Improving Performance of Map Reduce by Using Draw and Data Rebalancing Scheme

Naveen S[1], Dr. E. Poovammal[2]

1 Research scholar, Dept of Computer Science and Engineering, SRM University, Chennai, India,

2 Professor& head, Dept of Computer Science and Engineering, SRM University, Chennai, India,

Abstract

Map reduce has became an important distributed processing model for large scale data intensive application like dataminig, web indexing. Hadoop is an open source implementation of Mapreduce. For increasing the performance of parallelism, hadoop is having its own data placement algorithm, but we observe that many data intensive application exhibits interest locality, so hadoop data placement algorithm does not consider this interest locality. The default random placement algorithm does not perform well and is the way below the efficiency of optimal data distribution. So for this problem new Data-grouping-Aware (DRAW) data placement scheme came into existence. Draw will increase the performance of map reduce by 36.4% in comparison with hardoop's default random placement algorithm. Our aim is to still increase the performance of mapreduce. In Hadoop distributed file system the replication is done to entire file which requires more space, so this system uses the concept of replication of blocks know as data rebalancing scheme. Such a data rebalancing scheme can minimizes data transfer among slow and fast node in the cluster during the execution of Hadoop application. The data-replication approach has several limitations. Storing replica does require an unreasonably large amount of disks capacity, which in turn increases the cost of Hadoop clusters. So based on these limitations, we proposed data rebalancing scheme. In this scheme, only frequently accessed blocks should be replicated and it should be placed in a node which is not having the same block. In addition to Draw scheme, this proposed system implementing data rebalancing scheme. So that this will increase performance more than the previously used .

***Keywords**: Hadoop, Data Rebalancing scheme, Mapreduce, DRAW.*

1. Introduction

ApacheHadoop is an open-source software framework for storing and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top level project being built and used by a global community of contributors and users.
The Apache Hadoop framework is composed of the following modules:

1. Hadoop Distributed File system (HDFS) – a distributed file –system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster.
2. Hadoop Yarn – a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users application.
3. Hadoop Mapreduce – programming model for large scale data processing.

All the modules in hadoop are designed with a fundamental assumption that hardware failures are common and that should be automatically handled by the framework. Apache hadoop's Mapreduce and hdfs components are originally derived respectively from Google's Mapreduce and Google file system (GFS) papers. For the end-users, though Mapreduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig, Apache Hive among other related projects expose higher level user interfaces like Pig Latin and a SQL variant respectively. The emerging data intensive application place a demand on high performance computing resources with massive storage. Academic and pioneers have been developing big data parallel computing frameworks and large scale distributed file systems to facilitate high performance runs of data intensive applications such as bio informatics[20]. Astronomy [19], and high-energy physics [17]. In practice, many scientific and engineering applications have ***interest locality***: 1) domain scientists are only interested in a *subset* of the whole data set, and 2) scientists are likely to access one subset more frequently than others. For example, in the bioinformatics domain, X and Y chromosomes are related to the offspring's gender. Both chromosomes are often analyzed together in generic research rather than all the 24 human chromosomes [11]. Regarding other mammal's genome data pools, the chimpanzee is usually compared with human [14], Another example is, in the climate modeling and forecasting domain, some scientists are only interested in some specific time periods [23]. In summary, these co-related data have high possibility to be processed as a group by specific domain applications. Here, we formally define the "**data grouping**" to represent the possibility of two or more data (e.g., blocks in Hadoop) to be accessed as a group. Such data grouping can be quantified by a ***weight***: *a count that* these data have already been accessed *as a group*. The potential assumption is that if two pieces of data have been already accessed together for many times, it is highly possible for them to be accessed as a group in future.

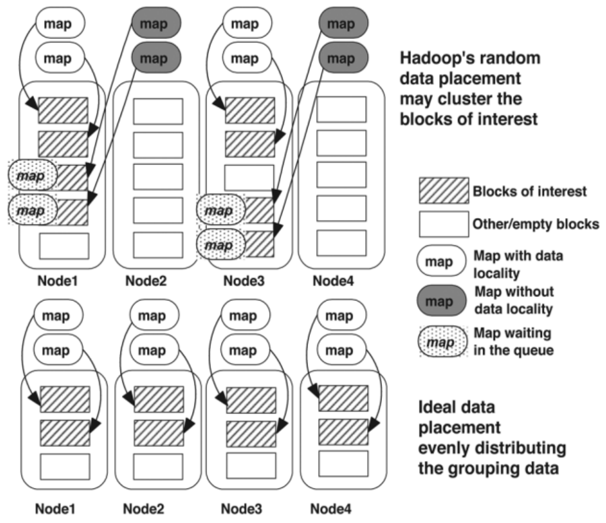

map
Node1
Node2
Node3
Node4
Hadoop's random data placement may cluster the blocks of interest
Blocks of interest
Other/empty blocks
Map with data locality
Map without data locality
Map waiting in the queue
Ideal data placement evenly distributing the grouping data

Fig. 1. Simple case showing the efficiency of data placement for MapReduce programs.

