J Internet Serv Appl (2012) 3:319–327
DOI 10.1007/s13174-012-0069-8

SI: DATA INTENSIVE COMPUTING

Record-Based Block Distribution (RBBD) and weighted set cover scheduling (WSCS) in MapReduce

Qiangju Xiao · Pengju Shang · Jun Wang

Received: 15 May 2012 / Accepted: 16 September 2012 / Published online: 23 October 2012

Abstract The massive increase in data volume with the development of computation capability has outmoded compute-intensive clusters for the analysis of large-scale datasets due to the network bottleneck caused by the large amount of data transferred over the network. Chunk-based storage systems are typical data-intensive clusters introduced to do big data analysis. They split data into blocks of the same predefined size and randomly store them across nodes. These systems adopt the strategy of co-located computing and storage to reduce the network transfer by scheduling computation to the node with the most required data. It performs well when the record as the input of the analysis is mostly on the same node. However, this does not always hold to be true, because there is a gap between the records and the blocks. Blocks are scattered across the data nodes with no regard to the semantics of the record. The current solution overlooks the relationship between the computation unit as a record and the storage unit as a block. For records contained in one block, there is no data transfer to schedule by block locations. On the other hand, in practice, one record could be consisted of several blocks which widely applies to binary files, as a result, extra data transfer is incurred to prepare the input data, because these blocks are randomly stored across the nodes and need to be transferred to the selected compute node. Our contribution is to develop a Record-Based Block Distribution (RBBD) framework for data-intensive analytics to eliminate the gap between records and blocks, reducing the data transfer volume before the analytics are processed. Meanwhile, a weighted set cover scheduling (WSCS) is implemented to further improve the performance of the data-intensive analytics by choosing the best combination of data nodes to perform the computation. Our experiments show that using our RBBD framework and WSCS, the data transfer volume is reduced by an average of 36.37 % and our weighted set covering algorithm outperforms the random algorithm by 51–62 %, with the deviation from the ideal solutions of not more than 6.8 %.

Keywords Hadoop · Data-intensive · MapReduce · HDFS · Co-located compute and storage · HEC

Q. Xiao · P. Shang · J. Wang (✉)
Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, 32816, USA
e-mail: jwang@eecs.ucf.edu

Q. Xiao
e-mail: qxiao@eecs.ucf.edu

P. Shang
e-mail: shang@eecs.ucf.edu

1 Introduction

The further development of computation capability and large-scale storage systems facilitates research that interweaves between different scientific fields. This eventually produces large-scale datasets. Scientific applications running on high end computing (HEC) platforms can generate large volumes of output [1]. Virtualization, simulation and other scientific applications running on HEC are the major contributors. The output data size has been growing with rapid speed, reaching peta-scale sizes. Since I/O bandwidth is the bottleneck for evaluated applications [2], it imposes great challenges for computing science researchers to resolve this issue by reducing the volume of data to be transferred as much as possible.

Hadoop system [3], a data-intensive system with chunk-based storage, is designed to solve these challenges caused by the increasing volume of data that computer-intensive clusters could hardly handle. Data are split into blocks and stored randomly across the data nodes. Data are then divided into

splits according to the user input, the tasks are distributed to the node storing the largest part of the split. When tasks are launched on the node, records are read from the split according to the input format. However, blocks are not equal to records. There is a gap. Records consist of several blocks which are distributed across the nodes as shown in Fig. 1.

As shown in Fig. 1, the input data as a record could be several blocks randomly scattered among nodes. Since the relationship between them is not considered on current storage mechanisms, blocks from the same node are randomly stored across the data nodes, then computations over records with several blocks will cause extra data transfer to gather the input data.

To address this problem, we design Record-Based Block Distribution (RBBD) which optimizes block distribution based on the relationship of the records and blocks by storing all the blocks from the same record on the same node as much as possible to reduce data transfer across nodes. We use a weighted set cover scheduling (WSCS) to reduce transfer volume. Figure 2 shows the high-level system view with RBBD and WSCS. Although, the original implementation works well in situations with records residing on one block, our experiments show that it can lead to severe performance problems when its underlying assumptions are broken. Our strategy improves co-located computation by reducing the actual data amount across network by a factor of 36.37 %.

When prior knowledge of the record and block relation is hard to retrieve, WSCS is adopted. The scheduling algorithm utilizes the information of (1) available slots for scheduling, (2) locations of file chunks and (3) the network transfer latency of multiple chunks to determine the cost of each candidate node for the map task and picks the best candidate for scheduling. When prior knowledge of the record and block relation is accessible, we use RBBD together with WSCS to reduce the extra data transfer. RBBD distributes the blocks of the same record onto the same node at the time when the data are uploaded to the storage system.

The rest of this paper is organized as follows: Sect. 2 states the related research in the data-intensive analysis. Sect. 3 details the design and implementation of RBBD. Section 4 demonstrates the weighted set cover approach. Section 5 presents algorithms, results and analysis respectively. Finally, Sect. 6 concludes the paper and introduces our plan for the future work.

