

HTS:Research on Task Scheduling Technology for Heterogeneous Storage

*Note: Sub-titles are not captured in Xplore and should not be used

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

Abstract—Nowadays, a trend in data center is that there are more and more heterogeneous storage devices, i.e. SSD and HDD, which may differ greatly in performance. However, data processing frameworks such as MapReduce often considers that hardware is homogeneous and task scheduling is not aware of the difference between heterogeneous storage devices. In this case, if a large number of tasks are scheduled on low-speed storage nodes, it will greatly cause data processing bottlenecks. Furthermore, due to the different performance of each disk, it is difficult to make all disks have the balanced load. In this paper, we formulate the above problem as Heterogeneous Storage-aware Task Scheduling Problem (HTS), and show its NP-hardness. In order to solve this problem, we propose a heuristic algorithm and a random algorithm whose distance from the optimum value is within t with high probability. Experiments show that the our algorithm can reduce the task execution by up to 55% over the storage-unaware scheduling mechanism.

Index Terms—big data processing, heterogeneous storage, task scheduling

I. INTRODUCTION

Nowadays, in data centers that support big data processing platforms, i.e. Hadoop [14] and Spark [15], there are more and more heterogeneous devices. On the one hand, due to the purchase of hardware devices from different hardware providers, such as Seagate [16] and Samsung [17], the different standard hardware lead to heterogeneous; on the other hand, the use and maintenance of equipment may also lead to heterogeneous. However, the underlying heterogeneity of devices is often neglected in big data platforms, which results in the mismatch between upper software and lower hardware. The result is a waste of hardware resources or low performance of big data processing.

In order to improve the performance of big data processing, we focus on heterogeneous storage devices. A data processing workload, named a job, are usually divided into parallel tasks which scheduled on each node for execution. A task execution is usually divided into two parts, the first part is the data reading stage, and the other part is the task execution stage. On the one hand, the speed of the processor is much faster than the I/O, on the other hand, the node usually has multiple processors. The above two aspects lead to that data reading stage will be the bottleneck of task execution, i.e. in TABLE I. However, there are greatly differences in read performance between heterogeneous disks. Therefore, what we need to do is to select high-speed and low-load disks (If there are a large amount of tasks reading data on a high-speed disk at the same time, it may also cause bottlenecks in task execution) for tasks as much as possible.

However, there are two aspects which make it difficult to solve the problem. On the one hand, due to the different performance of heterogeneous disks, the same task might have different loads to the different disks. For example, the latency of read the same data m_i from DISK-1 and DISK-2 are 0.3s and 0.1s, respectively. If task t_i selects reading the data in DISK-1, the load to disk is 0.3 and selects DISK-2 is 0.1. On the other hand, because the data has been deployed on different disks before the task arrives, the task can only read data from fixed disks. For example, for HDFS data storage strategy, the redundancy factor of the data m_i is 3, when a task needs to read the data m_i , only the 3 disks storing replica of m_i can be selected. To avoid the bottleneck of reading during task execution, what we need to do is to balance the load of all disks. But the above two aspects make it difficult.

Previous studies on heterogeneity have also been conducted to improve the performance of data processing. However, most

of the research focuses on heterogeneous computing resources [21] [22], ignoring the heterogeneous storage resources.

In this paper, we focus on the data reading stage during the task execution based on the heterogeneous storage. There are two aspects that affect data reading in task execution, one is the read performance of the disk itself, the other is the load of the disk, that is, the number of tasks that are reading the data from the disk. Focus on the above two points, we formulate it as an optimization problem and propose a heuristic algorithm and a random algorithm to solve the problem. Experiments show that the performance of the proposed algorithm is 40% better than the Storage-unaware scheduling mechanism. More concretely, our contributions are as follows:

- We analyzed the impact of heterogeneous disks on task execution and formulate the heterogeneous storage-aware task scheduling problem which is proved to be NP-hard.
- We propose a heuristic algorithm, HTS-greedy, and a performance-guaranteed random algorithm, HTS-rdm, to solve this problem. The HTS-rdm algorithm whose distance from the optimum value is within t with high probability $1 - O(e^{-t^2})$ where t denotes the difference between solution found by the HTS-rdm and the optimal solution.
- We implemente the above two algorithm through simulation experiments. The results of experiment show that the performance of the proposed algorithm is improved by 40% compared with the storage-unaware scheduling mechanism.

The rest of the paper is organized as follows. The related works are presented in Section II. In Section II, we present the related works of the HTS 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.

TABLE I
COMPARISONS OF READ TIME AND EXECUTION TIME

For 640MB data	
Time of data reading stage	Time of Execution stage
3s	1.5s

II. RELATED WORKS

The related works consist of two parts. For the first part, due to limited link bandwidth, some work is focused on data locality. Furthermore, in heterogeneous environments, it is not enough to consider data locality alone and some other studies have been carried out to accelerate data processing by studying hardware heterogeneity [1].

