LDBAS: Location-aware Data Block Allocation Strategy for HDFS-based Applications in the Cloud

Hua Xu, Weiqing Liu, Guansheng Shu and Jing Li
Department of Computer Science and Technology,
University of Science and Technology of China
Hefei, Anhui, 230026 - China
[e-mail: {boxwh, cslwqxx, sgs2012}@mail.ustc.edu.cn, lj@ustc.edu.cn]
*Corresponding author: Jing Li

Received December 12, 2016; revised April 11, 2017; revised June 2, 2017; accepted November 9, 2017; published January 31, 2018

Abstract

Big data processing applications have been migrated into cloud gradually, due to the advantages of cloud computing. Hadoop Distributed File System (HDFS) is one of the fundamental support systems for big data processing on MapReduce-like frameworks, such as Hadoop and Spark. Since HDFS is not aware of the co-location of virtual machines in the cloud, the default scheme of block allocation in HDFS does not fit well in the cloud environments behaving in two aspects: data reliability loss and performance degradation. In this paper, we present a novel location-aware data block allocation strategy (LDBAS). LDBAS jointly optimizes data reliability and performance for upper-layer applications by allocating data blocks according to the locations and different processing capacities of virtual nodes in the cloud. We apply LDBAS to two stages of data allocation of HDFS in the cloud (the initial data allocation and data recovery), and design the corresponding algorithms. Finally, we implement LDBAS into an actual Hadoop cluster and evaluate the performance with the benchmark suite BigDataBench. The experimental results show that LDBAS can guarantee the designed data reliability while reducing the job execution time of the I/O-intensive applications in Hadoop by 8.9% on average and up to 11.2% compared with the original Hadoop in the cloud.

Keywords: Hadoop, HDFS, big data processing, data block allocation, cloud

A preliminary version of this paper appeared in IEEE CLOUD 2016, June 26-July 2, New York, USA. This version includes an algorithm for data recovery problem and more comprehensive experimental evaluation. The research is funded by the National Hi-Tech Research and Development Program of China under Grant 2014AA01A302, the CERNET Innovation Project under the contract No. NGII20150110, and the National Key Research and Development Program under 2016YFB0201402. We are also grateful to the Network Information Center of University of Science and Technology of China for the support of hardware devices.

http://doi.org/10.3837/tiis.2018.01.010 **ISSN : 1976-7277**

1. Introduction

In recent years, data has grown in an explosive way and in various fields. There is an increasing demand for data processing systems to perform large-scale data analytics. Since traditional data processing systems have a restriction of scalability for massive data [1][23], the MapReduce model [2] was proposed to handle big data problem efficiently. Especially, Hadoop [3], the popular open-source implementation, has been widely used in industry and academia. However, the economic cost and maintenance cost of physical infrastructure for deploying Hadoop is prohibitive for some middle-small enterprises or individuals. As a result, there is a growing interest in running Hadoop in the cloud, due to the advantages of cloud computing, such as on-demand self-service and rapid elasticity [4]. For example, Amazon has provided the service of Hadoop in the cloud to the customers through EMR [5].

Hadoop Distributed File System (HDFS) [8] is one of the fundamental support systems of MapReduce-like frameworks [10], such as Hadoop [3], Spark [9], etc. It is designed to store large-scale data sets reliably and to stream them at high bandwidth to upper-layer applications. The key to achieving these goals is an excellent strategy of data block allocation that consists of two stages in HDFS. One stage is the initial data allocation, which is responsible for the task of allocating data blocks of new files into HDFS. The other stage is the data recovery, which is in charge of reallocating the missing data blocks when some nodes break down.

The default scheme of data allocation in HDFS performs well in the physical environments. However, it does not perform well in the cloud environments. This is because HDFS is not aware of the situation that multiple virtual machines (VMs) may co-locate on the same physical machine (PM). It still treats the virtual nodes as physical nodes, which brings two serious problems: data reliability loss and performance degradation.

- **Data reliability loss.** HDFS achieves the objective of storing massive data reliably by placing multiple replicas of a data block in different physical nodes. In the cloud environments, though different replicas of one data block are allocated to different virtual nodes by default, the virtual nodes with different replicas of the same data block may co-locate on the same physical machine. It will harm the data reliability and raise the possibility of data loss when the physical host crashes [6].
- **Performance degradation.** In the cloud, the processing capacities (PCs) of VMs with the same configuration may be actually unequal among different PMs (especially for the disk I/O bandwidth), due to the competition of hardware resources among the co-located VMs [7]. Meanwhile, Hadoop is designed for the homogeneous environments, and it allocates data blocks almost evenly across different nodes. It will cause unbalanced processing loads for the nodes with different PCs. Once the nodes with higher PCs finish processing the data blocks at local, they have to execute excessive remote tasks (known as poor *data locality*). Since network bandwidth is lower than local disk I/O bandwidth in most cases, remote tasks are usually less efficient than local tasks, and may lead to plenty of unnecessary network traffic. Hence, the unbalanced processing loads would severely decrease the performance of HDFS-based applications in the cloud.

To solve these two problems, a straightforward solution is to avoid the co-location of VMs. However, it will decrease the resource utilization and harm the economic profit of cloud providers. Hence, some other schemes are proposed from the aspects of task scheduling or data block allocation. New task scheduling [7] detected and tackled stragglers more accurately,

eventually mitigated the performance degradation for Hadoop in heterogeneous environments. However, this category of approaches cannot solve the problem of data reliability loss from the aspect of improving task scheduling. The block allocation strategy [6] was proposed to address above two problems. Nevertheless, they only considered the scenario that the number of VMs in different PMs are equal, which is not universal in the practical cloud environments. As a result, such method cannot adapt to various kinds of virtual topologies in the cloud. To the best of our knowledge, aforementioned two problems have not been well resolved together by existing work.

These two problems occur not only after the initial data allocation but also after the data recovery. When some nodes crash, the missing data blocks should be recovered promptly to avoid losing more replicas. Even if the data blocks are nicely distributed after the initial allocation, the aforementioned two problems may still exist after the data recovery. This is because that it makes the same assumptions for the storage nodes with the stage of initial data allocation, so that they follow the same allocation strategy. Hence, there may exist these two problems (data reliability and performance degradation) in the whole lifecycle of data. They greatly affect the effectiveness of HDFS in the cloud. We will further discuss these two problems with a more detailed analysis in the section of motivation.

In this paper, we propose a novel location-aware data block allocation strategy (LDBAS) for HDFS in the cloud to mitigate performance degradation and enhance data reliability at the same time. Firstly, we leverage the relative locations of VMs to form the logical network topology of the virtual cluster, which would not expose the actual underlying network topology of cloud datacenter for safety concerns. By using the logical network topology, we avoid the situation that two same replicas are allocated to the same PM, and improve the data reliability eventually. Then, considering the differences among processing capacities of virtual machines in the cloud, we strive to achieve the state that the numbers of blocks in different nodes are proportional to their processing capacities. This can contribute to more balanced processing loads for local data blocks of all nodes and better data locality. In a word, with the optimized data block allocation, we can enhance the data reliability of HDFS and improve the performance of HDFS-based applications in the cloud at the same time. Finally, we apply LDBAS for two stages of data block allocation (initial data allocation and data recovery) to keep the improvement in the whole lifecycle of data in HDFS.

To summarize, our contributions in this paper are illustrated as follows:

- We propose a novel location-aware data block allocation strategy (LDBAS) to alleviate the performance degradation for HDFS-based applications in the cloud while guaranteeing the expected data reliability. We apply LDBAS to the initial data allocation and data recovery, and design the corresponding algorithms.
- We implement LDBAS into an actual Hadoop cluster in the cloud environments and conduct extensive experiments. The results show LDBAS can guarantee the designed data reliability and reduce the job execution time of I/O-intensive applications in Hadoop by 8.9% on average and up to 11.2% compared with original Hadoop in the cloud.

This paper is an extended version of [21], based on which additionally includes a data recovery algorithm using LDBAS and a more comprehensive experimental evaluation. The rest of this paper is organized as follows. Section 2 provides the background and motivation. The proposed data block allocation strategy will be described in details in Section 3. We show the experimental evaluation in Section 4. Section 5 raises some discussion. Related works are discussed in Section 6. Finally, we conclude this paper in the last section.