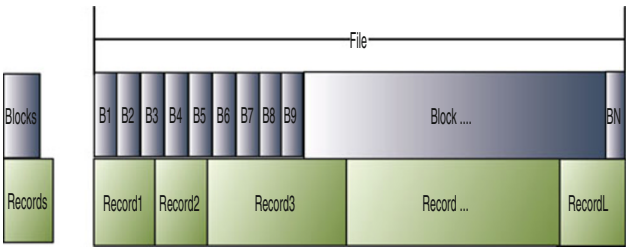


Fig. 1 Record and blocks demonstration over a single file

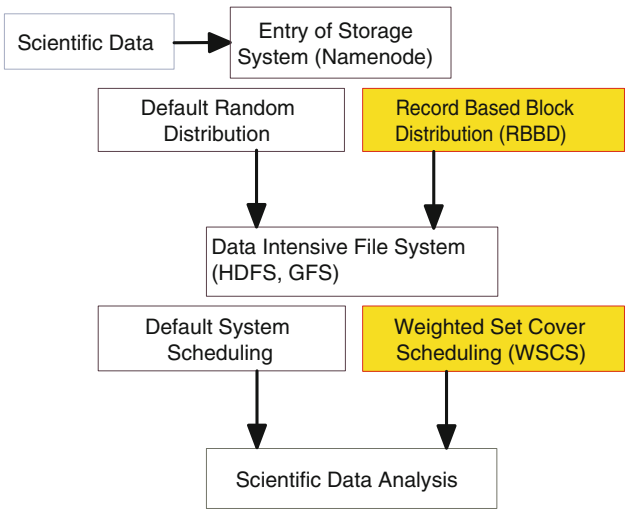


Fig. 2 High-level system view with RBBD and WSCS

2 Related works

There have been a lot of large-scale data processing frameworks in the scientific research area. MapReduce [4], Bigtable [5], Dryad [6] and many storage systems are contributing to the storage and analysis of the large-scale data. Dryad is a research project at Microsoft Research for the general-purpose runtime for execution of data parallel applications. However, our approach is for the optimization of the data layout distribution and scheduling improvement on data-intensive clusters, of which the typical system is Hadoop. Some approaches contribute to the scheduling improvement of MapReduce performance such as the performance improvement research in Hetterogeneous Environment [7] and Mantri [8]. Mantri proposes a solution to improve the MapReduce performance by improving the outliers, such as using network-aware placement of tasks, smart restart of outliers and protecting outputs of valuable tasks. In Mantri, they do not address the extra data transfer caused by the random block placement, rather they focus on the task monitoring and management.

Binary files are widely used in scientific applications especially as the simulation output. Structured format such as NetCDF [9], HDF5 [10] and Gadget [11] applies to a lot of scientific data with binary format. There is a project, SciHadoop, that explores the porting of NetCDF on MapReduce framework [12]. However, The characteristic of binary file processing is that the whole record should be inputted as a whole, otherwise the data do not mean anything. Our RBBD

framework distributes the blocks according to the predefined metadata and processes them as a whole, which makes the analysis proceed correctly.

For the storage research on Hadoop systems, such as intermediate storage research [13–15]. Research of Guo et al. [15] specifically aims to achieve optimal data locality by a cost matrix model with scheduling problem and improves the default Hadoop scheduling strategy. However, their solution does not take into account the difference of the data access patterns between binary files and common files.

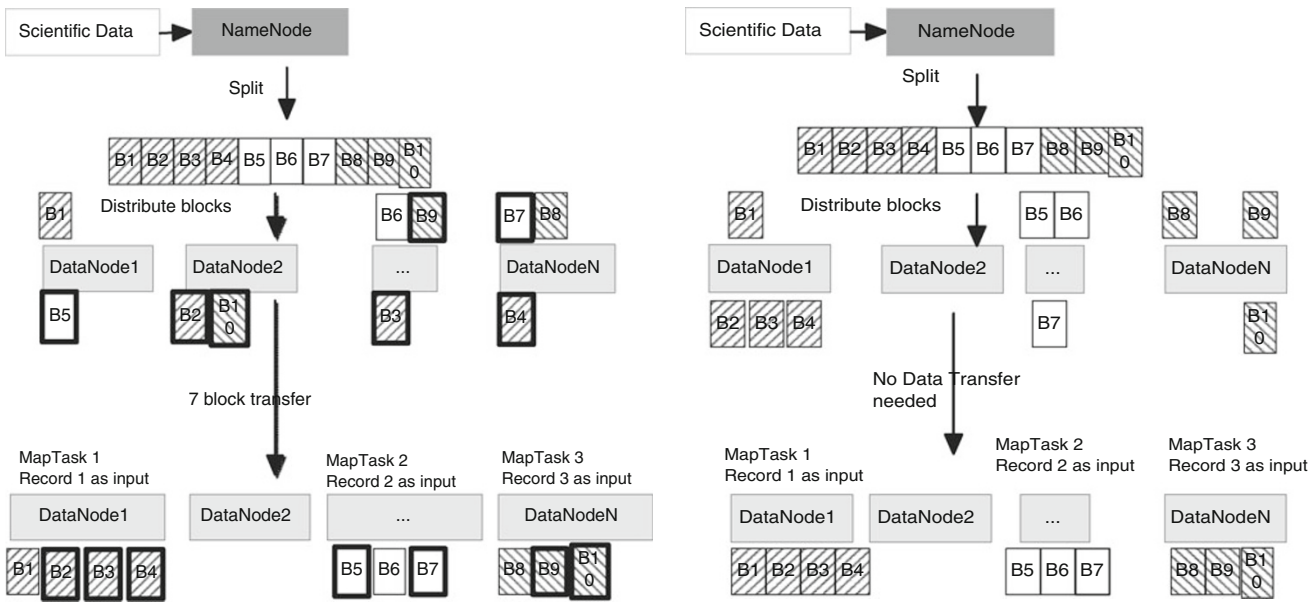
There are also researches on the access patterns for MapReduce framework such as MRAP [16]. The author made a design change for the MapReduce by allowing the Map function to access two inputs and they also improved the data layout according to the data semantics to erase the gap between scientific data and HDFS storage format. However, our work is to erase the gap between the block and the actual map input as a record, optimize the block distribution from the view of the record, and is designed to overcome the unsplittable feature of the binary file.