Due to the limited network bandwidth resources in data center, if a large amount of data is transmitted between nodes during task execution, it will greatly affect the task performance. Therefore, processing data locally as much as possible can improve performance, i.e. 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 scheduling tasks with input data at that node, and delayed scheduling tasks without input data at that nodes. Ganesh [3] makes multiple replicas of high-frequency data by analyzing the access frequency of data, improve the data locality in this way. Cristina [4] proposes a distributed adaptive data replication algorithm DARE, which can help scheduler achieve better data locality. Jalaparti [5] believes that most of the production work is repetitive and predictable, which enables us to plan ahead. Coordinating and improving the tasks and data placement of these jobs can effectively reduce bandwidth usage and improve data locality. All of these work improves the performance of data processing by improving data locality.

However, in heterogeneous data centers, due to the different performance of data storage devices, task completion 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 does not perceive the type of disk where data is stored. 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 specifcily 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 will introduce the background of big data processing based on heterogeneous storage and build the system model. After that, the heterogeneous storage-aware task scheduling problem is formulated as a minimization problem whose objective is to minimize the maximum disk's reading time.

A. Background and motivation

A large number of jobs are running on big data processing platforms. During execution, jobs are divided into parallel tasks which scheduled to heterogeneous nodes. However, the

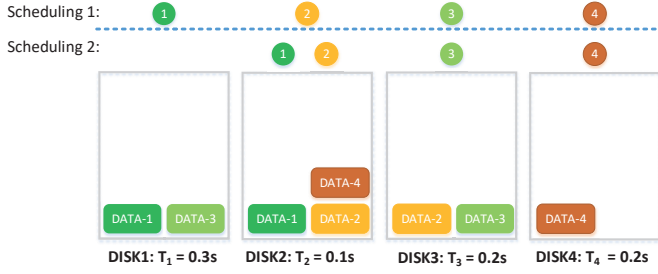


Fig. 1. Comparison of task time between general task scheduling and disk-aware task scheduling. Scheduling1 is unaware of disk *read-time* and the completion time of all tasks is $\max\{T_1 * N_1, T_2 * N_2, T_3 * N_3, T_4 * N_4\} = \max\{0.3*1, 0.1*1, 0.2*3, 0.2*1\} = 0.3$. Scheduling2 is aware of disk read time and the completion time of all tasks is $\max\{0, 0.1*2, 0.2*1, 0.2*1\} = 0.2$. Scheduling2 is the optimum.

general task scheduler is usually storage-unaware. The high-speed disk and the low-speed disk might be deployed the same number of tasks. Then, the tasks that read data from low-speed disk have a longer execution time. Specifically, they will be the bottleneck of job (The job doesn't finish until the completion of all tasks). Therefore, storage-unaware scheduling mechanism may cause the execution time of jobs too long to guarantee performance.

In order to improve the performance of data processing jobs, tasks should read data from high performance and low load disks. To specifically express the performance and load of the disk, we introduce parameter *read-time*. This value, i.e. 0.2s, is the time required to read a data replica from the disk. The smaller the value is, the better the disk performance has. Note that disks are heterogeneous, the *read-time* of each disk are different. If there are N tasks reading data from the disk, the load of the disk is *read-time* * N . In data centers, each data has C replicas which deployed on C disks whose *read-time* are different. When generating a large number of tasks, we consider deploying tasks to minimize the maximum disk load, thereby avoiding disk reading bottlenecks and speeding up data processing.

We will use an example to illustrate the task scheduling problem in heterogeneous storage. As shown in Fig.1, in the data center there are four disks, DISK1, DISK2, DISK3 and DISK4. Each disk has a different *read-time*, which are $T_1 = 0.3$, $T_2 = 0.1$, $T_3 = 0.2$ and $T_4 = 0.2$, respectively. The deployment of data replicas is shown in Fig.1. When tasks are deployed, a general scheduler may place tasks such as scheduling1, and the execution time of the task is $\max\{T_1 * N_1, T_2 * N_2, T_3 * N_3, T_4 * N_4\} = \max\{0.3*1, 0.1*1, 0.2*3, 0.2*1\} = 0.3$. Storage-aware task schedulers may place tasks as scheduling2, with a task execution time of 0.2. Obviously, the scheduling2 is optimum.

Example shows that when choosing the disk for a task, two things need to be considered: the first is the *read-time* of a disk, and the second is the load of the disk. These two things are able to ensure that there are no bottlenecks during the task execution and the data processing can be accelerated.

TABLE II
NOTATIONS USED IN THIS PAPER.