2. Background and Motivation

In this section, firstly we describe the default strategy used by HDFS to allocate data blocks. Then we identify two problems caused by the default allocation strategy for HDFS in the cloud environments, which motivate us to design a new data allocation strategy in HDFS.

2.1 Background: The Design of HDFS

Hadoop Distributed File System [8], is one of the fundamental support systems for Hadoop [3], Spark [9] and other MapReduce-like big data processing frameworks [10]. It is designed to store large-scale data sets reliably and to stream those data sets at high bandwidth to upper-layer applications. To achieve these goals, HDFS splits a file into small data blocks and place them on different nodes in the cluster. Moreover, replication strategy is used to maintain data reliability, and the replication factor is set as 3 by default. The redundant replicas can also be useful to improve the read bandwidth for upper-layer applications. Furthermore, HDFS provides an API that exposes the locations of file blocks and allows applications to schedule a task to the node which hosts the data block. Scheduling a task close to the corresponding data is known as data locality. These important characteristics make Hadoop more efficient.

1) Two Stages of Data Allocation in HDFS: HDFS consists of two important stages of data allocation to maintain the data reliability. One stage is the initial data allocation when data blocks are firstly written into HDFS. Multiple replicas of data blocks should be allocated into different nodes as far as possible. The other stage is the data recovery, which will be triggered once some DataNodes break down. The missing replicas of data blocks in the corrupt nodes should be reallocated to the live DataNodes in time to avoid losing more replicas.

2) Default Strategy of Data Allocation in HDFS: HDFS supports tree hierarchical network topology as shown in **Fig. 1**, which assumes that HDFS runs on a cluster consisting of many data centers filled with racks of computers. The bandwidth of nodes within a subtree may be greater than the bandwidth between subtrees in general, which means that keeping all replicas within the same rack can reduce the write cost. However, the whole rack may lose connection with other nodes, which will make data unavailable. Hence, the default data block allocation strategy in HDFS is designed to get a tradeoff between minimizing the write cost and maximizing the data reliability and aggregate read bandwidth.

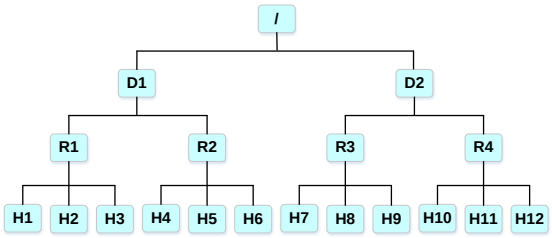

Fig. 1. Tree hierarchical network topology of HDFS, where D means the data centers, R indicates the racks and H represents the nodes

This strategy is used in two stages of data allocation in HDFS. In the stage of the initial data allocation, HDFS places two replicas on different nodes at one rack and the third replica at another rack. If the replication factor is more than three, the rest replicas are allocated randomly on different nodes. Similarly, in the stage of data recovery, HDFS finds out the missing data blocks in the broken nodes, and reallocates them into the live nodes while making all the replicas satisfying the similar data reliability as in the initial allocation.

3) Assumptions in the Default Strategy of Data Allocation in HDFS: The default allocation strategy makes two implicit assumptions:

- The crashes of nodes are independent with each other unless the whole rack loses connection with the others.
- The nodes are homogeneous and serve almost the equal processing capacities, hence HDFS keeps almost the same number of data blocks between DataNodes.

In the physical environments, it ensures sufficient data reliability and provides excellent read bandwidth for upper-layer applications. However, it does not perform well in the cloud environments, mainly due to the co-location of VMs. We explain this in the next subsection.

2.2 Motivation

HDFS works effectively in the physical environments. However, the two assumptions described in the last subsection do not hold in the cloud environments, which brings two problems: data reliability loss and performance degradation for upper-layer applications.

Data reliability loss: The assumption 1 breaks down as DataNodes may co-locate in the same PM inside the virtual cluster. Once a PM crashes, there will be more than one crashed DataNode, which means that two or more replicas of the same file block may be lost at the same time. For example, as shown in **Fig. 2**, all replicas of block 1 and 10 would be lost once *PM1* crashes. This situation does not only happen after the initial data allocation, but also after the data recovery when some data nodes break down. In **Fig. 2**, the block **5** is nicely allocated across three different PMs. Once *PM3* breaks down, the replica of block **5** in *VM6* will be lost. After the data recovery, the missing replica may be reallocated to *VM1* under *PM1*. Since there already exists one replica in *VM3* under *PM1*, it results in bad data reliability for block **5**.

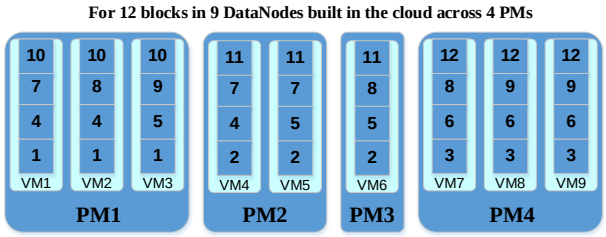

Fig. 2. One example of file blocks distribution for HDFS in the cloud

In order to identify the seriousness of this problem, we constructed a virtual Hadoop cluster with 16 virtual DataNodes in 8 PMs based on OpenStack cloud platform [11]. Then, we wrote data sets into HDFS with sizes of 4GB, 8GB, and 16GB, and counted the data reliability, which was reflected by the proportion of blocks with 3 replicas across different PMs. The result is shown in **Fig. 3**(a). Similarly, we also observed the data reliability after data recovery by simulating one DataNode's crash. The result is shown in **Fig. 3**(b). Both of the results show that merely around 70% of data blocks could achieve the expected data reliability.

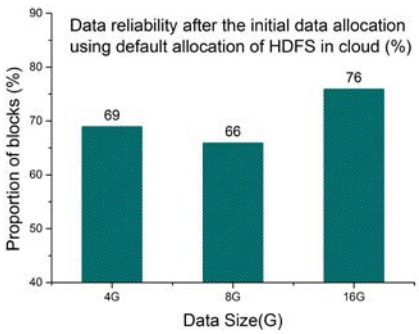

Fig. 3. (a) Data reliability after initial data allocation for HDFS in the cloud

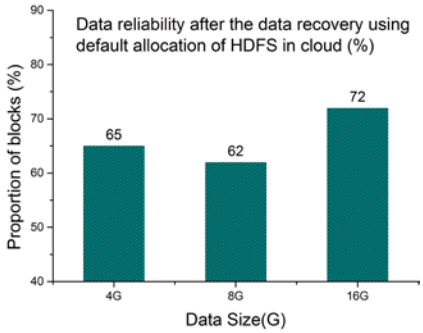

Fig. 3. (b) Data reliability after data recovery for HDFS in the cloud

Performance degradation: The performance degradation comes from many reasons. Except for the additional load of virtualization, the main reason is the heterogeneity of processing capacities between VMs in the cloud, which violates the assumption 2. Generally, the co-located VMs share and compete for physical resources in their host machine, especially the disk I/O bandwidth. The contention is particularly serious when the virtual nodes execute some similar work at the same time, such as in map phase when they read data from the disk in Hadoop [7]. Therefore, the processing capacities of VMs with the same configuration may be actually unequal among different PMs, especially when PMs host different numbers of VMs. For example in **Fig. 2**, the disk I/O bandwidth of *VM6* may be much higher than *VM1*.

The heterogeneity of processing capacities greatly affects Hadoop's performance under the default strategy of data allocation in HDFS. The reason is that the distribution of data blocks is almost even across DataNodes, no matter after the initial data allocation or data recovery by using the default strategy. It will lead to poor data locality, which means the nodes with higher PCs may execute many remote map tasks and read massive data through the network. Since network bandwidth is lower than local disk I/O bandwidth in most cases, excessive remote map tasks will lead to longer task time in map phase and competition for network resource with the shuffle operation. Hence, the performance of upper-layer applications will decrease in the cloud environments by using the default allocation strategy in HDFS.

