Workload-Aware Scheduling for Data Analytics upon Heterogeneous Storage

Mingtao Ji, Yibo Jin, Zhuzhong Qian, Sanglu Lu
State Key Lab. for Novel Software Technology, Nanjing University, Nanjing, China
{jmt, yibo.jin}@smail.nju.edu.cn, {qzz, sanglu}@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 diverse demands of various big data workloads. Since the reading performance of various storage devices are quite different, traditional concurrent data fetching easily incurs unbalanced use among devices. It leads to the straggler in terms of the delay of getting data which increases the overall latency of data analytics. To avoid such unbalanced use on fetching large volume of data concurrently from storage devices, we formulate Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH), the goal of which is to minimize the maximum data fetching time for parallel data analytical tasks. We design a randomized algorithm (rWASH), to select a proper source device for each task based on delicate calculated probabilities, which can be proved concentrated on its optimum with high probability. Extensive experiments show that rWASH reduces the average data fetching time for tasks by up to 55% over the state-of-the-art algorithms.**

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

I. Introduction

Nowadays, there is a new trend that data centers have a variety of data analytical workloads, e.g., data streaming applications like Storm [1], and data analytics applications like Grep [2]. Different workloads might have different requirements on storage performance [3] [4] [5] [6]. For example, Storm is a compute-intensive application, whose computation significantly relies on its I/O rate while Grep is an I/O-intensive application, which has high I/O throughput on data. In order to meet these demands, a large number of heterogeneous storage devices [7] have been widely deployed for big data analytics frameworks within the cluster, e.g., Hadoop [8] and Spark [9].

However, the reading performance, i.e., data fetching time from heterogeneous devices, is quite different due to two main aspects. First, the different types of storage, e.g., SSD [10] and HDD [11], naturally result in different reading performance. As shown in Fig. 1, for the same number of data fetching concurrently, fetching data from HDD takes almost twice longer than that from SSD. Second, the number of data fetching concurrently on a device also affects the I/O performance. As further shown in Fig. 1, when the number of data fetching increases, the data fetching time also increases as a result of concurrent use.

Essentially, different reading performance results from the unbalanced use on storage devices, but traditional strategies [12] [13] [14] [15] often ignore to avoid it and incur relatively longer data fetching time. More specifically, fetching data from the devices with higher I/O performance actually earns shorter time than that from those with lower I/O performance [16]. However, large number of I/O requests on the higher I/O performance device still increase the data fetching time and worse correspond data analytics, which should be avoid as much as possible.

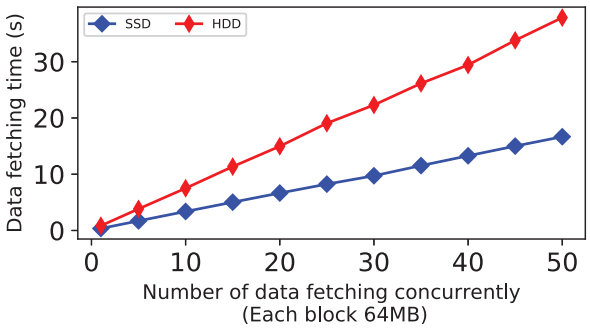


Fig. 1. Comparisons of (1) data fetching time for the same number of data fetching concurrently on various storage, i.e., SSD and HDD. (2) data fetching time for various I/O workloads, i.e., the number of data fetching concurrently.

This paper focuses on dealing with unbalanced use on heterogeneous storage devices in this paper. However, due to the multiple replicas of data and different I/O performance of devices, how to choose target source from feasible ones storing the replica is also a big challenge. Most of the previous researches have already focused on heterogeneous computing resources [17] [18] [19] [20], but ignore the heterogeneous storage devices. Although some works [7] [16] have already considered heterogeneous storage devices to accelerate data analytics, neither unbalanced I/O workloads nor multiple data replicas is considered. We formulate the Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH) which considers both of these factors. Then, we propose a randomized algorithm (rWASH) with guaranteed performance by carefully choosing source device based on delicate calculated probabilities. More concretely, our contributions are listed as follows:

- To balance the I/O workloads on heterogeneous storage devices, we propose Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH).
- We formulate randomized algorithm with guaranteed-performance (rWASH), which can be proved concen-

trated around its optimum with high probability, i.e., 1 - O(e^{-t^2}), where t is the concentration bound.
- Extensive simulations show that rWASH improves by up to 55% over the state-of-the-art algorithm in terms of average data fetching time for tasks.

The rest of the paper is organized as follows. In Section II, we present the related works of the WASH problem. Then the system model and algorithm are proposed in Section III and Section IV, respectively. In Section V, the theoretical analysis of proposed rWASH is presented. At last, we conclude this paper in Section VII by summarizing our main contributions.

II. RELATED WORKS

The related works are summarized from two main parts, i.e., fetching data locally and fetching data remotely. Due to limited bandwidth, the former focused on data locality [12] [13] [14] [15] to reduce data transmission of data analytics. However, such works failed to consider the heterogeneity, which didn't take full advantages of heterogeneous hardware. In the latter part, studies [21] [7] [22] [23] considered hardware heterogeneity by fetching data remotely.