\mathbb{D}	The set of heterogeneous disks in datacenter
\mathcal{M}	The set of data in datacenter
\mathbb{T}	The set of tasks to be scheduled
d_i	The disk i and $d_i \in \mathbb{D}$
t_j	The task j and $t_j \in \mathbb{T}$
m_l	The data l and $m_j \in \mathbb{M}$
r_l^k	The data replica k of data m_l
<i>read-time</i>	This time of reading a data block from disk
T_i	The read-time of disk d_i
I_i^j	A binary variable indicating if task t_i choose the replica that stored in d_i as input or not
$\phi(t_j)$	The data that task t_j needs, and $\phi(t_j) \in \mathcal{M}$
π_l^i	A binary variable indicating if data m_l are stored at disk d_i or not
h_{ul}, h_{dl}	the channel fading coefficient for uplink and downlink
$\phi(t_j)$	The data that task t_j needs
$N(i)$	Number of tasks which select data from disk i as input
τ	Size of each data replica in datacenter
C	The number of replicas of each data.

B. System Model

The notations we need are shown in the table II. The data center is equipped with heterogeneous disks, and \mathbb{D} is used to represent the set of all disks which are $d_1, d_2, \dots, d_{|\mathbb{D}|}$. \mathcal{M} denotes the set of all the data in data center, and each data is represented by $m_1, m_2, \dots, m_{|\mathcal{M}|}$. Each data m_l has C replicas which are $r_l^1, r_l^2, \dots, r_l^C$ stored on C different disks, with each data replica's size τ , i.e. 64MB or 128MB. If the disk d_i contains a replica of the data m_l , $\pi_l^i = 1$. Otherwise, $\pi_l^i = 0$. In datacenter with heterogeneous disks, each disk has its own *read-time* represented by T_i , and the value of T_i represents the time required to read a data replica from disk t_i .

When the query job arrives, the job will be divided into parallel tasks. \mathbb{T} denotes the set of all tasks which include $\{t_1, t_2, \dots, t_T\}$. Each task t_j corresponds to a unique input data $\phi(t_j)$. Since there are C replicas of each data deployed in C disks, the replicas of $\phi(t_j)$ exists in C disks, assuming that it is $d_{j1}, d_{j2}, \dots, d_{j|C|}$. So t_j can read $\phi(t_j)$ from one of C disks.

Next, if the task t_j chooses the replica of data $\phi(t_j)$ stored in disk d_i , then $I_i^j = 1$. In this case, the time of task t_j reading the replica from disk d_j is $I_i^j * T_i$. There may be some other tasks that select replicas stored in disk d_i , so the load of disk d_i is $\sum_j I_i^j * T_i$. In order to avoid a large number of tasks to choose the same disk, resulting in data processing bottlenecks. Our objective is to minimize the maximum disk load.

C. Heterogeneous Storage-aware Task Scheduling Problem Formulation (HTS)

A large number of tasks are running in the cluster of data centers. If the scheduler are unaware of the heterogeneity of disks, it is likely that there are a large number of tasks that read data from the same low-speed disk, resulting in bottlenecks. To

avoid this situation, we propose heterogeneous storage-aware task scheduling (HTS), 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:

$$\begin{aligned}
Min : \quad & \max_i \left\{ \sum_j I_i^j * T_i \right\} [\text{HTS}] \\
s.t. \quad & \sum_i I_i^j = 1, \text{ for } \forall j \quad (1) \\
& I_i^j \leq \pi_{\phi(t_j)}^{d_i}, \text{ for } \forall i, j \quad (2) \\
& I_i^j \in \{0, 1\}, \text{ for } \forall i, j \quad (3)
\end{aligned}$$

Constraint (1) indicates that task t_j can only select one disk to read data $\phi(t_j)$. Constraint (2) indicates task t_j can only read its input data from the disks which stores its input data. If one replica of data $\phi(t_j)$ is stored in disk d_i , then $\pi_{\phi(t_j)}^{d_i} = 1$, otherwise $\pi_{\phi(t_j)}^{d_i} = 0$. Constraint 3 denotes the range of decision variables, which can only be 0 or 1. $I_i^j = 1$ indicates that task t_j selects the replica stored on disk d_i as input, while task $t_j = 0$ is on the contrary.

The key to solve HTS problem is to determine the value of decision variables I_i^j . Next, we analyze the hardness of this problem and prove that the HTS problem is NP-hard. HTS problem can be reduced from the integer linear programming (ILP) problem. The ILP problem is NP-hard, and it follows that HTS problem is NP-hard. The specific proof process is as follows.

Theorem 1: The HTS problem is NP-hard.

Proof: The canonical form [11] of ILP is described below.

