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 these storage devices are quite different, traditional concurrent data fetching easily incurs unbalanced use among devices, leading to the straggler in terms of the task reading time as well as elongating the overall latency for 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), which essentially minimizes the maximal task reading time for parallel data analytical tasks. Afterwards, we 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 through our theoretical analysis. Extensive experiments show that rWASH reduces the average reading 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., some data streaming applications like Storm [1], and other machine learning based 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 the I/O performance while Grep is an I/O-intensive application which has high throughput on data processing. In order to meet these demands, a lot 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 heterogeneity of these devices easily leads to the divergent reading performance due to two main aspects. First, the different types of storage, e.g., SSD [10] and HDD [11], naturally results in different reading performance. As shown in Fig. 1, fetching the same amount of 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 reading 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 task reading time. More specifically, fetching data

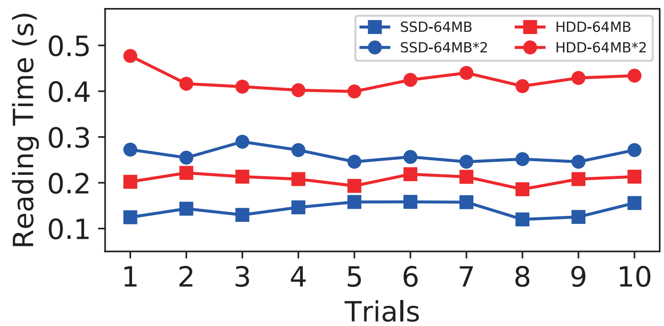


Fig. 1. Comparisons of (1) reading time for one piece of data with unified size (64MB) on various storage, i.e., SSD and HDD. (2) reading time for various I/O workloads, i.e., the number of data fetching concurrently.

from the devices with higher I/O performance actually earns shorter task reading time than that from those with lower I/O performance [16]. However, crowded I/O requests on a particular device unfortunately elongate the task reading time and correspond data analytics, which should be avoid as much as possible.

Thus, we propose to focus on dealing with unbalanced use on heterogeneous storage devices in this paper. However, due to the multiple replicas of data, how to choose target source from feasible devices 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 have already considered heterogeneous storage devices [7] [16], unbalanced I/O workloads and multiple data replicas are both not considered. In contrast, we formulate the Workload-Aware Scheduling problem for Heterogeneous storage devices. Afterwards, we propose a randomized algorithm (rWASH) with guaranteed performance by carefully choosing source device based on delicate calculated probabilities. More concretely, our contributions are as listed follows:

- To balance the I/O workloads on heterogeneous storage devices, we propose Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH).
- We propose randomized algorithm with guaranteed-performance (rWASH), which can be proved concentrated around its optimum with high probability, i.e., 1 - O(e^{-t^2}), where t is the concentration bound.
- The results of our extensive simulations show that our

rWASH improves by up to 55% over the art-of-the-state algorithm in terms of average tasks reading time.

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. For the first part, due to limited bandwidth, some works focus on data locality [12] [13] [14] [15] to reduce data transmission of data analytics. However, in heterogeneous scenario, it is not enough to consider data locality alone. Thus, some other studies [21] [7] [22] [23] have been carried out to accelerate data analytical tasks by considering hardware heterogeneity.

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, which keeps jobs waiting for a while until queuing delay reaches to a presetted 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 are repetitive and predictable, by which the scheduler could make scheduling plan ahead. All of these works improves the performance of data analytics by considering data locality. However, due to the fact that local disks might have high I/O workloads, previous strategies would elongate the reading time of local tasks.

Fetching Data Remotely. In heterogeneous cluster, data fetching remotely from those devices with higher performance on I/O 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 feature into consideration, which supported six 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 use among disks. Wang B [23] used Markov to model the use of nodes in the cluster to deploy data reasonably, but ignored the tasks reading time and the heterogeneity of devices.

Among these works, either the reading time of the tasks or the target disks is not considered. Thus, there is still a situation where a large number of tasks overload storage devices, causing bottlenecks. The difference between those works and ours is that our work specifically models the reading differences and the I/O workloads among disks. Then, each task chooses source disks based on delicate calculated probabilities, to speed up its reading time.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first introduce the background of heterogeneous storage devices used in data analytics and then build the system model. After that, we formulate the Workload-Aware Scheduling problem for heterogeneous devices.

A. Background and motivation

In 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 as a supplementary to speed up the Shuffle phase.