In general, these two problems come from the co-location and resource sharing of VMs. To solve these problems, a straightforward solution is to avoid the co-locations of VMs. However, it takes more restrictions for VM placement in the cloud. For example, the number of available physical hosts must be larger than the number of virtual HDFS nodes, even when many hosts may be not suitable to deploy HDFS nodes in the cloud, such as the hosts with shared storage only. Hence, avoiding the co-locations of VMs is practically unsuitable. This motivates us to design a preferable data block allocation strategy in HDFS to adapt to the cloud environments.

3. Our Strategy

This paper aims to present a new data block allocation strategy (LDBAS) of HDFS to address the problems of data reliability loss and performance degradation for upper-layer applications in the cloud. In this section, firstly we explain the principles we use to allocate different data block replicas in HDFS. After that, we describe the problem definitions of data block allocation when writing a new file to HDFS and data block recovery when some DataNodes fail. Finally, we describe our new algorithms corresponding to these two problems.

3.1 The Allocation Principles Used in Our Strategy

The primary insight of our new allocation strategy is as follow: the essential reason of these two problems is the co-location of VMs, so we want to make full use of the information of locations where the VMs are co-located in PMs for preferable data block allocation. This location information can be provided by the cloud vendors or obtained through some network topology inference technologies [12][13]. Besides, taking the potential safety risk for cloud vendors into consideration, we only exploit the information of the relative locations of VMs to form the network topology in our strategy, which is an abstract logical topology and would not expose the actual underlying network topology of the cloud data center. For example in **Table 1**, we just get the information about if two VMs are inside the same PMs, or if they are at the same rack. We will further discuss how to obtain the location information of a virtual cluster in more detail in Section 5.

Table 1. Network distance defined in our modified Hadoop

Source_location	Target_location	Distance	Meaning
/R1/PM1/VM1	*/R1/PM1/VM1*	0	the same node
	/R1/PM1/VM2	2	nodes in the same PM
	/R1/PM2/VM3	4	nodes in different PMs at the same rack
	/R2/PM3/VM4	6	nodes at different racks

Table 1 denotes the new network distance used in our new strategy. We add a new layer representing the physical host of the virtual nodes to identify the locations of DataNodes. New location description is like */R1/PM1/VM2*, which means the virtual node *VM2* is located in physical machine *PM1* under rack *R1*. Meanwhile, the network distance can be used to identify the relative location between two DataNodes. Based on the information of network topology, we give the principles which we use to allocate the data blocks.

To address the problem of data reliability loss, our simple principle is to avoid any two replicas of one data block in the same PM though in different VMs. For typical three-replica strategy, we summarize the principles to ensure data reliability in our strategy as follows:

- Principle 1: Do not place two same replicas in the same VM.
- Principle 2: Do not place two same replicas in the same PM.
- Principle 3: Place three replicas under two racks as far as possible.

These three principles guarantee data reliability when a single VM, PM or a whole rack crashes. Data reliability at the level of VMs should have the highest priority, then at the level of PMs, finally at the level of the racks.

To address the problem of performance degradation for upper-layer applications, we allocate an appropriate number of file blocks into DataNodes to balance all nodes' estimated processing time as far as possible. The estimated processing time indicates how much time the node will cost to process all local data blocks of this file. With balanced processing time, more data blocks are expected to be processed at local, which means better data locality. Hence, there may be fewer remote tasks and less network traffic, which is very useful to reduce the total job completion time. Moreover, considering that the requests of file blocks arrive at HDFS one by one, we can only estimate the processing time according to the information about the arrived data blocks. In short, our principle to improve the performance of data processing is described as follow:

- Principle 4: Place data block replicas on the DataNodes with the least current estimated processing time for each arriving data block.

Different methods for estimating processing time can be plugged into our strategy. We estimate every node's processing time as:

$$ProcessingTime = BlockSize * BlockNumber / ProcessingCapacity, \quad (1)$$

where *BlockSize* is the data block size chosen by users, *BlockNumber* is the number of data blocks that have been allocated to this node, and *ProcessingCapacity* (*PC*) denotes the average data size that the node can process per second in the map phase. *PC* can be a general configuration of each node in Hadoop, which users can configure according to their experience to satisfy their special needs. However, it is difficult to model a node's practical *PC* for unknown map tasks when writing a file into HDFS. An approximation is to model the ratio of processing speed between nodes since the nodes in Hadoop always execute the similar work.

In our strategy, we use read bandwidth as the *PC* of a node based on the following reasons.

- There are a lot of I/O-intensive tasks in Hadoop's jobs.
- The time cost of the read/write operations often accounts for most of the completion time in the map phase.
- In addition, the I/O interference is serious because of the virtualization implementation. Some researchers believe that I/O virtualization is the main bottleneck in the cloud [14].

Especially, we use the node's sequential read bandwidth as the *PC* since the sequential read for large files occupies a large proportion in the map phase. In our strategy, we configure it for each node after measuring it by using the open-source benchmark tool *Hdparm*. We control all VMs to run the measurement tool at the same time and get the ratio of *PCs* of all VMs. Then, we can get the static *PCs* and configure them after the virtual cluster has been initialized. On the other hand, the *PCs* may vary with the number of the VMs inside the same physical host over time. Hence, the *PC* should be dynamically reconfigured over time. We further discuss the measurement of the *PC* in more detail in Section 5.

3.2 Initial Data Allocation Problem

3.2.1 Problem Statement

Based on the aforementioned principles, we give a formalized description of initial data allocation problem and our new allocation algorithm.

In the cloud, a virtual Hadoop cluster consists of a set of virtual nodes (vm_1, vm_2,…, vm_m) which are hosted in different PMs under different racks. We define a matrix $\boldsymbol{D}$ of $m*m$ to identify the network distance between nodes, where m is the number of DataNodes. D_{ij} means the network distance between vm_i and vm_j. The detail of network distance is described in **Table 1**. Through the network topology, we can obtain the number of racks and PMs, denoted as n_{rack} and n_{PM}, respectively. Moreover, to identify the processing capacities of different nodes, we use a vector $\boldsymbol{P}$ with the length of m. P_i denotes the PC of the node vm_i.

When one request for writing a file to HDFS arrives at the NameNode, the NameNode will return a list of nodes with a length of r to the client (r represents the replication factor). Assuming that the total number of blocks is n, the initial data block allocation problem is to find r locations for n data blocks one by one, i.e., $r*n$ locations in total. Because there are m different possible locations for each block, we can describe the final allocation result as a matrix $\boldsymbol{A}$ with a size of $m*n$. A_{ij} denotes that if the data block j is allocated into the node vm_i, where $A_{ij} \in \{0,1\}, \forall i \in [0,m), j \in [0,n)$. Our objective is to minimize the standard deviation of the estimated processing time of all nodes in the map phase. If the size of each data block is s, we can get the estimated time T_i of processing local data blocks for each node as follow:

$$T_i = \frac{\sum_{0 \le j < n} A_{ij} * s}{P_i}, \forall i \in [0, m) \tag{2}$$

Then we can get the standard deviation of estimated processing time (STD-EPT) as follow:

$$\sigma_{time} = \sqrt{\frac{\sum_{0 \le i < m} \left(T_i - \sum_{0 \le j < m} T_j / m \right)^2}{m}} \tag{3}$$

We define the initial data block allocation problem as finding a matrix $\boldsymbol{A}$ to minimize the above objective function σ_{time}. Moreover, in order to enhance the data reliability, we propose several constraints (4)-(7) to this problem.

$$\forall i \in [0, m), j \in [0, n) : A_{ij} \in \{0, 1\} \tag{4}$$

$$\forall j \in [0, n) : \sum_{0 \le i < m} A_{ij} = r \tag{5}$$

The constraints (4) and (5) enhance the data reliability across VMs. The constraint (4) describes that there is at most one replica in the same virtual node for each data block. And the constraint (5) indicates that there should be r replicas in total across m virtual nodes for each data block to guarantee the data reliability.

$$\forall j \in [0, n) : D_{ab} \ge 4 \text{ and } D_{ac} \ge 4 \text{ and } D_{bc} \ge 4, \\ \text{if } n_{PM} \ge 3 \text{ and } A_{aj} = A_{bj} = A_{cj} = 1 \tag{6}$$

$$\forall j \in [0, n) : D_{ab} = 6 \text{ or } D_{ac} = 6 \text{ or } D_{bc} = 6, \\ \text{if } n_{rack} \ge 2 \text{ and } A_{aj} = A_{bj} = A_{cj} = 1 \tag{7}$$

Then, for simplicity, we give the constraints of the typical three replicas for data reliability across the PMs and racks as shown in (6) and (7). If the replication factor r is greater than 3, the rest replicas can be allocated randomly on different nodes to improve read bandwidth of upper-layer applications just as the default block allocation strategy of HDFS. Thus, among the r replicas of each block, at least three replicas should meet the constraints (6) and (7). We denote a, b, and c as the locations of these three replicas. To be specific,

- The constraint (6) enhances the data reliability across PMs. If the number of PMs is equal to or greater than 3, three replicas of the same block should be allocated on at least 3 different PMs. That is the distance between any two replicas should be more than 4.
- The constraint (7) enhances the data reliability across racks. If the number of racks is equal to or greater than 2, three replicas of the same block should be allocated on at least 2 different racks. That is at least one distance between two replicas should be 6.