3 Record-Based Block Distribution (RBBD)

In this section, we design RBBD at HDFS management level which optimizes the block distribution according to the record/block relationship. There are two access patterns for binary files with/without prior knowledge of the record/block relationship. RBBD only applies to the access pattern with prior knowledge. We use the uploading middleware (UPMIDD) to perform the block optimization at the time when the data are uploaded to the storage system. Compared to the existing HDFS block management, our RBBD eliminates the gap between the records and blocks, requiring no data transfer when performing the Map task. To make it simple, we use 1-replica to demonstrate how we will improve the HDFS block distribution as shown in Fig. 3. The general steps are as follows:

1. The user makes the request to upload certain files together with metadata.
2. Files are split into blocks of size set in the HDFS system configure file, the last block can be less than that.
3. Name node uses UPMIDD to retrieve record/block relationship.
4. Name node distributes at least blocks of the same record onto the same node, evenly distributes the block packs.
5. The user requests to perform analysis against the uploaded files.
6. Name node schedules the Map tasks to data nodes with all available blocks of the same record.
7. Data nodes perform the Map task, because all needed data are already on them.

Suppose there are N data nodes, the default block size is B bytes, the file to be uploaded comprises M blocks, and the size of the Mth block is $\leqslant B$ bytes, to make it simple to analyze, we assume that the block belongs to only one record, no overlap among blocks of different records.



(a) Default block distribution

(b) RBBD demonstration

Fig. 3 A detailed view of comparing HDFS default block distribution and our RBBD

1. We learned from the metadata that the number of blocks for the R records is an array as below:

$$R[R] = R_0, R_1, R_2, \ldots, R_{R-1} \tag{1}$$

2. For the random replacement, the probability of x blocks are placed on the same node is:

$$p = (1/N)^x \tag{2}$$

3. The chance that the blocks are placed as a record-based through random placement is:

$$p_{\text{opti}} = \prod_{j=0}^{R-1} (1/N)^{R[j]} = (1/N)^{\sum_{j=0}^{R[j]}} \tag{3}$$

4. Since it is assumed that there is no overlap, then the total number of blocks M equals $\sum_{j=0}^{R[j]}$, and the (3) is simplified to:

$$p_{\text{opti}} == (1/N)^M \tag{4}$$

Hence, the possibility of the random placement to achieve the optimal data placement according to the record is defined by (4). It is clear that the possibility is related to the number of blocks and the number of data nodes. We learn that the possibility of random placement decreases along with the increasing number of nodes in the Hadoop cluster and the increasing number of data blocks. Based on our analysis, for a small scale cluster (our test bed which only has 15 data nodes), when the number of blocks is larger than 40 (total data size is more than $2,560MB$), it is highly unlikely that (possibility = 10^{-47}) the random data placement will achieve an optimal data lay out. Unfortunately, most data-intensive applications work on even more data nodes and even more number of blocks, which eventually would further decrease the possibility.

We will discuss UPMIDD in detail in the next subsection.

3.1 Uploading Middleware

UPMIDD is a middleware in charge of block optimization according to the metadata provided by the user when users request to upload the data set. In HDFS, NameNode maintains the blocks and file relationship. What UPMIDD does is to replace the default random block distribution mechanism of HDFS, and optimize the distribution according to the metadata. The purpose of this middleware is to reduce the data transfer caused by the random placement of blocks.