Fetching Data Locally. Due to limited network bandwidth in data center, a large amount of data transmitted between nodes before task execution, greatly affects the performance of data analytics. Therefore, processing data locally as much as possible actually improves the performance, e.g., deploying tasks to the nodes where the input data is stored. In order to improve data locality, Matei [12] proposed delay scheduling. Essentially, it keeps jobs waiting for a while until queuing delay reach to a preset threshold or there is idle resource at local. Ganesh [13] made multiple replicas for frequently used data by analyzing the number of their accesses, improving the data locality. Cristina [14] proposed a distributed adaptive data replication algorithm DARE, which helped the scheduler for better data locality. Jalaparti [15] believed that most of the production workloads were repetitive and predictable, by which the scheduler could make the scheduling plan before hand. All of these works improves the performance of data analytics by considering data locality. However,since that local devices might have high I/O workloads, previous strategies would increase the data fetching time from crowded ones.

Fetching Data Remotely. In heterogeneous cluster, data fetching remotely from those devices with higher I/O performance could be better than that from local ones. However, most of the researches only focused on heterogeneous computing resources [17] [18] [19] [20] [7], but ignored the heterogeneous storage devices. For example, Xu [7] considered the current usability of underlying hardware (CPU, memory, I/O), but didn't consider the performance on different storage hardware. In Hadoop 2.3, HDFS [22] took the heterogeneous storage features into consideration, which supported multiple storage strategies. Users could choose one of these strategies to store files according to their demands. Based on such feature, Pan [16] proposed H-Scheduler to launch tasks according to the different performance between HDD and SSD. However, such scheduling mechanism used heuristic method to read data from HDD or SSD, ignoring unbalanced workloads among them. Wang [23] used Markov to model the use of nodes in the cluster, which guided suitable data distribution among heterogeneous nodes, but ignored the data fetching time and the heterogeneity of storage devices.

Among these works, either the data fetching time or the data replicas is not considered. Thus, there is still a situation where the use of storage devices is unbalanced, arising disks' bottlenecks. The difference between those works and ours is that our work specifically models the reading differences and the I/O workloads among storage devices. Then, the data fetching time of each task could be accelerated by choosing source disks based on delicate calculated probabilities.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first introduce the background of heterogeneous storage devices used in data analytics and then a experimental study is presented to motivate the need for accelerating data analytics. After that we build the system model and formulate the Workload-Aware Scheduling problem for Heterogeneous devices (WASH).

A. Background

In a data center, Hadoop and Spark widely use heterogeneous storage devices to support different workloads. For example, in MapReduce framework, Shuffle stage [24] [25] usually has a large requirement on I/O for storing intermediate data. In order to meet this demand, Crail [26] is proposed to be a high-performance I/O architecture for data processing, using NVMe SSD [27] as a supplementary to speed up the Shuffle phase.

Data and replicas. In data analytics system, large volume of data is generated, such as GPS information [28], system logs [29], etc. Due to their massive sizes, a large file is often divided into multiple fixed-size pieces, i.e., one $data$ block, stored in a distributed file system such as HDFS [22], 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 [10]. In order to avoid such data loss, traditional approach is to keep multiple $replicas$ of each data in storage, e.g., three backups across two racks [22]. Then, one piece of data will be stored in multiple disks with various I/O performance.

Job and tasks. A data analytics workload, named a job, includes many 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 source device for each task among multiple replicas. Note that the completion of a job relies on the most straggling task.

However, traditional schedulers for data analytics are usually unaware of disks' types and I/O workloads, which often leads to a straggler. In contrast, our scheduler is then designed to avoid the stragglers by balancing I/O workloads among heterogeneous storage devices.

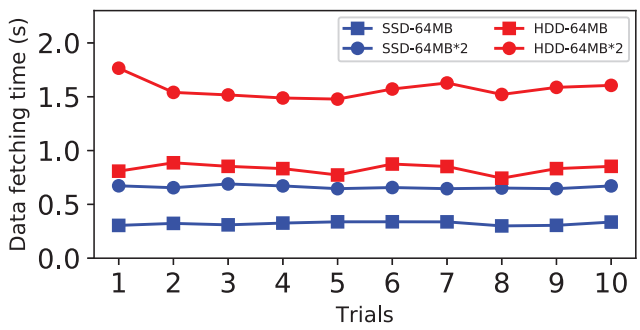
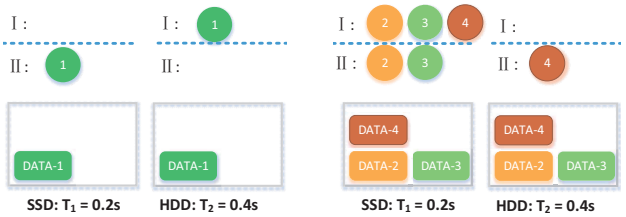


Fig. 2. Comparisons of data fetching time with 10 repeated trails when the number of data fetching concurrently is 1 and 2 in Fig. 1.



I : $max\{0s, 0.4s\}$,
II : $max\{0.2s, 0s\}$.
(a) Type of storage.

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

Fig. 3. Two aspects that affect the data fetching time: (a) Type of storage, (b) Load of storage. Delicate method, i.e., Scheduling II, improves 50% and 33% over scheduling I, respectively.