3.2.2 The Initial Data Allocation Algorithm

To find the best locations for the replicas of each arriving block, the ideal solution is to find out all the combinations of any 3 DataNodes. After that, the combinations could be traversed to find the best combination with the minimum contribution to STD-EPT while satisfying the data reliability. The ideal data block allocation algorithm (IDBAS) is described in Algorithm 1. However, this traversal algorithm is highly time-consuming. In each loop, the time complexity of looking for all the combinations of any 3 nodes is $O(\binom{m}{3})$, which means $O(m^3)$ (m is the number of DataNodes). The time complexity of calculating the STD-EPT is $O(m)$. As a result, traversing all the combinations to find out the nodes with the minimum contribution to STD-EPT will cost nearly $O(m^4)$. When there are 128 data nodes, the response time will exceed 1 second for a single block request (the detailed result of the simulation experiment is

described in Section 4.1). Moreover, the NameNode usually processes more than dozens of requests at the same time. Thus, IDBAS is not suitable for HDFS due to the high time cost.

Algorithm 1: the ideal data block allocation algorithm (IDBAS)

Input:
D: the matrix of network distance between nodes
P: the array of processing capacity of nodes
s, r, n: the block size, block replication factor and block number of file writing to HDFS
Output:
A: the allocation matrix of file blocks
1. Initialize $A_{ij} = 0$, $time_i = 0$
2. nodecombination ← the combinations of any 3 datanodes
3. **for** block = 1 to n **do**
4. update $time_i$ //using the equation (2)
5. *set*←all the combinations in nodecombination which meet the constraints (6) (7)
6. **if** *set* is NULL //there are no combinations meeting all the constraints
7. *set* ←all the combinations in nodecombination which meet the constraints (6)
8. **if** *set* is NULL //there are no combinations meeting only the constraint (6)
9. *set* ←nodecombination //use all the combinations as the candidate
10. chosenNodes ← the nodes in the *set* which contribute the minimum time to the standard deviation of estimated processing time
11. update *A* for the replicas on the nodes in chosenNodes
12. **end for**
13. **return** *A*

On this basis, we propose a new greedy algorithm (shown in Algorithm 2) to allocate each arriving data block, eventually to minimize the objective function while satisfying the constraints.

As seen in Algorithm 2, we firstly update the estimated processing time of all DataNodes for the arrived blocks of this file in line 3. Then, we use three steps to allocate the three replicas for each block in lines 5-20. We allocate the first replica on nodeA whose estimated processing time is the minimum in the cluster. Then we seek nodeB for the second replica according to the network distance between nodeA and nodeB (*distanceAB*). The priority of *distanceAB* is as follows: 6 > 4 > 2. We choose nodeB whose estimated processing time is the minimum from the candidate nodes in the cluster. To seek nodeC for the third replica, we consider the combination of the distances: {*distanceAC*, *distanceBC*}. The priority of the distance combination is as follows: {4, 6}>{6, 6}>{4, 4}>{2, 2}, which can enhance the data reliability as far as possible. Finally, we choose nodeC whose estimated processing time is the minimum from the candidate nodes. For those replicas that exceed the replication factor 3, we straightly choose the nodes whose estimated processing time is the minimum in line 21.

Algorithm 2: the initial data block allocation algorithm (LDBAS-initial)

Input:
m: the number of datanodes
D: the matrix of network distance between nodes
P: the array of processing capacity of nodes
s, r, n: the block size, block replication factor and block number of file writing to HDFS
Output:
A: the allocation matrix of file blocks
1. Initialize $A_{ij} = 0$, $time_i = 0$

```
2.  for block = 1 to n do
3.    update time_i //using the equation (2)
4.    Initialize setA←{1,2,…,m}, setB←NULL, setC←NULL
5.    nodeA←the node in setA where time_node is the minimum
6.    for distanceAB is (6, 4, 2) in turn and setB is NULL do
7.      for i = 1 to m do
8.        if D_nodeA,j == distanceAB then setB.append(i)
9.      end for
10.   end for
11.   nodeB←the node in setB where time_node is the minimum
12.   if nodeB is NULL then update A and return A
13.   for (distanceAC,distanceBC) is (4, 6) (6, 6) (4, 4) (2, 2) in turn and setC is NULL do
14.     for i←1 to m do
15.       if (D_nodeA,i, D_nodeB,i) is (distanceAC, distanceBC) then setC.append(i)
16.     end for
17.   end for
18.   nodeC←the node in setC where time_node is the minimum
19.   if nodeC is NULL then update A and return A
20.   update A_nodeA,block A_nodeB,block A_nodeC,block
21.   allocate the rest replicas on random nodes with the minimum time
22. end for
23. return A
```

From the algorithm, we can see that the time complexity in each loop of LDBAS is O(m). The time complexity of choosing each replica is O(m) in line 5, in lines 6-12, and in lines 13-19. It will save a great deal of time cost compared with the ideal traversal algorithm. We will evaluate the time cost in the simulation experiments in Section 4.1.

3.3 Data Block Recovery Problem

Once some DataNodes crash, the NameNode will scan all the files to find out all the missing data block replicas which need to be restored. These corrupted data blocks will be added to a queue called *blockToReplication*. Our objective is to find the nodes where the replicas should be reallocated for each corrupted data block, to satisfy the data reliability and improve the performance of upper-layer data processing applications. Firstly, we give the formalized description of this problem. Then, based on the principles mentioned in Section 3.1, we suggest our new data recovery algorithm for this recovery problem.

3.3.1 Problem Statement

Similar to Section 3.2.1, we define a matrix $\boldsymbol{D}$ to identify the network distance between m live DataNodes, and define a vector $\boldsymbol{P}$ to identify the processing capacities of DataNodes. When some DataNodes crash, HDFS will find out all the corrupted data blocks by scanning the whole file blocks periodically, and then add them to the queue *blockToReplication*. For each data block d in the queue *blockToReplication*, we use r_d to indicate the number of total replicas which the block d needs, and n_d to represent the total number of data blocks of the same file. $\boldsymbol{A^d}$ denotes the current distribution of file data blocks for block d, where the size of $\boldsymbol{A^d}$ is $m*n_d$. Then we need two steps to restore these missing data block replicas.

Firstly, we need to seek the source node which the additional replicas can copy from for each block d. To avoid exorbitant load for DataNodes in the process of recovery, there should be no excessive recovery work on the same node. We use W_t to refer the current recovery work

(the number of data blocks to be recovered) in node *t* and *MaxLimit* to indicate the upper limit of recovery work for each node.

Secondly, we need to look for a set of target nodes N_d for the data block *d*. After restoring the replicas on these nodes, it will lead to a new data distribution called A_{new}^{d}. A_{new}^{d} needs to satisfy the data reliability and balanced block distribution to improve the performance of data processing applications. We can formally define the recovery problem as finding the node sets N_d for the block *d* to minimize the STD-EPT. The objective function is similar to the equation (3), but T_i and A_{ij} in (3) should be T_i^d and A_{ij}^d respectively, which represents the current information of data blocks that belong to the same file with block *d*.