When a client uploads data to the HDFS, the name node determines the record-based storage by the input parameter isRecordBased, if isRecordBased is true, then this upload will be completed by UPMIDD. The UPMIDD reads the user-provided metadata with record delimitation information, and then determines the block/record relationship together with the default block size and distributes the blocks of the same record on to the same node. The workflow of UPMIDD is shown in Fig. 4. isRecordBased is also used to determine whether this file needs to be distributed according to the metadata and the metadata are also stored in HDFS for use when the user requested rebalance on HDFS. Suppose that the user provides the below metadata:

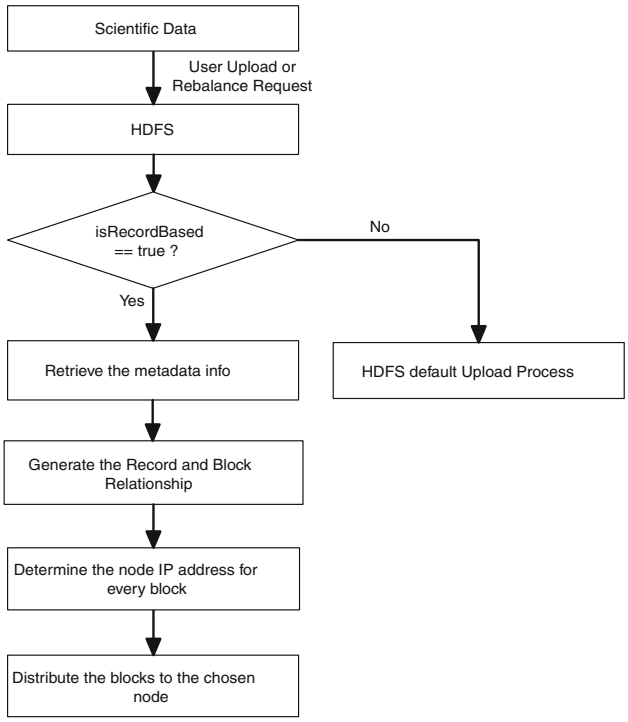


Fig. 4 RBBD flow chart

```
#recordId startoffset endoffset
1              N1          N2
2              N3          N4
3              N5          N6
.............................
M              N(2M-1)     N(2M)
```

We assume that the block size set for this HDFS system is B. UPMIDD then distributes the blocks as Algorithm 1.

Algorithm 1 Pseudocode for UPMIDD using metadata

Input: Metadata provided by the user specifying the record delimitation as shown above and the block size in the HDFS configuration file.
Output: Optimized distribution according to the record/block relationship.
Steps:

1. Define two data structures Block and Record:
   class Block { int blockId; int recordId; int startOffset; int endoffset; int nodeIp; }
   class Record { int[] blockIdList; int recordId; int startOffset; int endoffset; }
2. Get the block list
   Block: B1[0, S-1], B2[S, 2S-1], ..., BX[XS, N(2M)];
3. Retrieve the record/block relationship matrix;
4. Retrieve the online data nodes information and set the node IP address into the data structure RecordBlock.nodeIp;
5. Distribute the blocks to the node with ip address of RecordBlock.nodeIp.

4 Weighted Set Cover Scheduling (WSCS)

When MapReduce tasks are received by the NameNode, if there are M map tasks launched on an application then, there is a set $S = \{s_1, s_2, \ldots, s_M\}$ such that s_i represents a data split consisting multiple blocks and multiple records, and will be processed by the ith map task. The NameNode returns all of the nodes containing a data block (including the replica nodes) per block of the input files to the job scheduler. For example, if there are three replicas, then three nodes will be returned for a specific block. In case of multiple blocks, the total number of nodes returned is equal to three times the number of blocks. We call this set K, and K_i will correspond to all the nodes that contain blocks from the input files.

We want to find a set ($A \subset K$) of nodes, where one of the nodes is the primary node to host/schedule the map task, and others are the secondary nodes and will provide the missing data to the primary node. The cost of data transfer from the secondary nodes to the primary node should be at a minimum. This node- selection problem is similar to the weighted set cover problem, which models many resource-selection problems. A split and a node in the scheduling problem exactly corresponds to an element and a set, respectively, in the weighted set cover problem. The only difference is that "a split also represents a set of records, not a single element". Each node contains at least one of the records. We need to find the set of nodes for each split. The weighted set cover problem has been proven to be NP-hard so that a heuristic and iterative algorithm is generally used to solve it. The algorithm starts by retrieving all the nodes containing the blocks from a record. It then starts the iteration with an empty set, A_i, which denotes a collection of the nodes selected until it reaches the last iteration. At each iteration, the algorithm selects a node n_i, and adds it to A_i. Nodes that are added to the set A_i are considered to be covering a part of split s_i. The algorithm finishes the iteration when all the blocks in the s_i are covered, and then A_i gives a set of nodes, where the first one is used as a primary node and others as secondary nodes.