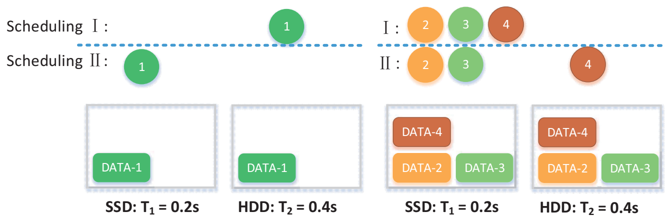
Data and replicas. In data analytics system, large volume of data is generated, such as GPS information [27], system logs [28], etc. Due to their massive sizes, a large file is often divided into multiple fixed-size pieces, i.e., one $data$ block, stored in a distributed file system such as HDFS [22], 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 straggler tasks. In contrast, our scheduler is then designed to avoid the stragglers by balancing I/O workloads among heterogeneous storage devices.

Motivation and example. We will use a simple example to illustrate the importance on choosing heterogeneous storage devices. As shown in Fig. 2, 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 = 0.2 and T_2 = 0.4, respectively.

- The traditional scheduler deploys tasks like scheduling I of Fig. 2(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



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

I : $max\{0.2s * 3,\ 0\}$,
II : $max\{0.2s * 2,\ 0.4s\}$.
(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 II, improves 50% and 33% over scheduling I, respectively.

from SSD has a shorter reading time than that from HDD, i.e., scheduling II is better than scheduling I.

- In Fig. 2(b), since the number of fetching data concurrently is 3 from SSD, the reading time of scheduling I is then $0.2s * 3 = 0.6s$, while the task reading 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 reading time than that from higher ones, i.e., scheduling II is better than scheduling I.

This simple case reveals two important findings: (1) The types of heterogeneous storage devices affect task reading time. (2) The I/O workloads of storage devices 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 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-$l$ 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 parallel tasks by data analytics framework, denoted by $\mathcal{T} = \{1, 2, ..., |\mathcal{T}|\}$. Each task-$j$ corresponds to an input data whose index is defined as ϕ_j. Since there are C replicas for each data stored in different C disks, task-j can read data-ϕ_j among 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 π_l^i as

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

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

Then, we denote by $load_i$ the reading 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.

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

For parallel tasks, if the scheduler is unaware of the different reading performance of heterogeneous disks, it would easily lead to the unbalanced use on disks, elongating the reading time of analytical tasks. To avoid such bottleneck, we propose Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH) whose goal is to minimize the maximal reading time. Detailed description is illustrated as follows:

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

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

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

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

Constraint (1) and Constraint (2) guarantee that task-j only fetches data-ϕ_j from one of those source disks storing its input. 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 [29].

IV. Design of Randomized Schema

The key to minimize the overall reading time for each task is to find the optimal source disk storing its input among heterogeneous disks. However, due to its inherent complexity of integral decisions for scheduling, the optimal solutions is hard to be obtained. Since it takes less time to read data from disks with higher I/O performance than that from those

Algorithm 1 WASH-greedy

Require: Task-j and its input data-ϕ_j, $\forall j$.

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

with lower I/O performance, we first explore a heuristic algorithm intuitively, which often chooses these disks with higher I/O performance, named WASH-greedy. However, due to its preference on high I/O performance disks, WASH-greedy may easily overload these disks, making them being bottlenecks. Furthermore, to solve WASH effectively as well as to avoid bottlenecks, we design a randomized algorithm (rWASH) 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. Greedy-based Inspiration

In this subsection, we first introduce WASH-greedy which selects source disk with minimal reading time, i.e. T_i, greedily.

Algorithm 1 shows the details of WASH-greedy. Line 1 initializes $Result = \{\}$. Lines 2-6 are used to select a source disk for each task-j. In line 3, function f is used to find the indexes of those disks which store the input data of task-j, i.e., data-ϕ_j, represented by $\{d_1^j,\ d_2^j, \ldots\ d_C^j\}$. Next, in line 4, the disk with minimal T_i is selected from those C disks. After $|\mathcal{T}|$ iterations, the selections of source disks for $|\mathcal{T}|$ tasks are completed. Then, each task will read its input data according to $Result$, as shown in line 7.