In order to enhance the data reliability of data block *d*, we define the same constraints as the constraints (4)-(7). Different from the initial data allocation problem, the matrix *A* in (4)-(7) should be A^d instead. Moreover, to guarantee that there is no excessive recovery work on one same node, we introduce one additional constraint (8) to this problem. It makes sure that there are no more than *MaxLimit* workloads of restoring data blocks in each DataNode.

$$\forall t \in [0, m): \ W_t \leq MaxLimit, W_t \in Z \tag{8}$$

3.3.2 The Data Recovery Algorithm

Based on the principles in Section 3.1.1, we introduce our new data recovery algorithm as described in Algorithm 3. It consists of two steps to restore the missing replicas. Firstly, we choose the source node with the minimum recovery work to avoid overwhelming network load in one same node, which can restore the data blocks as soon as possible in lines 7-10.

Then, given the source node, we look for the target nodes to create additional replicas in lines 12-28. For each data block in the queue *blockToReplication*, we calculate the estimated processing time according to the distribution of live data blocks in line 3. If some blocks of the file have got target recovery nodes but have not been written to the metadata, we update the distribution matrix in advance based on the result of target nodes in the last loop in line 2. This can guarantee the accuracy of the estimated processing time in each loop.

When choosing target nodes, we consider the typical three replicas for a data block in lines 12-26 and the rest replicas will be allocated on random nodes whose processing time are the minimum according to line 27. For typical three replicas, there will be one or two replicas that need to be created additionally. When we choose target node *dn1* for the second replica, we consider the set of candidate nodes that satisfy the data reliability (the network distance between *dn0* and *dn1* is 6, 4 and 2 in turn). Then we choose the node with the minimum time in the candidate nodes as *dn1*. For the third replicas, we consider the network distance between the nodes of existing replicas. If they are across rack (the network distance is 6), we consider the nodes at the same rack with source node or *dn1*. Similarly, we choose the node with the minimum time in the set of candidate nodes as *dn2*. If the network distance between the source node and *dn1* is lower than 6, we will choose a node which has the minimum time and has the network distance of 6 with the source node or *dn1*.

Algorithm 3: The data recovery algorithm

Input:

D: the matrix of network distance between nodes
P: the array of processing capacities of nodes.
block: the first block in the queue of blocks which need to be restored
A^{block}: the distribution matrix of live data blocks for the *block*.

Output:
srcNode: the source node which the additional replicas can copy from
targetNodes: the set of target nodes where the replicas should be created
1. *srcNode* = chooseSrcNode(*block*)
2. update the block distribution matrix A^{block} for the file which has chosen the target nodes for a part of data blocks.
3. update *ProcesingTime* using the information of A^{block} and P //using the equation (2)
4. *targetNodes* = chooseTargetNode(*srcNode*, *block*)
5. return (*srcNode*, *targetNodes*)
6.
7. **function** chooseSrcNode(*block*)
8. *liveNodes* = *block*.getLiveReplicaNode()
9. *srcNode* ← the node with the minimum work and (node.work < *MaxLimit*) in *liveNodes*
10. return *srcNode*
11.
12. **function** chooseTargetNodes(*srcNode*, *block*)
13. **if** *block.numLiveReplicas* == 1 and *numNeedReplicas* >0 **then**
14. *dn1*← the node whose processing time is minimum and where $D_{srcNode,dn1} == 6$
15. targetNodes.add(*dn1*)
16. **if** --*numNeedReplicas* == 0 **then**
17. return *targetNodes*
18. **if** *block.numLiveReplicas* <=2 **then**
19. **if** $D_{srcNode,dn1} == 6$ **then**
20. *dn2*←the node whose processing time is minimum and where $D_{node,dn1} == 4$ or $D_{node,srcNode} == 4$
21. *targetNodes*.add(*dn2*)
22. choose the nodes whose processing time is minimum for other replicas
23. return *targetNodes*
24. **else:**
25. *dn2*←the node whose processing time is minimum and where $D_{node,dn1} == 6$
26. *targetNodes*.add(*dn2*)
27. choose the nodes whose processing time is minimum for additional replicas
28. return *targetNodes*

4. Experimental Evaluation

In this section, we comprehensively evaluate our new strategy through the simulation experiments and realistic experiments in a Hadoop cluster. In particular, we mainly evaluate it in three aspects: data reliability, the performance of data processing applications and the time cost of the algorithm for allocating each data block. We will describe these metrics in detail in the subsequent subsections.

4.1 Simulation Experiment

Firstly, we carry out a series of simulation experiments to compare our initial allocation algorithm (LDBAS-initial) with the ideal traversal algorithm which contains the theoretical optimal solution (IDBAS). We use two metrics to evaluate the effectiveness of LDBAS-initial.

The standard deviation of estimated processing time (STD-EPT): *STD-EPT* is the same metric as the objective function shown in (3), which can reflect the quality of data block

distribution. Through this theoretic evaluation, we can find the gap between the solution of our strategy and the theoretic optimal solution. The results are shown in 4.1.2.

The time cost of the initial data allocation algorithm: this metric represents the execution efficiency of two algorithms. Through the theory evaluation of the time cost of the algorithms, we attempt to prove the high time cost of IDBAS, which motivates us to design LDBAS-initial with the lower time cost. The results are shown in 4.1.3.

4.1.1 Simulation Setup

We compare these two algorithms by simulating allocating data blocks to DataNodes and run each group independently for 20 times. There are two variables in the simulation.

The number of data blocks: This variable can reflect the change of *STD-EPT* along with the increase in the number of data blocks. We set 32 DataNodes on 20 physical machines at 4 racks, and adjust the number of data blocks from 64 to 1024. After allocation, we analyze the allocation result and calculate the *STD-EPT* using the equation (3). The value of processing capacity of each DataNode is set as the reciprocal of the number of virtual nodes in the same PM, which varies from 0 to 1 data block per second.

The number of nodes: This variable can reflect *the time cost* for each data block along with the increase in the number of nodes. We set the number of nodes from 8 to 128 and simulate allocating 256 data blocks into these nodes. Then we record the average execution time of allocating each data block.

4.1.2 Effectiveness of Algorithms in Simulation

Fig. 4 shows the standard deviation of estimated processing time (STD-EPT) for the local data blocks of all nodes. The lower STD-EPT means the better distribution of data blocks. As illustrated, the STD-EPT of LDBAS is always close to the optimal solution (IDBAS) along with the increase in the number of data blocks. Since the value of processing capacity is set as from 0 to 1 data block per second, the mean processing time is from 10 to 150 seconds along with the increase of data blocks. It shows that the standard deviation of the processing time of LDBAS and IDBAS both are extremely small (both of them are less than 1 second). Hence, the results show that we can achieve the near-optimal distribution of data blocks.

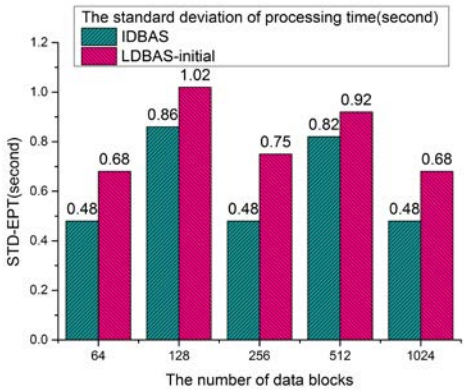

Fig. 4. STD-EPT in simulation (lower is better)

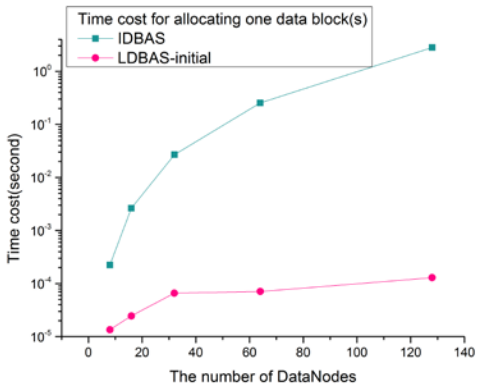

Fig. 5. Time cost of allocating each data block (lower is better)

4.1.3 Time Cost of Algorithms in Simulation

As illustrated in **Fig. 5**, LDBAS can get the allocation result for each data block in less than **10^{-3} second** along with the increase in the number of DataNodes. However, the time cost of IDBAS increases in an explosive way. And when the number of DataNodes is 128, the execution time of IDBAS is more than **1 second** for one each data block allocation. As the NameNode usually handles more than dozens of requests for data block allocation at the same time and has some other managing tasks for the whole cluster, the cost of IDBAS is obviously unacceptable. In contrast, our strategy can allocate each data block within **1 millisecond** which satisfies the real-time requirement and get the near-optimal solution at the same time.