Algorithm 2 Pseudocode for scheduling using Weighted Set Cover Problem

Input: Set S, consisting of splits required by M map tasks in a MapReduce Application: $S = \{s_i, s_j, ..., s_k, ...|0 < i, j, k <= M\}$; Set C, nodes with the information that data block distribution on each node n_i: $C = \{n_1, n_2, ..., n_N\}$.
Output: Find a set A for all splits in S such that $A_i \subseteq C$, and A_i has nodes that completely cover all the blocks in split s_i.
Steps: $A = \{A_i, A_j, ..., A_k, ...\}$, A and all A_x are empty, we will start computing each A_x and add it to set A. For i from 1 to M, [Iterate through all the elements in set S] A_i = empty 1) Compute K_i, This will be a list of nodes with required data blocks. Data blocks will be located on multiple nodes. 2) Compute w for all the nodes in K_i. w is the weight assigned to each block as explain earlier. Also, mark the nodes which will be used for the data transfer. 3) Find the Block contribution of each node in K_i. 4) Compute the price as $\frac{w}{\|BlockContribution\|}$ for each node in K_i. 5) Sort the values in ascending order, the first value will correspond to the primary node. Add the node to A_i. 6) Select the secondary nodes for a primary node based on the calculations in step 2. Add the nodes to A_i. 7) $A = A \bigcup A_i$

This algorithm (Algorithm 2) is used for all map tasks, and determines the optimal nodes for all map tasks belonging to one application.

Table 1 CASS cluster configuration

Fourteen compute/data nodes and 1 name node	
Make and model	Dell PowerEdge 1950
CPU	2 Intel Xeon 5140, Dual Core, 2.33 GHz
RAM	4.0 GB DDR2, PC2-5300, 667 MHz
Internal HD	2 SATA 500GB (7200 RPM) or 2 SAS 147GB (15K RPM)
Network connection	Intel Pro/1000 NIC
Operating system	Rocks 5.0 (Cent OS 5.1), Kernel:2.6.18-53.1.14.e15

5 Experimental results and analysis

In our experiments, we demonstrate how RBBD framework performs for the two access patterns of binary files as described in Sect. 3. The first access pattern of the binary files performs weighted set cover scheduling over the existent binary files with no prior metadata available on HDFS, whereas the second access pattern deals with the binary files with prior knowledge of the metadata about the record and block relationship. We also show the data transfer improvement due to block-optimized distribution for the second access pattern. The most challenging part of this work is to fairly evaluate RBBD framework using various data access patterns against the existing access patterns currently implemented in HDFS. Unfortunately, it turned out that most binary analytics programs need to be developed. In addition, there are no matured benchmarks for us to test our design. For the first access pattern, we simulate the scheduling algorithm with comprehensive experiments. We have used the Halo Finding application from the astrophysics domain, Gadget II [17] binary format and data in GeoTIFF [18] format produced by Beijing-1 Satellite provided by Center for Earth Observation and Digital Earth of Chinese Academy of Sciences (CAS) to evaluate the RBBD framework. We use both HDFS default storing implementation and our RBBD to store the blocks across the data nodes and compare the network transfer. In the next subsection, we will describe the testbed and benchmark setup used to generate results.

5.1 Test bed

Our test bed consists a total of 15 totally heterogenous nodes with Hadoop 0.20.1 installed. All these nodes are in the same rack, and the configurations for the Hadoop cluster is shown in Table 1. One node is set as the NameNode and JobTracker, and other 14 nodes are configured to be DataNodes and TaskTrackers.

5.2 Simulation of WSCS

We simulate the scheduling algorithm with comprehensive experiments. There are two steps in our simulation: (1) set up environments with different configurations; (2) test the performance of random algorithm and our scheduling algorithm, and compare them with that of the ideal solution.

In the first step, we simulate a large-scale cluster consisting of 100 racks, 5, 000 nodes in total. We also create a global routing table followed the instructions on ["TCP/IP network administration, p 146–148"] and the metrics distribution observed in the real routing table for our campus network. In the widely used Routing Information Protocol (RIP) and other dynamic routing protocols (such as OSPF, OSI, and RIPng), the metric in a routing table denotes the cost of the path through which the packet is to be sent. To focus the efficiency of our scheduling algorithm, we do not consider the local transfer latency caused by disk I/O or cache miss; the only transfer latency for a chunk is the network latency denoted by the metric in the routing table. We assign a number of different types of routine costs following a uniform distribution. The cost starts from 0 ms (in the same rack), the increasing step is 10 ms (based on the observations from part of the routing tables in our campus network). For simplicity, we do not simulate the dynamic updates in the routing table. The number of replicas for each chunk is set to be 3 as default in HDFS. Every rack contains the same number of nodes. All chunks are distributed into the nodes based on the following rules: any two of the three replicas should not be in the same node; each replica is randomly assigned to a rack.