B. Motivational study

1) Experimental study: To show that the types and I/O workloads of heterogeneous storage have great influence on data fetching time, we conduct several data fetching experiments upon heterogeneous storage devices, i.e., HDD and SSD. We configure that each data block has a fixed-size, i.e., 64MB, corresponding to one data fetching task. Due to that the number of data fetching tasks deployed on each storage devices is no more than 50 in a submitted job in general, the number of data fetching concurrently is set from 1 to 50.

More specific, the results of our experiments are shown as Fig. 1. X-axis represents the number of data fetching concurrently and Y-axis represents the overall data fetching time for all deployed tasks. The red line denotes the time of fetching data from HDD while the blue line represents the time of fetching data from SSD. On the one hand, fetching data from different storage devices takes quite divergent time for the same number of data fetching concurrently. For example, in Fig. 1, 20 task fetching corresponding data from HDD concurrently, the overall time taken is around 15s while the data fetching time is 7.5s from SSD. It is due to these two storage devices store data in different ways [30] [31]. One the other hand, the data fetching time is approximately in a line the number of data fetching increasing for the particular storage devices as shown in Fig. 1. More detailed is shown in Fig. 2, we display the data fetching time for 10 repeated trailed trails when the number of data fetching concurrently is 1 and 2. The red line marked by circular represents the data fetching time with only one data fetching task while the red line marked by square represents the data fetching time with two data fetching task concurrently. For each trail, the former almost is twice larger than the latter. From above all, we first conclude that the time of fetching a fixed-size data block, e.g., 64MB, for each storage devices might be quite different. Second, due to the fact that the data fetching time is approximately in a line the number of data fetching increasing, we could represent the overall data fetching on a particular storage devices by summing the time of all data fetching from that device.

2) Motivational example: Then, we will use a simple example to illustrate the importance on choosing heterogeneous storage devices. As shown in Fig. 3, there are two types of disks, i.e., SSD and HDD. The reading time for a data replica from SSD and HDD is $T_1 = 0.2s$ and $T_2 = 0.4s$, respectively.

- The traditional scheduler deploys tasks like scheduling I of Fig. 3(a), whose reading time for a data replica is $0.4s$ while the reading time of the delicate method, i.e., scheduling II, is $0.2s$. Obviously, the task that reads data from SSD has a shorter reading time than that from HDD, i.e., scheduling II is better than scheduling I.
- In Fig. 3(b), since the number of data fetching concurrently is 3 from SSD, the data fetching time of scheduling I is then $0.2s * 3 = 0.6s$, while the data fetching time of the delicate method, i.e., scheduling II, is $max\{0.2s * 2, 0.4s\} = 0.4s$. Apparently, the tasks that read data from such storage device with a lower I/O workloads have a shorter time than that from higher ones, i.e., scheduling II is also better than scheduling I.

This simple case reveals two important findings: (1) The types of heterogeneous storage devices affect data fetching time. (2) The I/O workloads of storage devices also have effects on data fetching time. In order to shorten the delay of fetching data, we should take both of the these two findings into consideration.

C. System Model

The major notations used are listed in Table I. The data center is equipped with heterogeneous disks, hence we denote by $\mathcal{D}$ the heterogeneous disks set, i.e., $\mathcal{D} = \{1, 2, ..., |\mathcal{D}|\}$. Each disk-$i$ corresponds to a different reading time for only one replica as its reading performance, represented by T_i, $i \in \mathcal{D}$. $\mathcal{M}$ is defined as the set of all data in data center, $\mathcal{M} = \{1, 2, ..., |\mathcal{M}|\}$. Each data-$l$, $l \in \mathcal{M}$, has C replicas stored in C different disks whose size is τ, e.g., 64MB. 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

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

For each of submitted job, it is often divided into multiple parallel tasks by data analytics framework, denoted by $\mathcal{T} = \{1, 2, ..., |\mathcal{T}|\}$. Each task-$j$, $j \in \mathcal{T}$, corresponds to one piece of data, whose index is represented by ϕ_j. Since there are C replicas for each data stored in different C disks, task-j can

TABLE I
MAJOR NOTATIONS USED IN THIS PAPER.

Variable	Description
T_i	The time of reading one replica from disk-i for only one task without other concurrent tasks
ϕ_j	The index of data that task-j needs, as its input
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-l are stored in disk-i or not
N_i	Number of tasks which reads data from disk-i
τ	Unified size of each data replica in data center
C	The number of replicas for each data
Decision	**Description**
I_i^j	A decision variable indicating whether task-j chooses the replica stored in disk-i as its input or not

read data-ϕ_j from one of those C disks, whose indexes can be represented as $\{d_1^j, d_2^j, ... d_C^j\}$.

I_i^j is a decision variable, indicating whether task-j chooses the replica stored in disk-i as its input or not. Specifically, we define I_i^j as

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

The distribution of data can be obtained before tasks' arrival, although the traditional distributed file system, e.g., HDFS, doesn't provide the interface to change data distribution for universality and isolation. Then, we denote by $load_i$ the data fetching time of all tasks from disk-i, i.e.,

$$load_i = \sum_j \{I_i^j * T_i\}.$$

In order to balance the use on heterogeneous storage devices as well as optimize the reading time of data analytical tasks, our objective is to minimize the maximal $load$ of all disks.