4.2 Experiments in an Actual Hadoop Cluster

In this section, we implement our strategy into an actual Hadoop cluster and conduct a series of experiments to evaluate our strategy (LDBAS) compared with the default data block allocation strategy (DDBAS) used in HDFS. We will describe the experiments from four aspects as follows: the data reliability, the time cost for allocating each data block, the standard deviation of estimated processing time and the effect of improving performance for some data processing applications. We firstly describe the experimental environment, then give the experimental results and analysis.

4.2.1 Experimental Environment

- **Hadoop cluster in the cloud.** We implement LDBAS into a virtual Hadoop cluster on a private cloud at local based on OpenStack platform. Hadoop version is 2.6.0. We apply for 17 KVM-based VMs to run virtual Hadoop cluster, which consists of one NameNode and 16 DataNodes. The ResourceManager is running with NameNode and the NodeManagers are running with DataNodes. All VMs are configured with 4 virtual cores, 8 GB memory, and 80GB disk. These VMs are hosted on nine PMs, which are under two racks connected by Gigabit Ethernet. These 17 VMs' locations are shown in **Table 2**. We configure the processing capacity for each node by measuring disk I/O bandwidth with a benchmark tool known as *Hdparm*. We control all VMs to run the measurement tool at the same time, then we get the ratio of sequential read bandwidth as the PCs for all VMs.
- **Benchmark suite.** We choose a popular open-source benchmark suite known as BigDataBench [15]. BigDataBench models many typical workloads for big data applications, e.g., MicroBench and search engine. In our work, we use three different workloads to evaluate our strategy: RandomWriter, Sort, and WordCount.

Table 2. VMs' locations in the experimental environment

Rack	Physical Host	Virtual Nodes
Rack1	PM1	NameNode
	PM2	DataNode1 DataNode2 DataNode3
	PM3	DataNode4
	PM4	DataNode5
	PM5	DataNode6 DataNode7 DataNode8
Rack2	PM6	DataNode9 DataNode10
	PM7	DataNode11
	PM8	DataNode12 DataNode13 DataNode14
	PM9	DataNode15 DataNode16

4.2.2 Data Reliability

Firstly, we use RandomWriter to generate data sets and write them to HDFS under DDBAS and LDBAS. We generate different data sets with size: 4GB, 8GB, and 16GB. The block size is set to 64MB, and the replication factor is 3. We count the proportion of file blocks with 3 replicas across different PMs. **Fig. 6** shows that under the original strategy **29.7%** of data blocks on average cloud not achieve the expected data reliability. These blocks only have two or even one different replicas across different physical nodes. With our new strategy, **100%** of file blocks could be allocated with 3 different replicas across PMs in the cloud, which means the same data reliability in the physical environments.

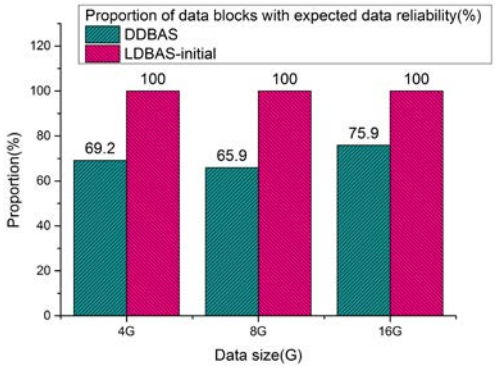

Fig. 6. Data reliability after initial data block allocation (higher is better)

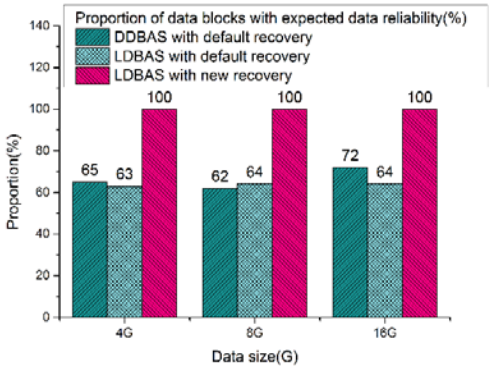

Fig. 7. Data reliability after data recovery (higher is better)

Then, we simulate the failure of DataNodes by disabling the network card. Then we check the data reliability after using different recovery algorithms. The evaluation result is shown in **Fig. 7**. As illustrated, the data reliability is very low (**70%**) after the default recovery algorithm, no matter under DDBAS or our LDBAS. Through our new recovery algorithm under LDBAS, data reliability can achieve the same result as in the physical environment. This proves the effectiveness of our new data block recovery algorithm for data reliability.

4.2.3 Time Cost of Algorithms

We evaluate the time cost of our algorithm in the runtime environment of actual Hadoop cluster. We record the allocation time for each data block in the source code of HDFS and writing them into the logs. The result is shown in **Fig. 8**. We can find that LDBAS only costs about twice of the time of DDBAS, which is about **1.5 to 2 milliseconds**. In addition, the time cost increases slowly along with the increase in data size. It shows that LDBAS can respond a request for writing a data block within the acceptable time in the actual runtime environments.

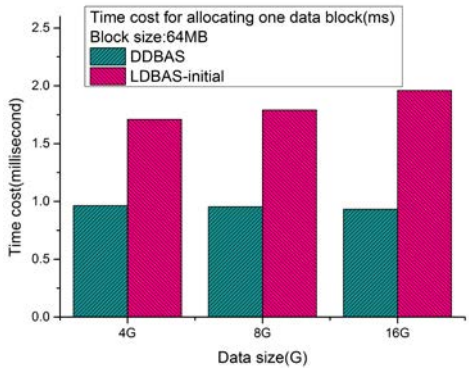

Fig. 8. Time cost for allocating each data block in runtime of HDFS (lower is better)

4.2.4 Standard Deviation of Estimated Processing Time

Similar to the simulation experiment, we analyze the standard deviation of estimated processing time (STD-EPT) for local data blocks in the actual runtime of Hadoop cluster, and then we show the effect of performance for data processing applications in next subsection. **Fig. 9** shows the results after the initial data block allocation under the default data block allocation strategy (DDBAS) and our new strategy (LDBAS). We can find that the STD-EPT under DDBAS is larger than STD-EPT under our new strategy, which means there may be more remote map tasks in the data processing applications and longer job completion time.

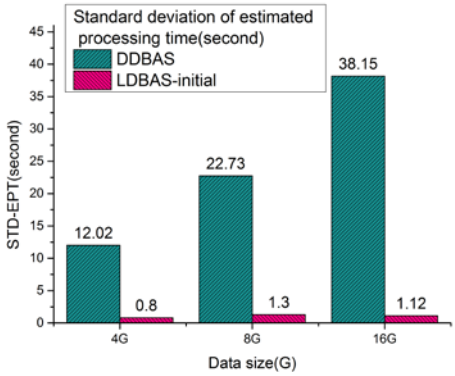

Fig. 9. STD-EPT after initial data block allocation in runtime of HDFS (lower is better)

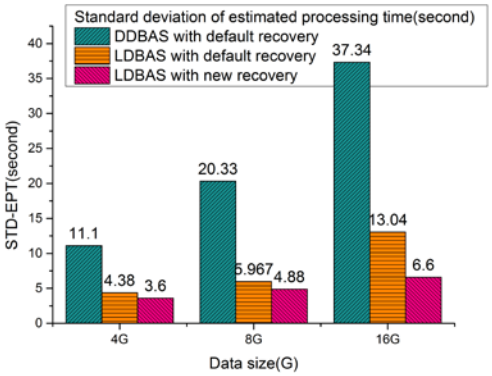

Fig. 10. STD-EPT after data recovery in runtime of HDFS (lower is better)

To evaluate our new data recovery algorithm for the performance of data processing applications, we record the STD-EPT under different recovery algorithms as shown in **Fig. 10**. The left two bars for each data size show that the default recovery algorithm will lead to the **worse data distribution**. LDBAS with both the default recovery and new recovery can achieve better performance, because of the good distribution of the initial allocation with LDBAS. However, new data recovery cannot achieve the expected STD-EPT as small as the STD-EPT after the initial allocation (which is about 1 second in **Fig. 9**. The reason is that the choices of the new replicas are limited to the locations of the live replicas of data blocks for data reliability, which makes it hard to distribute the data block to achieve the minimum STD-EPT in theory.