In the second step, we implement a random scheduling algorithm and our weighted set covering based scheduling algorithm. In both algorithms, we randomly pick one rack as the primary rack on which the read request is issued. Then, we check the routing table to obtain the network information (the transfer latency starting from primary rack). For the random algorithm, we randomly pick the racks from those who have the required chunks until all of the chunks are read. The algorithm is iterated 40 times and we get the average transfer latency. For weighted set covering algorithm, we use greedy algorithm to find the optimal or quasi-optimal solution which gives the combination of racks providing all of the required chunks with the smallest transfer latency. Besides these two solutions, we enumerate all the possible solutions and get the ideal one to prove the accuracy of our solution. We set up five different experimental environments. In the first three experiments, we use four types of network latencies (0–30 ms, 10 ms for one step) but a different number of required chunks. The purpose is to show how different request sizes affect the performance of different algorithms; in the last two experiments, we use the same number of required chunks, but vary the number of network latencies (8 types, 16 types) to explore the impact of different sizes of network. The results show that increasing the size of requests (the number of requested chunks) does not significantly increase the transfer latency. The reason is that since the transfer latency between any two nodes in the same rack is negligible, and the transfer latency from a node in one rack to any of the nodes in another rack is the same, we can say that transfer latency between any two nodes is same as the transfer latency between the two racks. As a result, the data transfers in our experiment actually is incurred among racks. Since there are only 100 racks, but there are many requested blocks, every rack will maintain some of the requested blocks. In other words, the possible candidate set for solution always has all of the racks (100 in our experiments). Our weighted set covering algorithm always chooses the optimal or quasi-optimal combination of racks from the same candidate set. As a result, the overall performances are similar. The small difference is due to the different primary rack we randomly picked. The results in Fig. 5 show that along with the increasing size of the network, the average transfer latency will obviously increase. This is caused by the higher network latencies that we add (50, 60 ms and so on). No matter what kind of configuration we set, our weighted set covering algorithm outperforms the random algorithm 51–62 %; the deviation from the ideal solutions is no more than 6.8 %.

5.3 Results of RBBD

In this section, we present three sets of results for RBBD: the default HDFS data block placement, the comparison of the network transfer volume of Map execution under the default HDFS storage strategy, and the RBBD storage strategy; the overhead of performing RBBD.

5.3.1 The block distribution

The random block distribution of the HDFS is adopted to balance the work load of data nodes. However, it also causes problems for cases we mentioned above where the record comprises several blocks. In this experiment, we used the GeoTIFF data provided by CAS to perform the block distribution test. We ran experiments for data with different record/block relationships to see how the random block distribution would turn out. We ran five groups of experiments: one file with records of which the average number of blocks of a record is 2, 4, 8, 10 and 16. We get the block distribution as Fig. 6. Firstly, we retrieve the average number of blocks to be transferred as shown in Fig. 6a for different sizes of data. To show the actual data transfer rate for every node, Fig. 6b shows the average number of blocks to be transferred on the data node performing the actual map task for different sizes of data, and average number of blocks of one record is 4. From Fig. 6a, for the data of the same size, if the average number of blocks of the record is equal to or larger than the

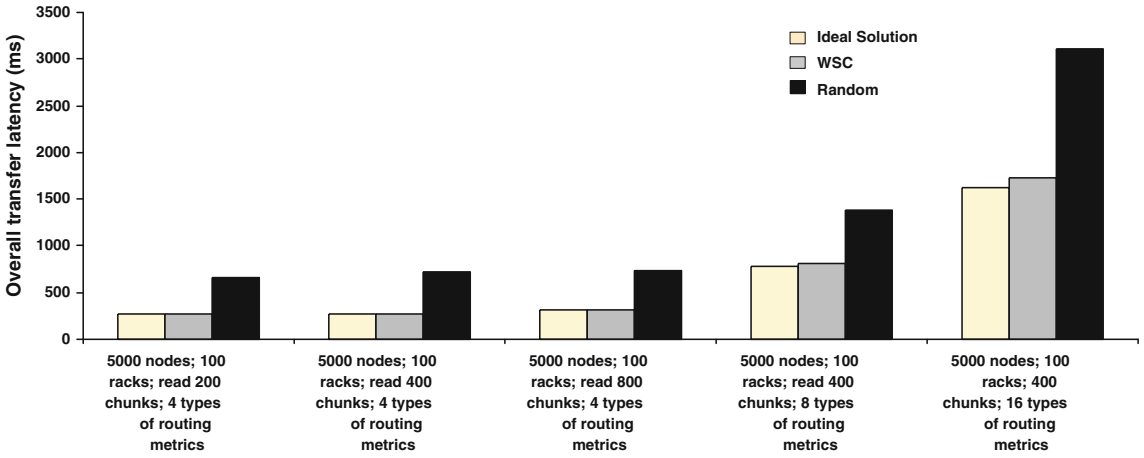
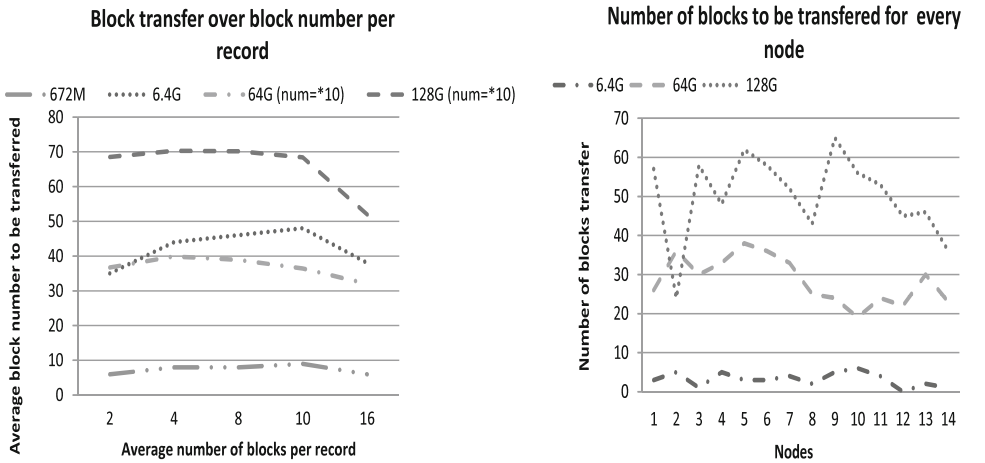


