \quad (19)$$

Observed from the (19), the feasible solution SOL(WASH-rdm) found by the algorithm and the optimal solution OPT (WASH) are approximated by probability $1 - O(e^{-t^2})$. When hundreds of tasks are deployed with $1 - O(e^{-t^2}) = 0.85$, the value of t is acceptable, only a few millisecond. □

V. PERFORMANCE EVALUATION

In this section, we make extensive simulations, comparing our algorithm, i.e., WASH-rdm, with WASH-rdm and storage-unaware scheduling algorithm. The results of simulations show that performance of our proposed WASH-rdm can improves by up to 55% and 25%, respectively.

A. Simulation Settings

Usually, the traditional scheduling mechanism are not aware of the heterogeneity of the disk. For example, in Hadoop the scheduler cannot know the load of the disk when deploying the task. When selecting data replica for tasks, it is usually a random selection strategy, which we call Hadoop-default. We compare our proposed algorithms with it.

Workload: Referring to the characteristics of Google traces [20], we classify workload into three categories: Small Workload, Medium Workload and Large Workload, as shown in TABLE II. In Small Workload, most of the jobs have 1-150 tasks. In has Large Workload, there are 50% of the jobs that has over 50 tasks. The Medium Workload is in the middle of them.

TABLE II
THE DIFFERENT TRACES USED FOR EXPERIMENT

Traces \ Number of tasks	1-150	150-500	≥ 500
Small Workload	96%	2%	2%
Medium Workload	50%	40%	10%
Large Workload	40%	10%	50%

Setting: In distributed file systems [19], data is distributed uniformly on disk. Therefore, in this experimental environment, we uniformly deploy 200000 data with three data replicas by default, each data block size τ = 128MB. 500

disks are set up by default. The T_i of the disk in the cluster is between 10ms and 500ms.

B. Simulation Results

For convenience of display, Fig.5. shows the results under different workloads upon 50 disks. X-axis denotes the disks. Y-axis denotes the load of each disk after running the three algorithms. The lower the peak of the curve, the better the result. The black dotted line represents the execution result of Algorithm WASH-greedy, and the green line represents the execution result of Algorithm WASH-rdm. Overall, we can see that our algorithm performs better than Hadoop-default. As shown in Fig.5(a), in the small workload, the result of Hadoop-default has large fluctuations. This is a result of that hadoop-default is not aware of the T_i and load of the disk when selecting a replica for the task. Even if the tasks are deployed uniformly on disks, the disk with low performance may have a relatively large load. In Fig. 5(a), low performance disks will have high peaks. Completion of all parallel tasks is the end of the big data processing job. Therefore, the task reading data from the disk with poor performance will be the bottleneck of data processing job. For example, Hadoop-default causes the tasks of reading data from disk-2 to be the bottleneck of big data processing in the Fig.5(a) which is 654. Algorithms WASH-greedy and WASH-rdm can maintain roughly all disks at a lower load, less than 300. The reason is that WASH-greedy can select the disk to read for the task according to load of the disks and WASH-rdm can select the disk to read for the task according to load of the disks the performance of the disk. The maximum load of WASH-greedy algorithm is 261 in disk-12. Compared with the Hadoop-default, the performance is improved by 64%. The maximum load of WASH-rdm algorithm is disk-4, and the corresponding load is 284. Compared with the baseline algorithm, the performance of WASH-rdm algorithm improves 57%. As the number of tasks increases, the load on all disks increases. For example, in Fig.5(b), when the number of tasks is about 2000, in Hadoop-default algorithm the maximum load is 2009 in disk-9. The results of WASH-greedy and WASH-rdm are that 784 in disk-8 and 770 in disk-6, respectively. The performance of our algorithm improves by 61% and 62%. When the number of tasks becomes larger, in the large Workloads of Fig. 5(c), the maximum load obtained by the three algorithms are 5546,2205 and 2646. WASH-greedy increases by 60% and WASH-rdm performance improves by 52%.