4.2.5 Performance Effect for Data Processing Applications

In order to verify the performance effect for real-world data processing applications, we test two typical benchmark experiments includes *Sort* and *WordCount*. Then we evaluate the performance through the metrics of the number of local map tasks and the job execution time.

Sort: Sort is a typical I/O-intensive application. After writing data sets to HDFS under different strategies, we run the Sort application to evaluate the performance change with our strategy. There are three data sets: 4GB, 8GB, and 16GB, which indicates 65, 129 and 257 map tasks for corresponding data sets. We run 10 times for each group experiment. Then we count the mean of the number of local map tasks and total job execution time.

The results show LDBAS can increase the number of local map tasks and reduce the total job execution time as shown in **Fig. 11 (a)** and **(b)**. We improve the proportion of local map tasks from 92.3% to 94.4% on average. Moreover, when the proportion is lower, we could improve the proportion by up to about 5%, for instance when data size is 4GB. More local map tasks mean fewer remote executions and less network traffic. Eventually, our strategy could reduce the total job execution time by 8.9% on average and up to 11.2% for the I/O-intensive applications such as *Sort*.

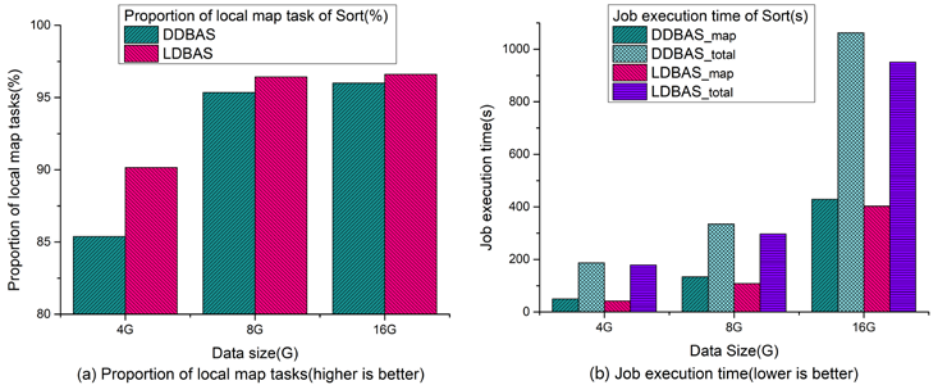

Fig. 11. Performance effect for Sort

WordCount: WordCount is a typical CPU-intensive application. In the same way, after writing data to HDFS under different strategies, we run the WordCount application for 10 times, and then measure the number of local map tasks and the total job execution time. The results show that our new strategy could only reduce the job execution time by 2.8% on average for the CPU-intensive applications as shown in **Fig. 12(b)**. However, we still can **increase** the number of local map tasks as shown in **Fig. 12(a)**, which can contribute to reducing the network overhead in the whole cluster.

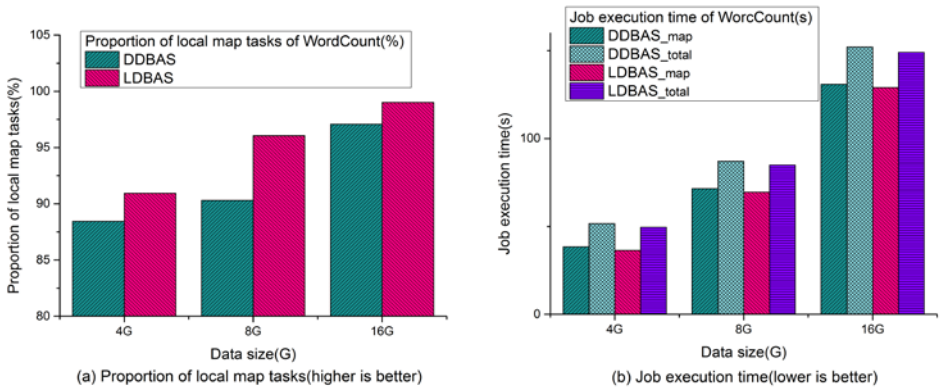

Fig. 12. Performance effect for WordCount

5. Discussion

Application workloads: LDBAS concentrates on improving the performance of I/O-intensive applications, so we consider disk I/O bandwidth as the processing capacities of nodes. In Hadoop, the compute capabilities can be represented by the number of map slots and reduce slots, or the value of cores and memory of NodeManager under Yarn [22]. However, the difference of disk I/O capacity in the heterogeneous cloud environments is greater than in the homogeneous physical environments, which has not been reflected clearly yet and seriously affects the performance of Hadoop in the cloud. The experimental results demonstrate the significance of this consideration. In addition, processing capacity can be a general configuration to represent some other metrics such as compute capacity and network capacity for other particular purposes.

Location information: The location information is an important precondition of LDBAS. We provide two methods to obtain the location information of a virtual cluster for two different roles in the cloud respectively (the cloud vendors and the cloud users). (1) If the cloud vendors would like to adopt our new strategy to improve their cloud services of Hadoop (i.e. Amazon Elastic MapReduce) for cloud users, they can obtain the location information with ease. Since they hold the whole network topology information, they can provide the relative location information of a virtual cluster, when they initialize the virtual Hadoop cluster for the cloud users. (2) If the cloud vendors would not like to adopt our new strategy and not expose the relative location information, the cloud users can still use our strategy to improve the virtual Hadoop cluster, when they rent the virtual machines and build the virtual Hadoop cluster by themselves. In this case, the cloud users have to infer the network topology through some schemes of network topology inference [12][13] without the help of cloud vendors. They can probe the latencies (e.g. round-trip time, RTT) between two VMs by simply using the commands ping. After getting all the RTTs between any two VMs, they can cluster the RTTs. Since the RTTs between two VM pairs with the same distance are always close, and the RTTs are significantly different between two VM pairs with different network distances [12], they can identify the relative locations between two VMs such as shown in **Table 1**.

Measurement of the Processing Capacity: In the experimental evaluation, we measure the processing capacity (PC) and configure it as a static value after the virtual cluster has been already initialized. Since the nodes inside Hadoop cluster always work together, we control all VMs to run the benchmark suite *Hdparm* at the same time. Then, we can get the ratio of

sequential read bandwidth as PC for all VMs. However, the PC of a VM may vary along with the number change of VMs inside the same PM over time. Hence, we suggest to dynamically adjust the processing capacity to reflect the real variability in the cloud environments, if someone applies LDBAS into the practical applications. For example, we suggest to periodically measure the PC by using the benchmark tools when the virtual cluster is idle. Alternatively, we suggest to record the actual PCs with the running processing of all VMs, then periodically reconfigure the value of PCs by analyzing the history records.

VM migration. In our strategy, we assume each DataNode and Tasktracker of Hadoop are in the same node, which can benefit from the data locality. Hence, we assume the nodes in virtual Hadoop will not be migrated by the cloud providers. If two VMs in two different PMs are migrated into the same PM, there will be data reliability loss. Fortunately, VM migration for this kind of VMs is not common in the cloud. This is because the data storage for Hadoop in the cloud is usually at local in the physical machine. It does not need multi-copies in the shared storage, as there is already the guarantee of data reliability in the level of HDFS. As a result, VM migration is extremely costly for this kind of VMs, due to the high cost of migrating the huge amount of local storage volume.

6. Related Work

Our work is related to the methods of optimizing Hadoop in the cloud. We categorize the existing approaches as the following.