For $m \times n$ matrix \mathbf{A}

$$\begin{aligned}
Min : \quad & \mathbf{c}^T \mathbf{x} \quad [\text{ILP}] \\
s.t. \quad & \mathbf{A} \mathbf{x} \geq \mathbf{b} \\
& \mathbf{x} \geq 0, \\
\text{and} \quad & \mathbf{x} \in \mathbb{Z}^n
\end{aligned} \quad (4)$$

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

- $\mathbb{D} = \{d_0, d_1, d_2\}; T_0 = 1, T_1 = 1, T_2 = 1; \mathbb{M} = \{m_0, m_1, \dots, m_{n-1}\}; C = 3$
- $T = \{t_0, t_1, \dots, t_{n-1}\}$

It means that there are 3 disks, d_0, d_1, d_2 , in the data center, each disk with a *read-time* 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, t_0, t_1, \dots, t_n to read n data, m_0, m_1, \dots, m_n , respectively.

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

$$\begin{aligned}
Min : \quad & \max \left\{ \sum_j I_0^j, \sum_j I_1^j, \sum_j I_2^j \right\} [\text{HTS} - \text{sp}] \quad (5) \\
s.t. \quad & I_0^j + I_1^j + I_2^j = 1, \text{ for } \forall j \\
& I_0^j, I_1^j, I_2^j \in \{0, 1\}, \text{ for } \forall j \in [0, n)
\end{aligned}$$

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:

$$Min : \quad \max \{x, y, z\} [\text{HTS} - \text{sp}] \quad (6)$$

$$s.t. \quad x + y + z = n \quad (7)$$

$$0 \leq x, y, z \leq n \quad x, y, z \in \mathbb{Z}$$

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, HTS-sp problem is converted to:

$$Min : \quad 0 * x + 0 * y + 0 * z + R [\text{HTS} - \text{ILP}] \quad (8)$$

$$s.t. \quad x + y + z \geq n$$

$$-x + R \geq 0$$

$$-y + R \geq 0$$

$$-z + R \geq 0$$

$$0 \leq x, y, z \leq n \quad x, y, z \in \mathbb{Z}$$

$$R \geq 0, R \in \mathbb{Z}$$

Obviously, HTS-ILP in (8) is an instance of ILP in (4). 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 HTS-ILP problem is NP-hard.

On the one hand, when the optimal solution of HTS-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 $\{I_0^0, I_0^1, \dots, I_0^{x-1}, I_1^x, I_1^{x+1}, \dots, I_1^{x+y-1}, I_2^{x+y}, I_2^{x+y+1}, \dots, I_2^{x+y+z-1}\}$ to be 1, which will make the corresponding HTS-sp problem obtain the optimal solution. On the other hand, when the variable $\{I_i^j\}$ make the HTS-sp problem obtain the optimal solution, the HTS-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 HTS-sp and HTS-ILP. Therefore, HTS-sp is NP-hard.

As a result of that HTS-sp is a special case in HTS, HTS problem is NP-hard. \square

IV. DESIGN OF ALGORITHMS FOR HTS PROBLEM

Due to the hardness of HTS problem, the optimal solution can not be obtained in polynomial time. In order to solve NP-hard problem, we propose a heuristic algorithm based on greedy idea, named HTS-greedy. However, this kind of algorithm usually has no guarantee of quality, although in some cases the solution is not bad, in some cases it is opposite. Furthermore, we design a quality assurance algorithm HTS-rdm, which makes the feasible solution approach to the optimal solution with probability $1 - O(e^{-t^2})$ where t is the difference between the solution found by HTS-rdm and optimal solution.

A. Heuristic Algorithm

In this subsection, we will introduce the heuristic algorithm HTS-greedy specifcily. Using the greedy idea, the HTS-greedy chooses the disk which stores $\phi(t_j)$ for each task t_j and which has the minimum load, and finally outputs one disk for each task.

Algorithm 1 HTS-greedy

Require: For each task t_j , decide a disk d_i that stores one replica of data $\phi(t_j)$.

```

1: for each disk  $d_i$  do
2:    $load_i \leftarrow 0$ 
3: end for
4:  $Result \leftarrow \{\}$ 
5: for each task  $t_j$  do
6:    $d_{j1}, d_{j2}, \dots, d_{j|C|} = f(\phi(t_j))$ 
      //Function f is a mapping from data to disks that
      stores each replica of the data.
7:    $load_{jmin} \leftarrow \min(load_{j1})$ 
       $load_{jmin} \leftarrow load_{jmin} + T_{jmin}$ 
8:    $Result \leftarrow Result \cup \{ \langle t_j, d_{jmin} \rangle \}$ 
9: end for
10: Return Result

```

Algorithm 1 shows the detailed of HTS-greedy. Line 1-4 initializes $load_i = 0$ and $Result = \{\}$, and $load_i$ denotes the total *read-time* of the $disk_i$. Line 5-9 is to select a disk for each task t_j . In line 6, function f is used to help task t_j find all disks that store replicas of data $\phi(t_j)$. Let $d_{j1}, d_{j2}, \dots, d_{j|C|}$ denote the C disks. Next, in line 7, the disk with minimum total *read-time* is selected from the C disks, and the *read-time* caused by task t_j is added to the $load_i$. Then put the task t_j and disk d_{j1} into the set $Result$. After one iteration, the algorithm completes the decision for a task and it needs \mathbb{T} iteration in total.