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

To avoid the bottleneck of storage devices, a better choices is to make reasonable decision for arrived reading tasks by selecting target disk for data fetching. Thus, we propose Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH). Detailed description is illustrated as follows:

$$\begin{aligned} Min: \quad & \max_i \{\sum_j I_i^j * T_i\} \qquad \text{[WASH]} \\ s.t. \quad & \sum_i I_i^j = 1, \ \ \forall j, & (1) \\ & I_i^j \le \pi_{\phi_j}^i, \ \ \forall i, j, & (2) \\ & I_i^j \in \{0, 1\}, \ \ \forall i, j. & (3) \end{aligned}$$

Constraint (1) and Constraint (2) guarantee that task-j only fetches data-ϕ_j from one of those source disks storing the replicas. If one replica of data-ϕ_j is stored in disk-i, then $\pi_{\phi_j}^i$ = 1, otherwise $\pi_{\phi_j}^i$ = 0. The key to solve WASH problem is to determine the value of those decision variables, i.e., I_i^j. However, in general case, the optimization with integral decisions is NP-hard [32].

IV. Design of Randomized Schema

The key to minimize the data fetching time for overall tasks is to find the optimal source disk storing the replica among heterogeneous disks. However, due to its inherent complexity of integral decisions for scheduling, the optimal solutions is hard to be obtained within polynomial time. To solve WASH effectively as well as to avoid the bottleneck, we design a randomized algorithm (rWASH) which takes the I/O workload into consideration and chooses source devices based on delicate calculated probabilities. Through our theoretical analysis, algorithm rWASH can be proved concentrated on its optimum with high probability, i.e., 1- O(e^{-t^2}), where t is the concentration bound.

Specifically, by relaxing the WASH problem to its relaxation version, i.e., WASH-relaxation, we use linear programming technique to solve WASH-relaxation. The solution obtained by linear programming technique, in a sense, represents the preference when selecting source disks. Therefore, our rWASH uses such results as a series of probabilities to choose disk for each task. More detailed is as follows:

*a) **Relaxation of WASH**:* A linear programming (LP) problem is solvable within polynomial time as well as shows a lower bound for the corresponding integer linear programming (ILP) problem. A natural point of view is to relax the WASH problem to a LP problem. The method is to change the domain of decision variables from integer domain $\{0, 1\}$ to real one [0, 1], i.e., WASH-relaxation. Such process is named $Relaxation$. Detailed description is illustrated as follows:

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

Then, we use linear programming technique to solve WASH-relaxation whose solutions are fractions distributed in [0, 1]. In a sense, the fraction solutions represent the preference when choosing the source disk for each task.

*b) **rWASH schema**:* Thus, our rWASH uses these fractions as probabilities to choose source disk for each task. In this way, the solution in fractional domain is mapped back to integer domain, which is named $Rounding$. Based on $Relaxation - Rounding$ strategy, we propose rWASH.

The detailed of rWASH is shown in Algorithm 1. Line 1 initializes the set of $Result$. In line 2, we use linear programming technique to solve WASH-relaxation. For the third line, our proposed rWASH uses rounding strategy to convert fractional solutions into integer solutions. The specific rounding strategy is shown as follows:

The value of $\{p_i^j\}$ shows the correlation between task-j and disk-i. Then, we select a disk-i for each task-j with the

Algorithm 1 rWASH

Require: Task-j and its input data-ϕ_j, $\forall j \in \mathcal{T}$.

1: $Result \leftarrow \{\}$
2: $\{p_i^j\}$ = WASH-relaxation
// $\{p_i^j\}$ is the solution of WASH-relaxation problem.
3: Use $\{p_i^j\}$ twice and get $< \{I_i^j\}_1, \{I_i^j\}_2 >$
// Apply rounding strategy twice.
4: Select $\{I_i^j\}$ from $< \{I_i^j\}_1, \{I_i^j\}_2 >$
// Select $\{I_i^j\}$ which minimizes WASH.
5: **for** each disk-i and each task-j **do**
6: **if** I_i^j == 1 **then**
7: $Result \leftarrow Result \cup \{\langle j, i\rangle\}$
8: **end if**
9: **end for**
10: Each task-j reads data according to $Result$.

probability p_i^j. The method is that, $\forall j$, we randomly select a fraction q_j from (0,1]. If $q_j \in (\sum_{r=1}^{k-1} p_r^j, \sum_{r=1}^{k} p_r^j]$, $2 \leq k \leq \mathcal{D}$, then $I_k^j = 1$, otherwise, I_k^j = 0. This approach ensures only one disk can be selected for each task as well as $Pr[I_i^j = 1] = p_i^j$ and also ensures $I_i^j = 0$ while $\pi_{\phi_j}^i = 0$.

In order to obtain more precise solution, we use rounding strategy twice, i.e., line 3. Then, we choose the one which minimizes the WASH between the two choices, i.e., power of two choices [33]. In lines 5-9, the results are stored in $Result$. After that, the tasks read the data according to $Result$.