Exploring a preferable data block allocation strategy in HDFS [6][17][18]. Geng et al. [6] observed the problems caused by the co-location of VMs and designed an improved file block allocation scheme. However, they only consider the situation that each PM hosts the same number of VMs, which is not universal in practical cloud systems. Our work addresses the similar problems but considers a more common scenario, which can adapt to various kinds of virtual topology. Hence, we realize the heterogeneity of processing capacity between VMs in the cloud environments as we explained in Section 2. Lei [17] proposed CRAVE, which focuses on exploring how to distribute data blocks according to the network bandwidth between virtual nodes, in order to improve the performance of the shuffle operation in Hadoop. Nevertheless, their block allocation strategy can't solve the problem of data reliability. VMware's project Serengeti [18] addressed the problem of data reliability, which introduced NodeGroup to the guarantee replicas are in different fault domains. However, it can not solve the problem of performance degradation caused by the heterogeneity of VMs in the cloud.

Improving the task scheduling of Hadoop. [7][24] proposed some task schedulers to enhance the success ratio of the backup tasks and eventually improve the performance of Hadoop in the heterogeneous cloud environments. We focus on improving the data locality of the map tasks, hence LDBAS may be able to cooperate with this category of approaches.

Improving the infrastructure service in the cloud to adapt to the characteristics of Hadoop applications. DRR [19], a dynamic VM reconfiguration technique for data-intensive computing in the cloud, was proposed to adjust the computing capability of individual VMs to improve the data locality for Hadoop running in the cloud. A dynamic block device reconfiguration algorithm was proposed by Kwonyong et al. [20] to reduce the data transfer time between VMs and thereby improving the performance of virtual MapReduce clusters in the cloud. Different from these studies, our work attempts to achieve the better data block distribution to improve the data locality and the performance of Hadoop in the cloud. Besides, we could solve the problem of data reliability loss which is not addressed by these approaches.

7. Conclusion

In this paper, we address the problems of data reliability loss and performance degradation for Hadoop in the cloud, which are mainly caused by the co-location of VMs. In general, the co-location of VMs can not be avoided in most cases. Hence, we propose a novel location-aware data block allocation strategy (LDBAS) to improve the performance while guaranteeing the data reliability for HDFS-based applications in the cloud. LDBAS allocates data blocks according to the locations and processing capacities of VMs to increase the number of local map tasks. We apply LDBAS for the initial data block allocation and the data recovery in HDFS. The experimental results demonstrate that LDBAS can improve the proportion of file blocks with the expected data reliability by from 70% to 100%. Moreover, for the I/O-intensive applications in Hadoop, we can reduce the total execution time by 8.9% on average and up to 11.2%. In addition, LDBAS can respond a request for writing one data block within an acceptable time in the actual Hadoop runtime. Though our strategy is based on Hadoop, we believe that LDBAS can also benefit Spark and other HDFS-based applications, since HDFS is a fundamental support system of MapReduce-like frameworks.

References

[1] Min Chen, Shiwen Mao, and Yunhao Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171-209, April, 2014. Article (CrossRef Link)
[2] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008. Article (CrossRef Link)
[3] Hadoop. Article (CrossRef Link).
[4] Peter Mell, and Tim Grance, "The NIST definition of cloud computing," *Technical Report*, 2011. Article (CrossRef Link)
[5] Amazon EMR. Article (CrossRef Link).
[6] Yifeng Geng, et al., "Location-aware mapreduce in virtual cloud," In *Proc. of International Conf. on Parallel Processing*, pp. 275-284, September 13-16, 2011. Article (CrossRef Link)
[7] Matei Zaharia, et al., "Improving MapReduce performance in heterogeneous environments," in *Proc. of 8th USENIX conf. on Operating systems design and implementation*, vol. 8, no. 4, pp. 29-42, December 8-10, 2008.
[8] Konstantin Shvachko, et al., "The hadoop distributed file system," in *Proc. of 26th symposium on Mass storage systems and technologies*, pp. 1-10, May 3-7, 2010. Article (CrossRef Link)
[9] Matei Zaharia, et al., "Spark: Cluster computing with working sets," in *Proc. of 2nd USENIX conf. on Hot topics in cloud computing*, vol. 10, pp. 10-10, June 22-25, 2010.
[10] Sherif Sakr, Anna Liu, and Ayman G. Fayoumi, "The family of mapreduce and large-scale data processing systems," *ACM Computing Surveys*, vol. 46, no.1, pp. 11, October, 2013. Article (CrossRef Link)
[11] OpenStack Sahara. Article (CrossRef Link).
[12] Dominic Battré, et al., "Evaluation of network topology inference in opaque compute clouds through end-to-end measurements," in *Proc. of International Conf. on Cloud Computing*, pp. 17-24, July 4-9, 2011. Article (CrossRef Link)
[13] Mark Coates, et al., "Maximum likelihood network topology identification from edge-based unicast measurements," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 11-20, June, 2002. Article (CrossRef Link)
[14] Jeffrey Shafer, "I/O virtualization bottlenecks in cloud computing today," in *Proc. of 2nd Conf. on I/O virtualization*, pp. 5-5, 2010.
[15] Lei Wang, et al., "Bigdatabench: A big data benchmark suite from internet services," in *Proc. of 20th International Symposium on High Performance Computer Architecture*, pp. 488-499, February 15-19, 2014. Article (CrossRef Link)

[16] Kento Aida, et al., "Evaluation on the performance fluctuation of hadoop jobs in the cloud," in *Proc. of 16th International Conf. on Computational Science and Engineering*, pp. 159-166, December 3-5, 2013. Article (CrossRef Link)
[17] Lei Lei, "Towards a high performance virtual hadoop cluster," *Journal of Convergence Information Technology*, vol. 7, no. 6, 2012.
[18] VMware Serengeti. Article (CrossRef Link).
[19] Jongse Park, et al. "Locality-aware dynamic VM reconfiguration on MapReduce clouds," in *Proc. of 21st international symposium on High-Performance Parallel and Distributed Computing*, pp. 27-36, June 18-22, 2012. Article (CrossRef Link)
[20] Kwonyong Lee, et al., "A dynamic block device reconfiguration algorithm in virtual MapReduce cluster," *Cluster computing*, vol. 17, no. 4, pp. 1171-1183, 2014. Article (CrossRef Link)
[21] Hua Xu, et al, "Location-Aware Data Block Allocation Strategy for HDFS-Based Applications in the Cloud," in *Proc. of 9th International Conf. on Cloud Computing*, pp. 252-259, June 27-July 2, 2016. Article (CrossRef Link)
[22] Vinod Kumar Vavilapalli, et al., "Apache hadoop yarn: Yet another resource negotiator," in *Proc. of 4th annual Symposium on Cloud Computing*, pp. 5, October 1-3, 2013. Article (CrossRef Link)
[23] Changqing Ji, et al, "Big data processing in cloud computing environments," in *Proc. of 12th International Symposium on Pervasive Systems, Algorithms and Networks*, pp. 17-23, December 13-15, 2012. Article (CrossRef Link)
[24] Shin-Jer Yang and Yi-Ru Chen, "Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds," *Journal of Network and Computer Applications*, vol. 57, pp. 61-70, November, 2015. Article (CrossRef Link)

Hua Xu received the B.E. degree in Computer Science and Technology from the University of Science and Technology of China (USTC) in 2012. He is currently a Ph.D. student in the Department of Computer Science and Technology in USTC. His research interests include distributed systems, cloud computing, and big data processing.

Weiqing Liu received the B.E. degree in Computer Science and Technology from the University of Science and Technology of China (USTC) in 2011. He is currently a Ph.D. student in the Department of Computer Science and Technology in USTC. His research interests include cloud computing, mobile computing, and big data processing.

Guansheng Shu received the B.E. degree from Jilin University (JLU) in 2011. He is currently a Ph.D. student in the Department of Computer Science and Technology in USTC. His research interests include cloud computing, mobile computing, and big data processing.

Jing Li received his B.E. degree in Computer Science from the University of Science and Technology of China (USTC) in 1987, and Ph.D. in Computer Science from USTC in 1993. Now he is a professor in the Department of Computer Science and Technology in USTC. His research interests include distributed systems, cloud computing, big data processing, and mobile cloud computing.