Next, an example is given to illustrate the algorithm. In datacenter, there exists a set of heterogeneous disks $\mathbb{D} = \{d_1, d_2, d_3, d_4\}$ with *read-time* $T_1 = 0.3, T_2 = 0.1, T_3 = 0.2$ and $T_4 = 0.2$, respectively. Data set $\mathbb{M} = \{m_1, m_2, m_3, m_4\}$ are stored as Fig.1. Obviously, each data has two replicas. When query tasks $\mathbb{T} = \{t_1, t_2, t_3, t_4\}$ ($\phi(t_j) = j, 1 \leq j \leq 4$) comes, the algorithm HTS-greedy runs as follows:

Initial: $Result = \{\}, load_1=0, load_2=0, load_3=0, load_4=0$

- **Round 1:** for task t_1 : $f(\phi(t_1)) = \{1, 2\} = \{DISK1, DISK2\}$
 $load_1 = \min\{load_1 = 0, load_2 = 0\}$
 $load_1 = load_1 + T_1 = 0.3$
 $Result = Result \cup \langle t_1, d_1 \rangle = \{\langle t_1, d_1 \rangle\}$
- **Round 2:** for task t_2 : $f(\phi(t_2)) = \{2, 3\}$
 $load_2 = \min\{load_2 = 0, load_3 = 0\}$
 $load_2 = load_2 + T_2 = 0.1$
 $Result = Result \cup \langle t_2, d_2 \rangle = \{\langle t_1, d_1 \rangle, \langle t_2, d_2 \rangle\}$
- **Round 3:** for task t_3 : $f(\phi(t_3)) = \{1, 3\}$
 $load_3 = \min\{load_1 = 0.3, load_3 = 0\}$

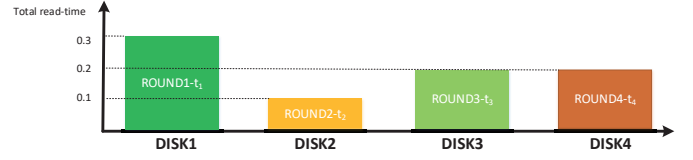


Fig. 2. The process of algorithm HTS-greedy execution for the input in Fig.1.

$load_3 = load_3 + T_3 = 0.2$

$Result = Result \cup \langle t_3, d_3 \rangle = \{\langle t_1, d_1 \rangle, \langle t_2, d_2 \rangle, \langle t_3, d_3 \rangle\}$

- **Round 4:** for task t_4 : $f(\phi(t_4)) = \{2, 4\}$

$load_4 = \min\{load_2 = 0.1, load_4 = 0\}$

$load_4 = load_4 + T_4 = 0.2$

$Result = Result \cup \langle t_4, d_4 \rangle = \{\langle t_1, d_1 \rangle, \langle t_2, d_2 \rangle, \langle t_3, d_3 \rangle, \langle t_4, d_4 \rangle\}$

The final result is in $Result$ and Tasks will follow $\{\langle t_1, d_1 \rangle, \langle t_2, d_2 \rangle, \langle t_3, d_3 \rangle, \langle t_4, d_4 \rangle\}$ to read data. The specific process of HTS-greedy algorithm is shown in Fig.2.

B. Randomized Algorithm

In the previous section IV-A, we designed a greedy heuristic algorithm HTS-greedy to solve HTS problems. However, this algorithm does not have performance guarantee. Therefore, in this subsection, we propose a performance-guaranteed random algorithm HTS-rdm. By relaxing the HTS problem to HTS-relaxation problem, HTS-rdm uses linear programming to solve HTS-relaxation. The decomposition obtained by linear programming is a fraction rather than an integer, which represents the preference when selecting disks. Subsequently, we prove that the solution found by the HTS-rdm algorithm can approach the optimal solution with probability $1 - O(e^{-t^2})$ where t is the difference between the solution found by HTS-rdm and optimal solution..

a) **Relaxation of HTS Problem:** According to the previous analysis III-C, this problem is NP-hard, which cannot be solved in polynomial time. But the linear programming (LP) problem is solvable in polynomial time, a natural point of view is to relax the HTS problem to LP. The method is to change the range of variables from integer domain $\{0, 1\}$ to real domain $[0, 1]$. As shown below, HTS-relaxation, this process is named **relaxation**. The solution of HTS-relaxation provides a lower bound for the original HTS problem (HTS 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**. Based on relaxation-rounding, we propose HTS-rdm algorithm.