Fig. 5 Five experiments with different configurations



(a) Default block distribution **(b)** Block transfer across data nodes

Fig. 6 Random block distribution of different record/block relationships

total number of data nodes, there is a sharp reduction in the average block number to be transferred, because at that time, the possibility of the 16 blocks of the same record to be on the same node (across 14 data nodes) increases.

5.3.2 Comparison of network transfer

In this experiment, we tested the network transfer using the Gadget II data. By default, the HDFS stores the blocks randomly across the data nodes, but RBBD optimized the blocks according to the block/record relationship to keep one replica of all blocks to the same record on a same node. We made experiments on datasets of size 28.8, 57.6, 115.2, 172.8, 230.4, 288 and 576 GB, uploading them to the HDFS to retrieve the block distribution diagram as shown in Fig. 7.

If we use the WSCS to make tasks locate all records of the nodes, we learn that optimizing the block distribution according to the block/record relationship, can reduce the amount of transferred data by 196.10 GB for a 576-GB dataset. We can reduce the data transfer volume to an average of 36.37 %. For data-intensive computations, the data volume has been increasing to tera bytes, and the transfer has always been the important factor of influencing the performance. For example, if we have a data set of size 1 TB, then by optimizing the block distribution, we can get an amount of 363.7 GB reduced in data transfer.

5.3.3 Real application

We use a mass analyzer working with astrophysics data sets [17] for Halo Finding to perform the experiments. There are positions and velocities of particles in these data sets. Each particle has the following attributes: (x, y, z) for the position coordinate and (v_x, v_y, v_z) for the velocity, and particle mass M, particle tag T. The mass analyzer reads these data and calculates the average mass in each area with a predefined

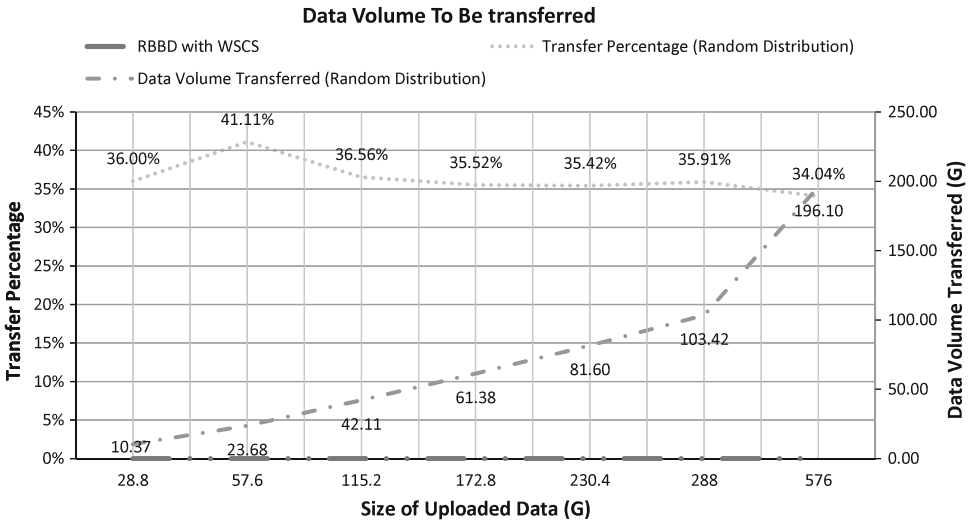


Fig. 7 Data volume transferring during map computation

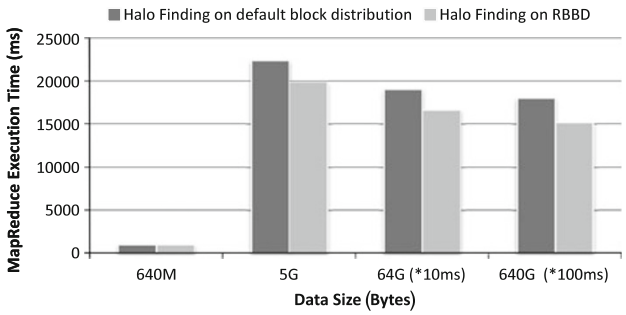


Fig. 8 MapReduce execution time comparison between default HDFS and RBBD

area size. Since the particle information as a whole input is very large to fit in one analysis program, the particles are divided into several records and computed, respectively. The records are of random size and stored across data nodes. We first execute the analysis using MapReduce with blocks randomly stored by default on HDFS, and then re-execute the same program again after the blocks are distributed across the nodes using RBBD and we got the comparison results shown in Fig. 8. We learned that with the increasing size of data, the improvement of execution increases. When the data size reaches 640 GB, we achieve an improvement of up to 20 %.