V. Analysis of Algorithm rWASH

In this section, we prove that rWASH algorithm is concentrated on its optimum with high probability, i.e., 1- O(e^{-t^2}), where t is the concentration bound. Firstly, we show that the difference between disk-i's reading time contributed by any task-j and its expectation could be bounded through Martingale Analysis [34]. Then, we use Azuma's Inequality to illustrate the gap between the feasible solution and the optimal solution. For simplification, SOL denotes the feasible solution solved by rWASH, and OPT represents the optimal solution.

Theorem 2: $Pr[SOL - OPT \leq t] \geq 1 - O(e^{-2t^2})$.

Proof: Firstly, the contribution on additional workload of each task-j to disk-i's $load$ is expressed as

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

From the rounding strategy in Algorithm 1, we obtain

$$Pr[I_i^j = 1] = p_i^j.$$

The expectation of Z_i^j then is represented as

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

The difference between disk-i's reading time contributed by any task-j, i.e., Z_i^j, and its expectation, i.e., $E[Z_i^j]$, is defined as

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

For each disk-i, $L_i^{|\mathcal{T}|}$ denotes the sum of Q_i^j, $\forall j \in \mathcal{T}$, i.e.,

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

Then, the expectation of L_i^r, on the condition L_i^1, L_i^2, ..., L_i^{r-1} ($r \geq 1$), is represented as follows:

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

The Equation (8a) and Equation (8b) hold due to Equation (7) and Equation (6), respectively. Due to the result in Equation (8) that $E[L_i^r | L_i^1, L_i^2, ..., L_i^{r-1}] = L_i^{r-1}$, we conclude that L_i^1, L_i^2, ..., $L_i^{|\mathcal{T}|}$ is a martingale sequence [35]. For completeness, let L_i^0 = 0. After considering each item and the relationship between two consecutive items in such martingale sequence, $\forall r \geq 1$, we have the following equation:

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

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

Equation (9a) and (9b) hold due to Equation (7) and Equation (6), respectively. For given p_i^j, T_i and $E[Z_i^r]$ are both constant. Thus, in Equation (10), the difference of any two consecutive items, i.e., L_i^r and L_i^{r-1}, $\forall r \geq 0$, in the martingale sequence, has a constant bound, i.e., g_i^r. Based on Equation (8), Equation (10) and Azuma's Inequality, we have

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

Substituting Equation (6) and Equation (7) into the Equation (11), we have

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

where it equals to

$$\begin{aligned} Pr\{\sum_{j=1}^{|\mathcal{T}|} Z_i^j \leq \sum_{j=1}^{|\mathcal{T}|} E[Z_i^j] + t\} &\geq 1 - exp\{-\frac{t^2}{2\sum_{i=1}^{|\mathcal{T}|}(g_i^k)^2}\} \\ &= 1 - O(e^{-t^2}). \end{aligned} \tag{12}$$

For simplification, let $S_i = \sum_{j=1}^{|\mathcal{T}|} Z_i^j$, $E_i = \sum_{j=1}^{|\mathcal{T}|} E[Z_i^j] \overset{(12a)}{=} \sum_{j=1}^{|\mathcal{T}|} p_i^j * T_i$, where (12a) holds due to Equation (5). After substituting S_i and E_i into Inequality (12), we have

$$Pr\{S_i \leq E_i + t\} \geq 1 - O(e^{-t^2}). \tag{13}$$

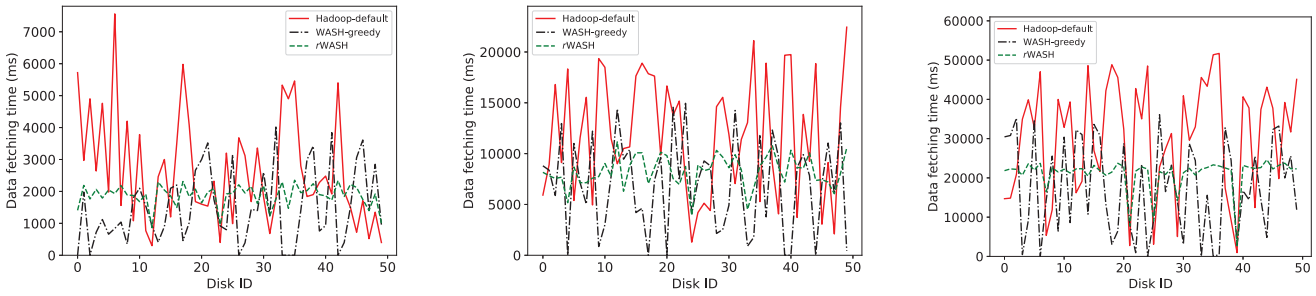
S_i represents the actual workload of disk-i, and E_i denotes the solution of LP as well as the expectation workload of disk-i. Since LP provides a lower bound of the ILP problem (WASH is a minimization problem), $\forall x \in \mathcal{D}$, we have

$$E_x \leq OPT. \tag{14}$$

Without losing generality, u and v represent the indexes of the maximal S_i and E_i, respectively, i.e.,

$$S_u = S_{max} = \max_i S_i, \tag{15}$$

$$E_v = E_{max} = \max_i E_i. \tag{16}$$



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

Fig. 4. Characteristics of I/O workloads among 50 disks under different workloads. rWASH improves by up to 69%, 50% and 56% over Hadoop-default, as well as 42%, 25% and 30% over WASH-greedy in terms of the maximum data fetching time for concurrent data reading.