$$\begin{aligned}
 Min : \quad & \max_i \left\{ \sum_j p_i^j * T_i \right\} \text{ [HTS - relaxation]} \\
 s.t. \quad & \sum_i p_i^j = 1, \text{ for } \forall j \\
 & p_i^j \leq \pi_{\phi(t_j)}^{d_i}, \text{ for } \forall i, j \\
 & p_i^j \in [0, 1], \text{ for } \forall i, j
 \end{aligned}$$

The detailed procedure of HTS-rdm algorithm is shown in Algorithm 2. Line 1 initializes the Result set, which stores the final result. In the second line, the HTS-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 and output them. Then the task will read the disk according to the results. The specific rounding strategies are as follows:

The value of $\{p_i^j\} \in [0, 1]$ which shows the correlation between task t_j and disk d_i . Therefore, we can select disk d_i for task t_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.

Algorithm 2 HTS-rdm

Require: Decide a disk d_i for each task t_j .

- 1: Result $\leftarrow \{\}$
 - 2: $\{p_i^j\}$ = HTS-relaxation // is the solution of HTS-relaxation problem.
 - 3: Based on $\{p_i^j\}$, use rounding strategy and get $\{I_i^j\}$
 - 4: **for** $\forall i, j$ **do**
 - 5: **if** $I_i^j == 1$ **then**
 - 6: Result \leftarrow Result $\cup \{ \langle t_j, d_i \rangle \}$
 - 7: **end if**
 - 8: **end for**
 - 9: **Return** Result
-

b) Analysis of HTS-rdm Algorithm: In this section, we will prove that the HTS-rdm algorithm can approach the optimal solution with a probability $1 - O(e^{-t^2})$. Firstly, we prove that the difference between *read-time* contributed by any tasks t_j to d_i and expectation is matigales sequence [12]. Secondly, Based on the matigales sequence, we use Azuma inequality to illustrate the bound between the feasible solution and the optimal solution.

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

Proof:

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

Firstly, task t_j 's contribution to disk d_i 's load is expressed as:

$$Z_i^j = I_i^j * T_i \quad (9)$$

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

$$\Pr[I_i^j = 1] = p_i^j, \text{ for } \forall i, j$$

The expectation of Z_i^j is

$$\begin{aligned} E[Z_i^j] &= E[I_i^j] * T_i \\ &= (\Pr[I_i^j = 1] * 1 + \Pr[I_i^j = 0] * 0) * T_i \\ &= p_i^j * T_i \end{aligned} \quad (10)$$

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

$$Q_i^j = Z_i^j - E[Z_i^j] \quad (11)$$

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

$$L_i^{|\mathbb{T}|} = \sum_{j=1}^{|\mathbb{T}|} Q_i^j = \sum_{j=1}^{|\mathbb{T}|-1} L_i^j + Q_i^{|\mathbb{T}|} \quad (12)$$

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

$$\begin{aligned} &E[L_i^r | L_i^1, L_i^2, \dots, L_i^{r-1}] \\ &\stackrel{(12)}{=} E[L_i^{r-1} + Q_i^r | L_i^1, L_i^2, \dots, L_i^{r-1}] \\ &= E[L_i^{r-1} | L_i^1, L_i^2, \dots, L_i^{r-1}] + E[Q_i^r | L_i^1, L_i^2, \dots, L_i^{r-1}] \\ &\stackrel{(11)}{=} L_i^{r-1} + E[Z_i^r - E[Z_i^r] | L_i^1, L_i^2, \dots, L_i^{r-1}] \\ &= L_i^{r-1} + E[Z_i^r | L_i^1, L_i^2, \dots, L_i^{r-1}] - E[E[Z_i^r] | L_i^1, L_i^2, \dots, L_i^{r-1}] \\ &= L_i^{r-1} + E[Z_i^r] - E[Z_i^r] \\ &= L_i^{r-1} \end{aligned} \quad (13)$$

Therefore, $L_i^1, L_i^2, \dots, L_i^{|\mathbb{T}|}$ are matigales sequence [13]. For completeness, we let $L_i^0 = 0$. And for $\forall r \geq 1$, we have:

$$|L_i^r - L_i^{r-1}| \stackrel{(12)}{=} |Q_i^r| \stackrel{(11)}{=} |Z_i^r - E[Z_i^r]| \leq g_i^r \quad (14)$$

$$g_i^r = \max\{T_i - E[Z_i^r], E[Z_i^r]\} \quad (15)$$

From above (15), any two adjacent values L_i^r, L_i^{r-1} in the margingales sequence have constant bounds g_i^r . Based on (13) and (15), we can use Azuma's inequality. Then,

$$\Pr\{L_i^{|\mathbb{T}|} - L_i^0 \geq t\} \leq \exp\left\{-\frac{t^2}{2 \sum_{i=1}^{|\mathbb{T}|} (g_i^k)^2}\right\} \quad (16)$$