6 Conclusion

We have developed an extended HDFS uploading framework RBBD to allow users to specify data semantics for data-intensie analysis applications. Our approach reduces the overhead of data transfer caused by ignoring the record/block relationship for the default random block replacement. We provide functions and metadata templates to specify the record delimitations. We also studied the unsplittable feature of binary files; after specifying the delimitations through the metadata file, the program easily retrieves the correct input of the Map function. For experimentation, we ran a real application from astrophysics. Our results show an average data volume reduction rate of 36.37 %.

These tasks which access records of multiple blocks also map to the data nodes as much as possible with the help of WSCS by selecting the optimal nodes for scheduling map tasks on the basis of block locations retrieved from the name node. The research of improving block distribution in the dynamic HDFS is left for future work. In the future, we would implement the dynamic block balancing and scheduling schemes on a running Hadoop cluster. We would like to take into account of the intermediate data at Map phase to reduce data volume to be transferred and achieve a performance improvement.

Acknowledgments This work is supported in part by the US National Science Foundation Grant CCF-0811413, CNS-1115665, and National Science Foundation Early Career Award 0953946.

References

1. Zheng F, Abbasi H, Docan C, Lofstead J, Klasky S, Liu Q, Parashar M, Podhorszki N, Schwan K, Wolf M (2010) Predata: preparatory data analytics on peta-scale machines. doi:10.1.1.153.9094. http://citeseerx.ist.psu.edu/viewdoc/summary?. Accessed 23 July 2012
2. Fu B, Ren K, López J, Fink E, Gibson G (2010) Discfinder: a data-intensive scalable cluster finder for astrophysics. In: Proceedings of the 19th ACM international symposium on high performance distributed computing, ser. HPDC '10. New York, NY, USA, ACM, 2010, pp 348–351
3. Apache hadoop (2012) http://wiki.apache.org/hadoop/. Accessed 23 July 2012
4. Mapreduce (2012) http://en.wikipedia.org/wiki/mapreduce. Accessed 23 July 2012
5. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2006) Bigtable: a distributed storage system for structured data. In: Proceedings of the 7th

symposium on operating systems design and implementation, ser. OSDI '06. Berkeley, CA, USA, USENIX Association, 2006, pp 205–218. http://portal.acm.org/citation.cfm?id=1298455.1298475. Accessed 23 July 2012
6. Dryad (2012) http://research.microsoft.com/en-us/projects/dryad/. Accessed 23 July 2012
7. Konwinski A (2009) Improving mapreduce performance in heterogeneous environments: Master's thesis. EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-183.html. Accessed 23 July 2012
8. Ananthanarayanan G, Kandula S, Greenberg A, Stoica I, Lu Y, Saha B, Harris E (2010) Reining in the outliers in mapreduce clusters using mantri. In: Proceedings of the 9th USENIX conference on operating systems design and implementation, ser. OSDI'10, Berkeley, CA, USA, USENIX Association, 2010, pp 1–16. http://portal.acm.org/citation.cfm?id=1924943.1924962. Accessed 23 July 2012
9. Netcdf (2012) http://www.unidata.ucar.edu/software/netcdf/. Accessed 23 July 2012
10. Hdf (2012) http://www.hdfgroup.org/hdf5/. Accessed 23 July 2012
11. Gadget (2012) http://www.mpa-garching.mpg.de/gadget/. Accessed 23 July 2012
12. Scihadoop (2012) http://www.soe.ucsc.edu/share/technical-reports/2011/ucsc-soe-11-04.pdf. Accessed 23 July 2012
13. Ko SY, Hoque I, Cho B, Gupta I (2010) Making cloud intermediate data fault-tolerant. In: Proceedings of the 1st ACM symposium on cloud computing, ser. SoCC '10, New York, NY, USA, ACM, 2010, pp 181–192
14. Yuan D, Yang Y, Liu X, Chen J (2010) A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. http://hdl.handle.net/1959.3/88596. Accessed 23 July 2012
15. Guo Z, Fox G, Zhou M (2012) Investigation of data locality and fairness in mapreduce. In: Proceedings of third international workshop on MapReduce and its applications date, ser. MapReduce' 12. New York, NY, USA, ACM, 2012, pp 25–32
16. Sehrish S, Mackey G, Wang J, Bent J (2010) Mrap: a novel mapreduce-based framework to support hpc analytics applications with access patterns. In: Proceedings of the 19th ACM international symposium on high performance distributed computing, ser. HPDC '10, New York, NY, USA, ACM, 2010, pp 107–118
17. The cosmic data arxiv (2012) http://t8web.lanl.gov/people/heitmann/arxiv/. Accessed 23 July 2012
18. Geotiff (2012) http://trac.osgeo.org/geotiff/. Accessed 23 July 2012