Next, we study the impact of different factors workloads, degree of heterogeneity and the number of data replicas, on the load upon the default configuration. In Fig.6(a), the horizontal X-axis represents different workloads, and the Y-axis represents the maximum load in the cluster, which has been normalized. In the previous part, impact of Workloads is that the overall load on disk shows an increasing trend as the Workloads gets larger. Fig.6(a) shows the difference among the three workloads. The performance of our proposed algorithm can always be improved by 60% on average. Next, we observed the effect of degree of heterogeneity on the load. We divide the heterogeneity of the cluster into two kinds. One is that the T_i of disks is evenly distributed between 10ms and 100ms, named Low Heterogeneous Cluster. Another is High Heterogeneous Cluster, where the T_i of disks are evenly distributed between 10ms and 500ms. Other conditions are configured by default. From Fig.6(b), when the degree of heterogeneity gets larger, the overall load shows an increasing trend. This is because the large the degree of heterogeneity, the greater the performance difference between disks. This leads to hard balancing of disk loads. Through 6(b), we find that our algorithm can also improve the performance of 40-50% in clusters with high heterogeneity. Finally, we analyze the impact of the amount of data replicas on disk load. We set the number of data replicas as 2, 3 and 5. Observe that the higher the number of replicas, the lower the disk load. Reason for it is that with the number of data replicas increasing, the probability of data being deployed on high performance disks increases. Selecting data on high-performance disks at this time will greatly accelerate the completion of tasks. When C = 5, our algorithm can achieve 65% performance improvement.

All in all, the performance of the two algorithms proposed by us has reached an average of 55% improvement over baseline algorithm on average under different conditions.

VI. CONCLUSION

In this paper, in order to keep the balanced use on disks and speed up the reading time of data analytical tasks, the type of heterogeneous-storage and the workloads of storage devices both should be taken into consideration. Therefore, we formulate the workload-aware scheduling problem for heterogeneous-storage, showing its NP-hardness and design a randomized algorithm with performance-guarantee. The results of our simulation show that our proposed WASH-rdm speeds up data analytics up to 55% over the baseline algorithms.

APPENDIX

The Proof of Theorem 1

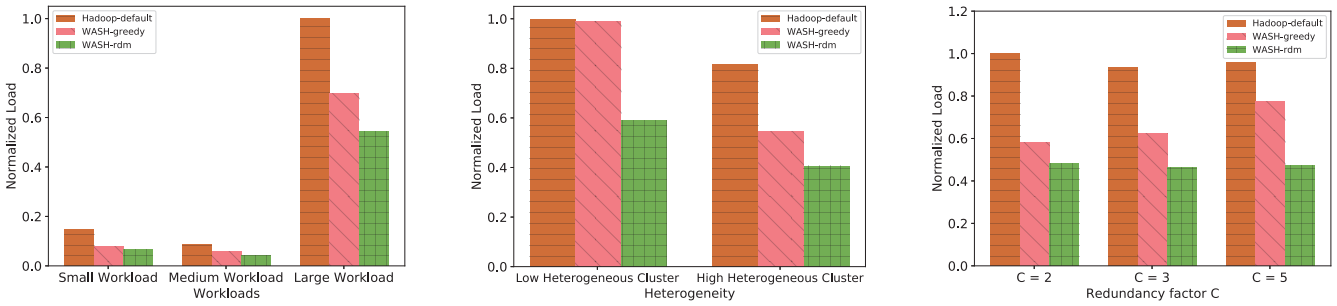
The canonical form [11] of ILP is described below.
For $m \times n$ matrix $\mathbf{A}$

$$\begin{aligned} Min: \quad & \mathbf{c}^T\mathbf{x} \quad [\text{ILP}] \\ s.t. \quad & \mathbf{Ax} \geq \mathbf{b}, \\ & \mathbf{x} \geq 0 \; and \; \mathbf{x} \in \mathbb{Z}^n. \end{aligned} \tag{20}$$

Taking the special case of WASH problem, the parameters are set as follows:

- $\mathcal{D}$ = {0, 1, 2};T_0 = 1, T_1 = 1, T_2=1; $\mathbb{M}$ = {m_0, m_1, ..., m_{n-1}}; C = 3
- T = {0, 1, ..., t_{n-1}}

It means that there are 3 disks, 0, 1, 2, in the data center, each disk with a T_i of 1. A total of n data, and each data has three replicas which deployed in the three disks, respectively. When the query job arrives, the job is divided into n tasks, {t0, 1, ..., n} to read n data, {0, 1, ..., n}, respectively.



(a) Results under varilous setting on Workloads

(b) Results under different Heterogeneity cluter.