Substitute equations (11) and (12) into the upper equation (16). Then,

$$\Pr\left\{\sum_{j=1}^{|\mathbb{T}|} Z_i^j - \sum_{j=1}^{|\mathbb{T}|} E[Z_i^j] \geq t\right\} \leq \exp\left\{-\frac{t^2}{2 \sum_{i=1}^{|\mathbb{T}|} (g_i^k)^2}\right\} \quad (17)$$

The (17) is equal to:

$$\begin{aligned} \Pr\left\{\sum_{j=1}^{|\mathbb{T}|} Z_i^j \leq \sum_{j=1}^{|\mathbb{T}|} E[Z_i^j] + t\right\} &\geq 1 - \exp\left\{-\frac{t^2}{2 \sum_{i=1}^{|\mathbb{T}|} (g_i^k)^2}\right\} \\ &= 1 - O(e^{-t^2}) \end{aligned} \quad (18)$$

Let $S_i = \sum_{j=1}^{|\mathbb{T}|} Z_i^j$, $E_i = \sum_{j=1}^{|\mathbb{T}|} E[Z_i^j] \stackrel{(10)}{=} \sum_{j=1}^{|\mathbb{T}|} p_i^j * T_i$. Then,

$$\Pr\{S_i \leq U_i + t\} \geq 1 - O(e^{-t^2}) \quad (19)$$

S_i denotes the load of disk d_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 (HTS is a minimization problem. For the maximization problem is on the contrary), we have:

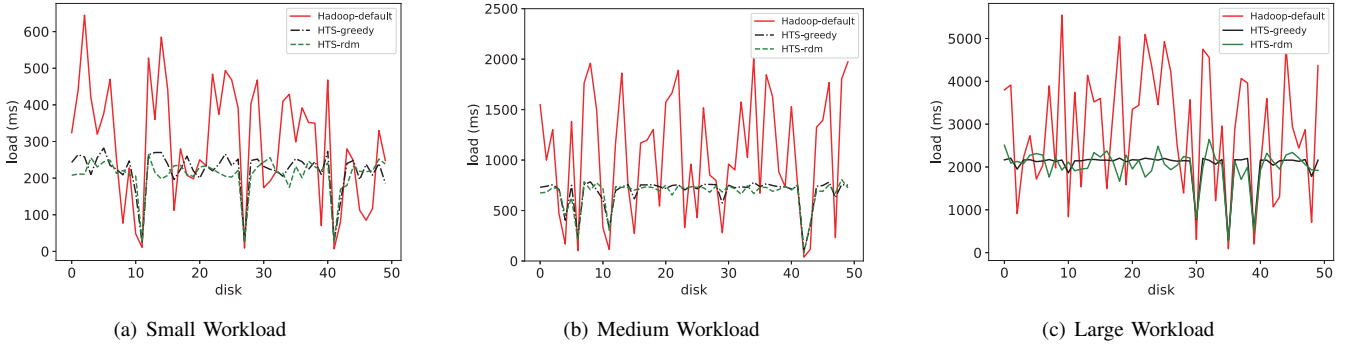


Fig. 3. Comparison of HTS-rdm and other Algorithm on different Workloads. 3(a) denotes the results on Small Workload which has about 500 tasks. 3(b) denotes the results on Medium Workload which has about 2000 tasks. 3(c) denotes the results on Large Workload which has 5000 tasks. X-axis denotes the disk. Y-axis denotes the load of each disk after running the three algorithms.

$$E_i \leq OPT(HTS) \quad (20)$$

Take S_u, E_v as

$$S_u = S_{max} = \max_i S_i \quad (21)$$

$$E_v = E_{max} = \max_i E_i \quad (22)$$

Then, we have the following inequalities,

$$\begin{aligned} SOL(HTS - rdm) = S_u &\stackrel{(18)}{\leq} E_u + t \stackrel{(22)}{\leq} E_v + t \\ &\stackrel{(20)}{\leq} OPT(HTS) + t \end{aligned} \quad (23)$$

From (19) and (23), we can conclude that,

$$Pr\{SOL(HTS - rdm) < OPT(HTS) + t\} \geq 1 - O(e^{-t^2}) \quad (24)$$

Observed from the (24), the feasible solution $SOL(HTS - rdm)$ found by the algorithm and the optimal solution $OPT(HTS)$ 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. \square

V. PERFORMANCE EVALUATION

In this section, we implement our algorithms HTS-greedy and HTS-rdm, and compare our algorithms with storage-unaware scheduling algorithm. The results show that performance can be improved by 40%.