Algorithm 2 WASH-greedy

Require: Task-j and its input data-ϕ_j, $\forall j \in \mathcal{T}$.

1: $Result \leftarrow \{\}$
2: **for** each task-j **do**
3: $\{d_1^j, d_2^j, \ldots d_C^j\} = f(\phi_j)$
// $f(\phi_j)$ is a set of disks storing data-ϕ_j.
4: $d_{min}^j \leftarrow \underset{d \in \{d_1^j, d_2^j, \ldots d_C^j\}}{\arg\min} \{T_d\}$
5: $Result \leftarrow Result \cup \{< j, d_{min}^j >\}$
6: **end for**
7: Each task-j reads data according to $Result$.

Then, we have the following inequalities,

$$SOL = S_u \overset{(18a)}{\leq} E_u + t \overset{(18b)}{\leq} E_v + t \overset{(18c)}{\leq} OPT + t. \quad (17)$$

Inequality (18a), (18b) and (18c) hold due to Inequality (12), Equation (16) and Inequality (14), respectively. Based on Inequality (13) and Inequality (17), we conclude that

$$Pr\{SOL < OPT + t\} \geq 1 - O(e^{-t^2}). \quad (18)$$

Then, the result of Inequality (18) can be improved to $1 - O(e^{-2t^2})$ by applying power of two choices, i.e., in lines 3-4 of rWASH. In practice, for given probability, e.g., $1-O(e^{-2t^2}) = 0.85$, when hundreds of tasks are deployed, t is acceptable. □

VI. PERFORMANCE EVALUATION

In this section, we conduct extensive simulations for evaluating rWASH. For completeness of our work, we design another one algorithm WASH-greedy. Due to the universality and isolation of common distributed file system like HDFS, intuitively hampers the task scheduler to obtain the workload on disks. Therefore, our greedy-based WASH-greedy only select source disk with minimum data fetching time, i.e., T_i, greedily rather than the disk with minimal workloads. Algorithm 2 shows the details of WASH-greedy. The results of our extensive simulations show that rWASH improves by up to 55% and 30%, respectively, compared with typically and widely used storage-unaware scheduling algorithm as well as WASH-greedy.

A. Simulation Setup

Traditional scheduling mechanisms are often unaware of the types and I/O workloads of storage devices. For example, the default scheduler in Hadoop, whose abbreviation is Hadoop-default, selects source devices storing the related data of tasks randomly. In our extensive simulations, we compare rWASH with WASH-greedy and Hadoop-default for evaluation.

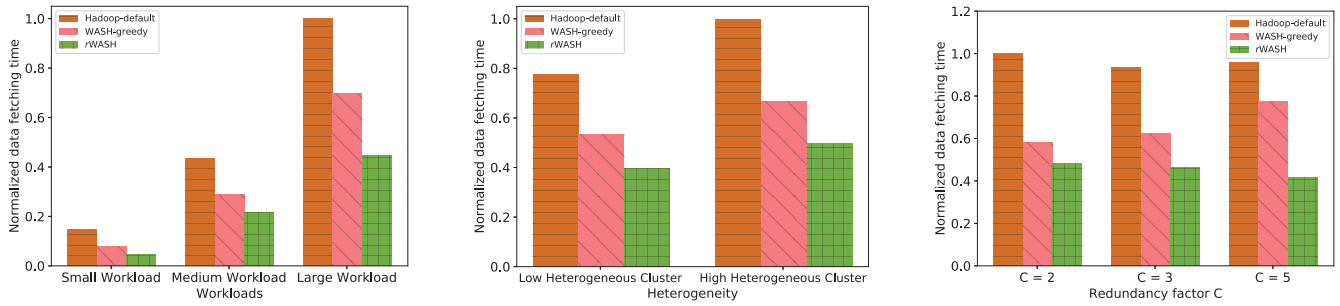
Workloads: According to the characteristics of Google traces [36], we classify the workloads into three categories: Small, Medium and Large. In small workload, most of the jobs have 1-150 tasks. In Large Workload, there are 50% of the jobs that own at least 500 tasks. And, the Medium Workload is in the middle of them in terms of the task number.

Settings: In traditional distributed file systems, e.g., HDFS [22], distributed datasets, i.e., data blocks, are uniformly stored among disks, whose unified size is 128MB. Therefore, in this experimental environment, we uniformly deploy 200000 data blocks among 500 disks, each of which has three data replicas. The data fetching time for only one piece of data, i.e., T_i, ranges from 100ms to 500ms, uniformly.