(c) Results under varilous setting on reolicas number

Fig. 6. Comparison of WASH-rdm and other Algorithm on different workloads, different heterogenelity cluter and different replica number C. 6(a) denotes effect of different workloads on reading time. 6(b) denotes the reading time of the three algorithm on different Heterogeneity cluter. 6(c) denotes the reading time of the three algorithm when C is different. Y-axis denotes the reading time of all tasks which has been normalized.

For this particular case, the problem description can be modified to:

$$Min: \quad \max\{\sum_j I_0^j, \sum_j I_1^j, \sum_j I_2^j\} \quad [\text{WASH} - \text{sp}] \tag{21}$$

$$\begin{aligned} s.t. \quad & I_0^j + I_1^j + I_2^j = 1, \ \forall j, \\ & I_0^j, I_1^j, I_2^j \in \{0,1\}, \ \forall j \in [0,n). \end{aligned}$$

Let $x = \sum_j I_0^j$, $y = \sum_j I_1^j$, $z = \sum_j I_2^j$, and the problem description can be modified to:

$$Min: \quad \max\{x,y,z\} \quad [\text{WASH} - \text{sp}] \tag{22}$$

$$\begin{aligned} s.t. \quad & x + y + z = n, \\ & 0 \le x,y,z \le n \ \ x,y,z \in \mathbb{Z}. \end{aligned} \tag{23}$$

In the minimization problem, x + y + z = n can be modified to x + y + z = n and let R denotes $\max\{x,y,z\}$.

Finally, WASH-sp problem is converted to:

$$Min: \quad 0*x + 0*y + 0*z + R \quad [\text{WASH} - \text{ILP}] \tag{24}$$

$$\begin{aligned} s.t. \quad & x + y + z \ge n, \\ & -x + R \ge 0, \\ & -y + R \ge 0, \\ & -z + R \ge 0, \\ & 0 \le x,y,z \le n \ \ x,y,z \in \mathbb{Z}, \\ & R \ge 0, R \in \mathbb{Z}. \end{aligned}$$

Obviously, WASH-ILP in (24) is an instance of ILP in (20). The decision version of 0-1 ILP (A variation in which only the restrictions must be satisfied, without optimization) is one of Karp's 21 NP-complete problems [9], so ILP is NP-hard. Further, the WASH-ILP problem is NP-hard.

On the one hand, when the optimal solution of WASH-ILP problem is obtained, the corresponding x, y, z can be converted into variable I_i^j in polynomial time by setting the value in $\{I_0^0, I_0^1, ..., I_0^{x-1}, I_1^x, I_1^{x+1}, ..., I_1^{x+y-1}, I_2^{x+y}, I_1^{x+y+1}, ..., I_2^{x+y+z-1}\}$ to be 1, which will make the corresponding WASH-sp problem obtain the optimal solution. On the other hand, when the variable $\{I_i^j\}$ make the WASH-sp problem obtain the optimal solution, the WASH-ILP also obtains the optimal solution by setting $x = \sum_j I_0^j$, $y = \sum_j I_1^j$, $z = \sum_j I_2^j$. It can be seen from the above that any solution can be polynomially transformed between the WASH-sp and WASH-ILP. Therefore, WASH-sp is NP-hard.

As a result of that WASH-sp is a special case in WASH, WASH problem is NP-hard. □

REFERENCES

[1] Ahmad F, Chakradhar S T, Raghunathan A, et al. Tarazu: optimizing mapreduce on heterogeneous clusters[C]//ACM SIGARCH Computer Architecture News. ACM, 2012, 40(1): 61-74.

[2] Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]//Proceedings of the 5th European conference on Computer systems. ACM, 2010: 265-278.

[3] Ananthanarayanan G, Agarwal S, Kandula S, et al. Scarlett: coping with skewed content popularity in mapreduce clusters[C]//Proceedings of the sixth conference on Computer systems. ACM, 2011: 287-300.

[4] Abad C L, Lu Y, Campbell R H. DARE: Adaptive data replication for efficient cluster scheduling[C]//2011 IEEE international conference on cluster computing. IEEE, 2011: 159-168.

[5] Jalaparti V, Bodik P, Menache I, et al. Network-aware scheduling for data-parallel jobs: Plan when you can[C]//ACM SIGCOMM Computer Communication Review. ACM, 2015, 45(4): 407-420.