Next, we use an example to illustrate the process of WAHS-greedy. In data center, there exits four heterogeneous disks, i.e., $\mathcal{D}$= {1, 2, 3, 4}, with $T_1 = 0.2$, $T_2 = 0.25$, $T_3 = 0.4$ and $T_4 = 0.6$, respectively. Furthermore, five pieces of data are stored as shown in Fig. 3. When a job with five tasks comes, algorithm WASH-greedy runs as follows:

Initial: Result = {}

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

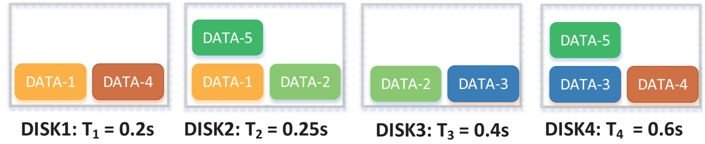


Fig. 3. Distribution of five pieces of data across 4 disks.

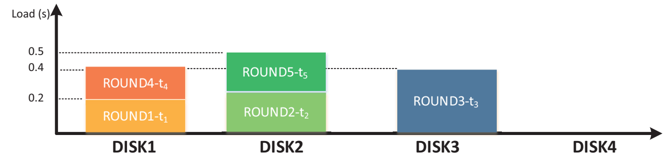


Fig. 4. The process of WASH-greedy for the data distributed in Fig. 3.

- **Round 5**:for task-5: f($\phi(5)$) = {2, 4}
  $2 = \arg\min\{T_2 = 0.25,\ T_4 = 0.6\}$
  Result $\cup \langle 5, 2\rangle = \{\langle 1, 1\rangle,\ \langle 2, 2\rangle,\ \langle 3, 3\rangle,\ \langle 4, 1\rangle,\ \langle 5, 2\rangle\}$

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

Due to the fact that WASH-greedy always shows its preference on the disks with higher I/O performance for each task, such algorithm may easily overload high performance disks. Thus, WASH-greedy fails to avoid the disks bottleneck, elongating the overall latency for data analytics.

B. Randomized task scheduling

In this subsection, we design a randomized algorithm, named rWASH, which avoids disk bottleneck by choosing source devices based on delicate calculated probabilities. 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. Subsequently, we prove that rWASH 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:*** A linear programming (LP) problem is solvable within polynomial time as well as shows a lower bound for the original 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:

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

Then, we use linear programming technique to solve WASH-relaxation whose solutions are fractions distributed

Algorithm 2 rWASH

Require: Task-j and its input data-ϕ_j, $\forall j$.

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** $\forall$ i, j, ($i \in \mathcal{D}$, $j \in \mathcal{T}$) **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 $Result$.

in [0, 1]. In a sense, the fraction solutions represent the preference when choosing the source disk for each task.

*b) r**WASH** schema:* Thus, our rWASH uses these fractions as probabilities to choose source disk for each disk. 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 2. 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 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 and $Pr[I_i^j = 1] = p_i^j$.

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 [30]. 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 will show that our rWASH 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. Firstly, we prove that the difference between disk-i's reading time contributed by any task-j and its expectation could be bounded through Martingale Analysis [31]. After that, we use Azuma's Inequality to illustrate the gap between the feasible solution and the optimal solution. For simplification, we use SOL to represent the feasible solution solved by rWASH, and OPT to represent the optimal solution of WASH problem in the following.

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 2, 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 all tasks, we denote by $L_i^{|\mathcal{T}|}$ the sum of the difference between Z_i^j and $E[Z_i^j]$, i.e., Q_i^j,

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

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 fact 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 [32]. For completeness, we 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

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

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

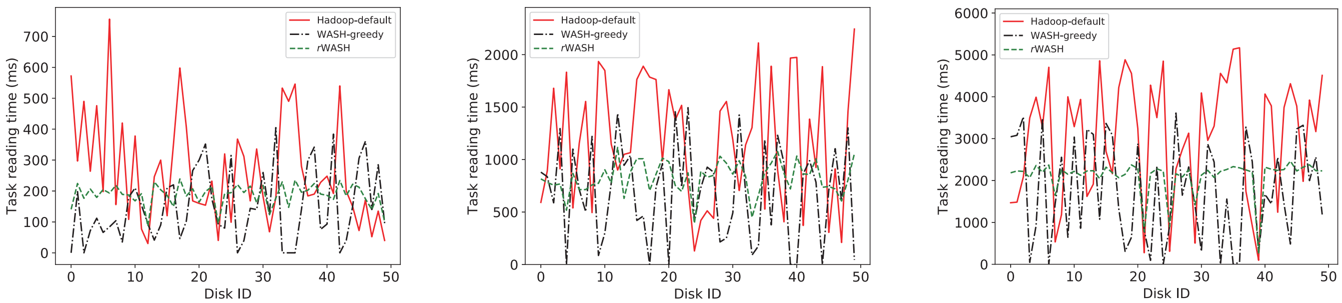
Equation (9a) and (9b) hold due to Equation (7) and Equation (6), respectively. For given p_i^j, T_i and $E[Z_i^r]$ are both constant. Thus, in Equation (10), the difference of any two consecutive items, i.e., L_i^r and L_i^{r-1}, $\forall r \geq 0$, in the martingale sequence, has a constant bound,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