B. Simulation Results

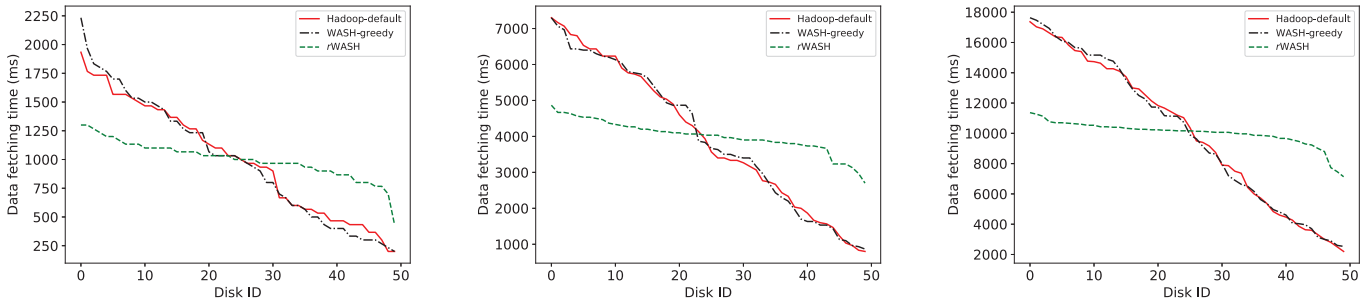
Characteristics of I/O workloads: Fig. 4 shows the results under various workloads among 50 disks. X-axis represents the IDs for different disks while Y-axis represents the data fetching time. Note that the completion of concurrent data fetching relies on the maximum data fetching time of all parallel tasks. The black dotted line illustrates the result of WASH-greedy while the green line illustrates the result of rWASH. In general, rWASH performs much better than Hadoop-default in terms of the maximal data fetching time. As shown in Fig. 4(a) further, i.e., in the small workload scenario, the result of Hadoop-default has large fluctuations since it is unaware of storage types and unbalance I/O workloads among heterogeneous storage devices when selecting a source device for each task. As a result, the straggler task arises which increases the completion time of concurrent data fetching, even if the data is distributed uniformly among storage devices.

More specifically, the maximal data fetching time under Hadoop-default is 7560ms since the bottleneck occurs within disk-6 for concurrent data fetching, and further influences



(a) Results under various I/O workloads (b) Results under various I/O performance (c) Results under various number of data replica

Fig. 5. Results under various scenarios with different I/O workloads, I/O performance and the number of data replicas. rWASH improves 58%, 50% and 53% over Hadoop-default, as well as 34%, 26% and 30% over WASH-greedy in terms of the maximum data fetching time.



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

Fig. 6. Results for various I/O workloads under skewed data distribution. Our rWASH improves by up to 27%, 41% and 34% over Hadoop-default, as well as 38%, 32% and 37% over WASH-greedy in terms of the maximum data fetching time.

the overall latency of data analytics, as shown in Fig. 4(a). Although WASH-greedy reduces the maximal data fetching time to 4050ms, it fails to avoid the hotspots on those disks with higher I/O performance. WASH-greedy actually shows its preference on the disk with higher I/O performance for data fetching, but crowded I/O workloads also increase the data fetching time due to the concurrent use. In contrast, rWASH selects the source devices based on delicate calculated probabilities with peak 2340ms in disk-17. Compared with Hadoop-default and WASH-greedy, rWASH reduces reading time for tasks by up 69% and 42%, respectively.

Further shown in Fig. 4(b) and Fig. 4(c), with the increase number of concurrent data fetching, the maximal data fetching time also increases among disks. When the number of tasks is about 2000 in middle workload scenario, the maximal data fetching time of Hadoop-default and WASH-greedy is 22440ms and 14950ms, respectively. The maximal data fetching time of rWASH is only 11160ms, which speeds up the concurrent data fetching by up to 50% and 25%, respectively. Furthermore, in large Workloads scenario, rWASH reduces reading time for tasks by up to 56% and 30% over Hadoop-default and WASH-greedy, respectively.

Scalability of rWASH: Next, we evaluate rWASH from three main aspects, including I/O workloads, I/O performance and the number of data replicas. In Fig. 5(a), it summarizes the maximal data fetching time for three I/O workloads, illustrating that the average improvement of rWASH compared with other two strategies is at least 34%. In Fig. 5(b), when the average I/O performance is high, i.e., the average reading time for one data block from disks is only 150ms in the scenario of low heterogeneity, the reduction of data fetching time by rWASH compared with Hadoop-default and WASH-greedy is 49% and 26%, respectively. When the average I/O performance is low, i.e., the average reading time for only one data block from disks is 300ms in the scenario of high heterogeneity, the reduction by rWASH compared with Hadoop-default and WASH-greedy is 51% and 25%, respectively.

As shown in Fig. 5(c), with the growth on the number of data replicas, the maximal data fetching time for Hadoop-default and rWASH decreases while the maximal data fetching time for WASH-greedy increases instead. The reason behind is that for large number of data replicas, more data is likely to be stored within those disks with high I/O performance. Thus, WASH-greedy leaves much heavier I/O workloads on those disks since it often shows its preference on those disks, making them being hotspots and increasing the maximal data fetching time due to the concurrent use. Fortunately, for rWASH, it has more choices among heterogeneous disks and has more opportunities to balance the I/O workloads for data analytical tasks. In general, rWASH takes not only the I/O performance of disks but also the I/O workloads into consideration, and is willing to choose almost the best source disk for each task. rWASH improves up to 57% deduction on data fetching time when the number of data replicas is 5.

Impact of data distribution: Fig. 6 shows the results for

various I/O workloads under skewed data distribution. More intuitive, we sort the reading time of each disk in descending order. The skewed data distribution actually influences the maximal data fetching time because more data is stored in few disks. For Hadoop-default and WASH-greedy, the feasible choices of source disks become small, and hence more I/O workloads are assigned to these disks with high I/O performance, incurring high data fetching time. For rWASH, it is willing to choose those with poor I/O performance for offloading. In general, rWASH improves by up to 27%, 41% and 34% over Hadoop-default, as well as 38%, 32% and 37% over WASH-greedy, respectively.