A. Simulation Settings

Usually, the general scheduling mechanism can not perceive 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 III. In Small Workload, most of the jobs have 1-150 tasks. In has Large Workload, there are 50% of the jobs that

TABLE III
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%

has over 50 tasks. The Medium Workload is in the middle of them.

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 $\tau = 128\text{MB}$. 500 disks are set up by default. The *read-time* of the disk in the cluster is between 10 and 500 in ms.

B. Simulation Results

For convenience of display, Fig.3. 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 HTS-greedy, and the green line represents the execution result of Algorithm HTS-rdm. Overall, we can see that our algorithm performs better than Hadoop-default. As shown in Fig.3(a), in the small workload, the result of Hadoop-default has large fluctuations. This is because hadoop-default does not perceive the *read-time* 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. 3(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.3(a) which is 654. Algorithms HTS-greedy and HTS-rdm can maintain roughly all disks at a lower load, less than 300. The reason is that HTS-greedy

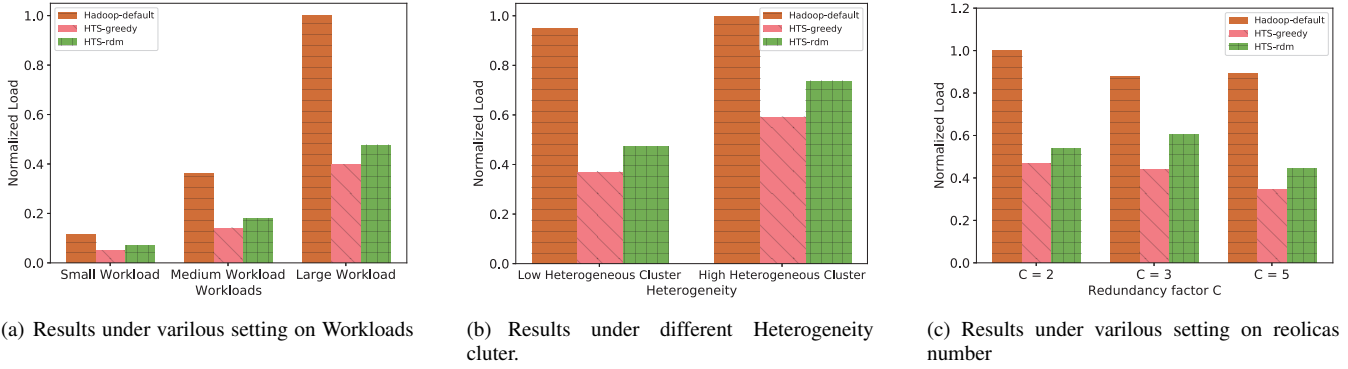


Fig. 4. Comparison of HTS-rdm and other Algorithm on different workloads, different heterogeneity cluster and different replica number C . 4(a) denotes effect of different workloads on completion time. 4(b) denotes the completion time of the three algorithm on different Heterogeneity cluster. 4(c) denotes the completion time of the three algorithm when C is different. Y-axis denotes the completion time of all tasks which has been normalized.

can select the disk to read for the task according to load of the disks and HTS-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 HTS-greedy algorithm is 261 in disk-12. Compared with the Hadoop-default, the performance is improved by 64%. The maximum load of HTS-rdm algorithm is disk-4, and the corresponding load is 284. Compared with the baseline algorithm, the performance of HTS-rdm algorithm improves 57%. As the number of tasks increases, the load on all disks increases. For example, in Fig.3(b), when the number of tasks is about 2000, in Hadoop-default algorithm the maximum load is 2009 in disk-9. The results of HTS-greedy and HTS-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. 3(c), the maximum load obtained by the three algorithms are 5546, 2205 and 2646. HTS-greedy increases by 60% and HTS-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.4(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.4(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 *read-time* of disks is evenly distributed between 10ms and 100ms, named Low Heterogeneous Cluster. Another is High Heterogeneous Cluster, where the *read-time* of disks is evenly distributed between 10ms and 500ms. Other conditions are configured by default. From Fig.4(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 4(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

There is a trend of more and more heterogeneous disks in data centers. Due to the heterogeneity, a mismatch occurred between the upper software and the lower hardware, which leads to the low performance of big data analysis. To minimize the latency of big data analysis. Based on this, we first formulate the heterogeneous storage awareness task scheduling problem, and prove the hardness of the problem. Then we propose two effective algorithms, heuristic algorithm and quality assurance random algorithm, to solve this problem. Experiments show that our algorithm speeds up the average completion time by 55%.

ACKNOWLEDGMENT

None

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] Hromkovi 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_\(probability_theory\)](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.