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

Fig. 5. Characteristics of I/O workloads among 50 disks under different workloads. rWASH improves by up to 57%, 51% and 57% over Hadoop-default, as well as 15%, 23% and 35% over WASH-greedy in terms of the maximal task reading time for concurrent data fetching.

$$Pr\{\sum_{j=1}^{|\mathcal{T}|} Z_i^j \le \sum_{j=1}^{|\mathcal{T}|} E[Z_i^j] + t\} \ge 1 - exp\{-\frac{t^2}{2\sum_{i=1}^{|\mathcal{T}|}(g_i^k)^2}\}$$
$$= 1 - O(e^{-t^2}). \quad (12)$$

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

$$Pr\{S_i \le E_i + t\} \ge 1 - O(e^{-t^2}). \quad (13)$$

S_i represents the actual load of disk-i, and E_i denotes the solution of LP, i.e., the expectation load 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

$$E_x \le OPT. \quad (14)$$

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

$$S_u = S_{max} = \max_i S_i, \quad (15)$$
$$E_v = E_{max} = \max_i E_i. \quad (16)$$

Then, we have the following inequalities,

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

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

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

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. The results of our extensive simulations show that rWASH improves by up to 55% and 25%, 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.

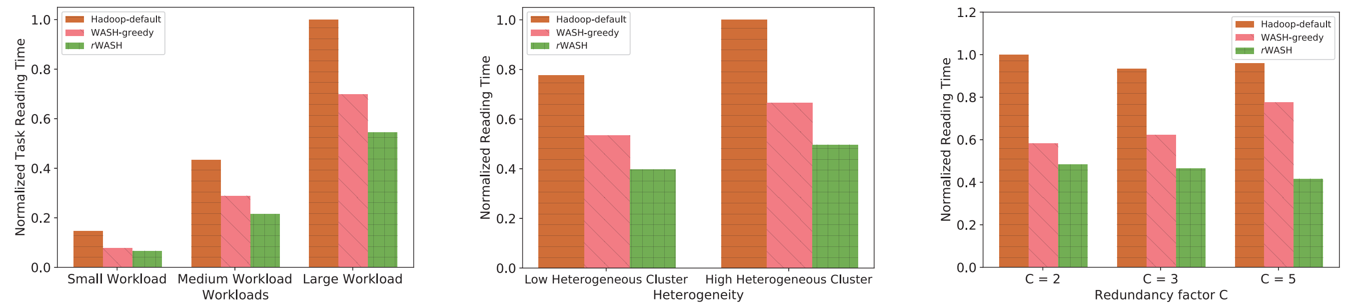
Workloads: According to the characteristics of Google traces [33], we classify the workloads into three categories: Small, Medium and Large. In small workload, most of the jobs have 1-150 tasks. In Large Workload, there are 50% of the jobs that own at least 500 tasks. And, the Medium Workload is in the 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 task reading time for only one piece of data, i.e., T_i, ranges from 10ms to 500ms, uniformly.

B. Simulation Results

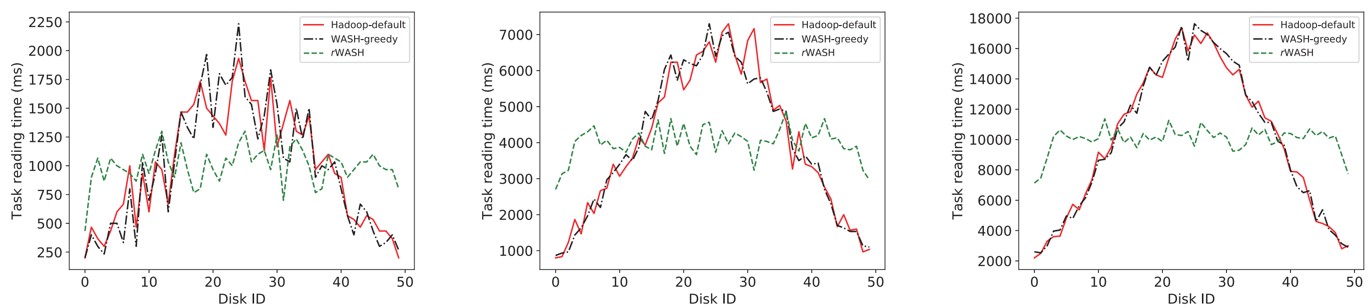
Characteristics of I/O workloads: Fig. 5 shows the results under various workloads among 50 disks. X-axis represents the IDs for different disk while Y-axis represents the task reading time. Note that the completion of concurrent data fetching relies on the maximal task reading time. The black dotted 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 task reading time. As shown in Fig. 5(a) further, i.e., in the small workload scenario, the result of Hadoop-default has large fluctuations since it is unaware of storage type and unbalance I/O workload among heterogeneous storage devices when selecting a source device for each task. As a result, the straggler elongates the completion time for concurrent data fetching, although the data is distributed uniformly among storage devices.