The overall average improvement compared with rWASH and Hadoop-default as well as WASH-greedy is 55% and 30%, respectively.

VII. CONCLUSION

In this paper, in order to avoid the unbalanced use on fetching large volume of data concurrently from heterogeneous storage devices as well as to speed up the data fetching time for data analytics, both the types of storage devices and the I/O workloads should be taken into consideration. Therefore, we formulate the Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH), and design a randomized algorithm (rWASH) which chooses source device for each task based on delicate calculated probabilities and can be proved concentrated on its optimum with high probability. The extensive simulations show that rWASH improves by up to 55% over the state-of-the-art baseline algorithm.

REFERENCES

[1] "Apache storm," https://storm.apache.org, 2018.

[2] M. Malik, K. Neshatpour, S. Rafatirad, and H. Homayoun, "Hadoop workloads characterization for performance and energy efficiency optimizations on microservers," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 355–368, July 2018.

[3] P. Gupta, "Accelerating datacenter workloads," in *26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016.

[4] J. Kong, "Datacenter storage system," Apr. 24 2014, uS Patent App. 13/694,001.

[5] C. Delimitrou, S. Sankar, K. Vaid, and C. Kozyrakis, "Decoupling datacenter studies from access to large-scale applications: A modeling approach for storage workloads," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2011, pp. 51–60.

[6] Y. Guo, Y. Gong, Y. Fang, P. P. Khargonekar, and X. Geng, "Energy and network aware workload management for sustainable data centers with thermal storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2030–2042, Aug 2014.

[7] L. Xu, A. R. Butt, S. Lim, and R. Kannan, "A heterogeneity-aware task scheduler for spark," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2018, pp. 245–256.

[8] "Apache hadoop," https://hadoop.apache.org, 2018.

[9] "Apache spark," https://spark.apache.org, 2018.

[10] "Hdd," https://en.wikipedia.org/wiki/Hard_disk_drive, 2018.

[11] "Ssd," https://en.wikipedia.org/wiki/Solid-state_drive, 2018.

[12] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 265–278.

[13] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with skewed content popularity in mapreduce clusters," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 287–300.

[14] C. L. Abad, Y. Lu, and R. H. Campbell, "Dare: Adaptive data replication for efficient cluster scheduling," in *2011 IEEE International Conference on Cluster Computing*, Sep. 2011, pp. 159–168.

[15] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 407–420.

[16] F. Pan, J. Xiong, Y. Shen, T. Wang, and D. Jiang, "H-scheduler: Storage-aware task scheduling for heterogeneous-storage spark clusters," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2018, pp. 1–9.

[17] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. New York, NY, USA: ACM, 2012, pp. 61–74.

[18] R. Gandhi, D. Xie, and Y. C. Hu, "PIKACHU: How to rebalance load in optimizing mapreduce on heterogeneous clusters," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 61–66.

[19] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010. [Online]. Available: https://aip.scitation.org/doi/abs/10.1109/MCSE.2010.69

[20] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, April 2010, pp. 1–9.

[21] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 61–74, Mar. 2012.

[22] "Hdfs," https://hadoop.apache.org/hdfs, 2018.

[23] B. Wang, J. Jiang, and G. Yang, "Actcap: Accelerating mapreduce on heterogeneous clusters with capability-aware data placement," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1328–1336.

[24] A. Davidson and A. Or, "Optimizing shuffle performance in spark," *University of California, Berkeley-Department of Electrical Engineering and Computer Sciences, Tech. Rep*, 2013.

[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] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas, "Crail: A high-performance i/o architecture for distributed data processing." *IEEE Data Eng. Bull.*, vol. 40, no. 1, pp. 38–49, 2017.

[27] D. Cobb and A. Huffman, "Nvm express and the pci express ssd revolution," in *Intel Developer Forum*. Intel, 2012.

[28] G. Beutler and E. Brockmann, "International gps service for geodynamics," in *Proceedings of the 1993 IGS Workshop*, vol. 369. Druckerei der Universitaet Bern, 1993.

[29] G. Berg, H. Koschitzky, A. Saguy, and O. Koschitzky, "System and method for analysis and management of logs and events," Feb. 22 2011, uS Patent 7,895,167.

[30] S. Shim, "Ssd storage device," 5 2019, uS Patent App. 29/628,078.

[31] A. Tsoukatos, T. Rausch, M. F. Erden, B. W. Parish, P. M. Ramakrishna, M. B. Tayefeh, S. S. Hon, C. Guo, T. K. Lim, S. W. Teo *et al.*, "High rpm hard disk drive testing," may 2019, uS Patent App. 16/058,819.

[32] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103.

[33] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, Oct 2001.

[34] G. Grimmett, G. Geoffrey, R., and S. David, "Policy gradient methods for reinforcement learning with function approximation," in *Probability and random processes*, 2000, pp. 1057–1063.

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

[36] “Google cluster trace,” https://code.google.com/p/googleclusterdata/, 2012.