[6] Xu L, Butt A R, Lim S H, et al. A heterogeneity-aware task scheduler for spark[C]//2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2018: 245-256.

[7] Pan F, Xiong J, Shen Y, et al. H-Scheduler: Storage-Aware Task Scheduling for Heterogeneous-Storage Spark Clusters[C]//2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2018: 1-9.

[8] Wang B, Jiang J, Yang G. Actcap: Accelerating mapreduce on heterogeneous clusters with capability-aware data placement[C]//2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, 2015: 1328-1336.

[9] Karp R M. Reducibility among combinatorial problems[M]//Complexity of computer computations. Springer, Boston, MA, 1972: 85-103.

[10] Hromkovič J. Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics[M]. Springer Science & Business Media, 2013.

[11] 2018. Integer programming. https://en.wikipedia.org/wiki/Integer_programming.

[12] Grimmett G, Grimmett G R, Stirzaker D. Probability and random processes[M]. Oxford university press, 2001.

[13] 2019. Martingale (probability theory). https://en.wikipedia.org/wiki/Martingale_(probability_theory)

[14] Wang B, Jiang J, Yang G. Actcap: Accelerating mapreduce on heterogeneous clusters with capability-aware data placement[C]//2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, 2015: 1328-1336.

[15] Apache Spark. https://spark.apache.org, 2019.
[16] Seagate. https://www.seagate.com/cn/zh/.
[17] Samsung. https://www.samsung.com/semiconductor/cn/, 2019.
[18] Guo Z, Fox G, Zhou M. Investigation of data locality in mapreduce[C]//Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society, 2012: 419-426.
[19] Hdfs. https://hadoop.apache.org/hdfs, 2019.
[20] 2012. Google Cluster Trace. https://code.google.com/p/googleclusterdata/.
[21] F. Ahmad, S. Chakradhar A. Raghunathan and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In ACM ASPLOS 2012.
[22] R. Gandhi, D. Xie, Y. Charlie. PIKACHU: How to Rebalance Load in Optimizing MapReduce On Heterogeneous Clusters. In USENIX ATC 2013.
[23] Malik M, Neshatpour K, Rafatirad S, et al. Hadoop workloads characterization for performance and energy efficiency optimizations on microservers[J]. IEEE Transactions on Multi-Scale Computing Systems, 2017, 4(3): 355-368.
[24] Gupta P K. Accelerating datacenter workloads[C]//26th International Conference on Field Programmable Logic and Applications (FPL). 2016.
[25] Kong J. Datacenter storage system: U.S. Patent Application 13/694,001[P]. 2014-4-24.
[26] Delimitrou C, Sankar S, Vaid K, et al. Decoupling datacenter studies from access to large-scale applications: A modeling approach for storage workloads[C]//2011 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2011: 51-60.
[27] Guo Y, Gong Y, Fang Y, et al. Energy and network aware workload management for sustainable data centers with thermal storage[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 25(8): 2030-2042.
[28] HDD. https://en.wikipedia.org/wiki/Hard_disk_drive
[29] SSD. https://en.wikipedia.org/wiki/Solidstate_drive
[30] Oceanbase. http://oceanbase.org.cn/?p=151. 2016-3-25.
[31] Stone J E, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems[J]. Computing in science & engineering, 2010, 12(3): 66.
[32] Xie J, Yin S, Ruan X, et al. Improving mapreduce performance through data placement in heterogeneous hadoop clusters[C]//2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW). IEEE, 2010: 1-9.
[33] Stuedi P, Trivedi A, Pfefferle J, et al. Crail: A High-Performance I/O Architecture for Distributed Data Processing[J]. IEEE Data Eng. Bull., 2017, 40(1): 38-49.
[34] Zumberge J F, Urban M P, Liu R, et al. International GPS Service for Geodynamics[J]. 1996.
[35] Kern M, Culhane S, Hoffman D. Systems, Methods and Media for Distributing Peer-to-Peer Communications: U.S. Patent Application 14/035,945[P]. 2014-1-23.
[36] Apache Storm. https://storm.apache.org.
[37] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
[38] Davidson A, Or A. Optimizing shuffle performance in spark[J]. University of California, Berkeley-Department of Electrical Engineering and Computer Sciences, Tech. Rep, 2013.