More specifically, the maximal task reading time under Hadoop-default is 756ms 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. 6. Results under various scenarios with different I/O workloads, I/O performance and the number of data replicas. rWASH improves 51%, 50% and 53% over Hadoop-default, as well as 20%, 26% and 30% over WASH-greedy in terms of the maximal task reading time.



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

Fig. 7. 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-greesy in terms of the maximal task reading time.

the overall latency of data analytics, as shown in Fig. 5(a). Although WASH-greedy reduces the maximal task reading time to 450ms, it ignores 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 worsen the task reading time due to the concurrent use. In contrast, rWASH selects the source devices based on delicate calculated probabilities with peak 343.6ms in disk-17. Compared with Hadoop-default and WASH-greedy, rWASH reduces reading time for tasks by up 57% and 40%, respectively.

Further shown in Fig. 5(b) and Fig. 5(c), with the increase number of concurrent data fetching, the maximal task reading time also increases among disks. When the number of tasks is about 2000 in middle workload scenario, the maximal task reading time of Hadoop-default and WASH-greedy is 2244ms and 1495ms, respectively. The maximal task reading time of rWASH is only 1116ms, which speeds up the concurrent data fetching by up to 51% and 15%, respectively. Furthermore, in large Workloads scenario, rWASH reduces reading time for tasks by up to 57% and 35% 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. 6(a), it summarizes the maximal task reading time for three I/O workloads, illustrating that the average improvement of rWASH compared with other two strategies is at least 20%. In Fig. 6(b), when the average I/O performance is low, i.e., the average reading time for only one data block from disks is 55ms in the scenario of low heterogeneity, the reduction of task reading time by rWASH compared with Hadoop-default and WASH-greedy is 49% and 26%, respectively. When the average I/O performance is high, i.e., the average reading time for only one data block from disks is 275ms in the scenario of high heterogeneity, the reduction by rWASH compared with Hadoop-default and WASH-greedy is 51% and 26%, respectively.

As shown in Fig. 6(c), with the growth on the number of data replicas, the maximal task reading time for Hadoop-default and rWASH decreases while the maximal task reading 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 elongating the maximal task reading 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 65% deduction on task reading time when the number of data replicas is 5.

Impact of data distribution: Fig. 7 shows the results for various I/O workloads under skewed data distribution.

More technically, the data is unbalanced stored among disks, which obeys Gaussian distribution [34]. The skewed data distribution actually influences the maximal task reading time because more data is crowded within a few disks. For Hadoop-default and WASH-greedy, the feasible choices of source disks become small, and hence more I/O workloads are crowded within these disks with high I/O performance, incurring high task reading 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.

In general, rWASH takes only several seconds for computation, i.e., the linear programming part in rWASH, which is acceptable for realistic deployment. The overall average improvement compared with rWASH and Hadoop-default as well as WASH-greedy is 55% and 25%, 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 task reading time for data analytics, the types of these storage devices and the I/O workloads should be both taken into consideration. Therefore, we formulate the Workload-Aware Scheduling problem for Heterogeneous storage devices (WASH), and design a randomized algorithm (rWASH). rWASH chooses source device for each task based on delicate calculated probabilities and can be proved concentrated on its optimum with high probability through our theoretical analysis. The results of our extensive simulations show that rWASH improves by up to 55% over the art-of-the-state 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] G. Beutler and E. Brockmann, "International gps service for geodynamics," in *Proceedings of the 1993 IGS Workshop*, vol. 369. Druckerei der Universitaet Bern, 1993.

[28] 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.

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

[30] 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.

[31] 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.

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

[33] "Google cluster trace," https://code.google.com/p/googleclusterdata/, 2012.

[34] C. Staelin and H. Garcia-Molina, *Clustering active disk data to improve disk performance*. Princeton University, Department of Computer Science, 1990.