By using hadoop's default random placement strategy, the overall data distribution may be balanced, but there is no guarantee that the data accessed as a group is evenly distributed. For example if a group of data stored in a single node, then all the map task will assign on same node or they will schedule on the other node by accessing data remotely or they will schedule on the same node which they have to wait for some time because those blocks are already accessing by other Map tasks. This kind of situations will degrade the performance of map reduce, so to avoid this we have find the grouped data which were accessed by the Map tasks and should distribute those blocks in different nodes so that performance can be increased. Dynamic data grouping is a effective for exploiting the predictability of data access patterns and improving the performance of distributed file systems [10]. We show an example in fig.1: if grouping data are distributed by hadoop's random strategy, the shaded map tasks with either remote data access or queuing delay are the performance barriers;. Now we briefly analyze the possibility for random data distribution to evenly distribute the same data from the same group. Our observation shows this possibility is affected by three factors:

1. The number of replica for each data block in each rack(NR);
2. The maximum number of simultaneous map tasks on each node (NS)
3. The data grouping access patterns. It is easy to conclude that the larger default random solution will achieve the optimal distribution:

a) If suppose NR is extremely large, eg., the number of replica for each data is same as number of nodes in cluster therefore we can achieve maximum parallelism but here there is a drawback that if NR is maximum then it will take more space

b) Assume NS(maximum number of simultaneous map tasks) is extremely large, then waiting time will be more for e.g., Amazon's EC2 and s3 [1],[25]; is limited by the hardware capacity. Moreover, the data grouping is not considered in default Hadoop, which results in a non optimal data placement strategy for the data-intensive application. To achive optimality Data-Grouping-Aware data placement scheme (DRAW)[26] was used . These Draw was proposed by jun wang, Qiangju Xiao, Jiangling Yin, Pengju Shang . The data grouping effects to significantly improve the performance for data-intensive applications with interest locality. Without loss of generality, we need to reduce the waiting time and also to minimize total space usage without replicating all files. For this we proposed Data Rebalancing scheme i.e, only frequently accessed data blocks should be replicated. Data rebalancing scheme was used.

With real world genome indexing [2] and astrophysics applications [9], DRAW is able to execute up to 59.8% more local map tasks in comparison with random placement. In addition, Data Rebalancing scheme reduces the completion time of map phase by more than 41.7% and the Map Reduce task more than 36.4%

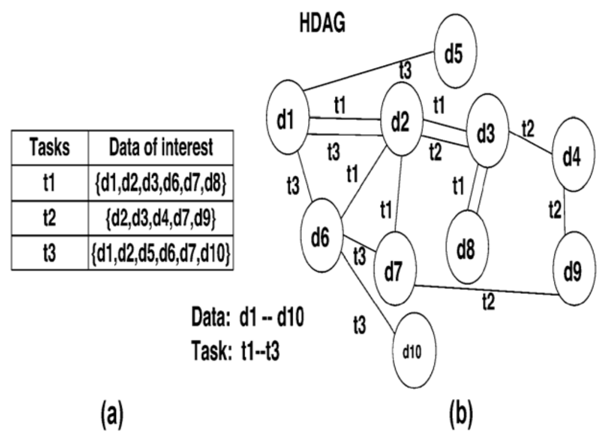
2. System Analysis

2.1 Data –grouping aware data placement algorithm

In this we explained about how DRAW can be used at rack level, which optimizes the grouping-data distribution inside a rack. Before going to start DRAW we need to exploit the Name Node log file to analyze which task is accessing which data, so HDAG will help to analyze them. A data access history graph (HDAG) to exploit system log files to know the data grouping information.

A. History data access graph (HDAG)

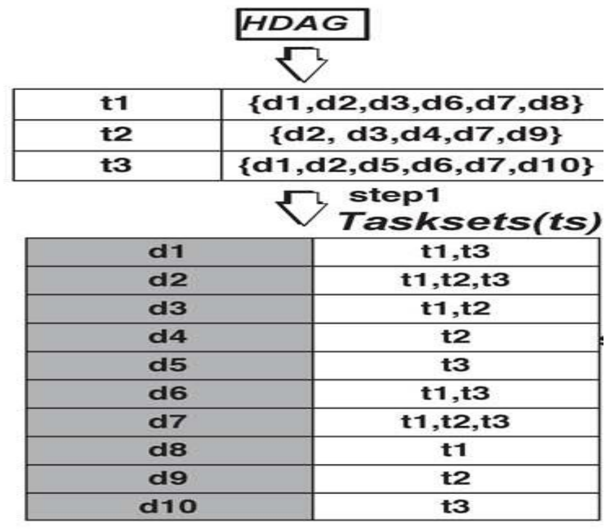
HDAG is a graph describing the access patterns among the files, which can be legend from the history of data accessed .In each hadoop cluster rack, Name Node maintains System logs recording every system operation including all files which have been accessed. By monitoring these files we can exploit mappings between tasks and files to accurately learn the file access Patterns. Note that in hadoop clusters, files are spilt in to blocks which is the basic data distribution unit; hence we need to translate the grouping information at file level in to block level fortunately, the mapping information between files and blocks can be found in the NameNode.Fig.2 shows an example of HDAG: given three Map Reduce tasks, t1 access d1,d2,d3,d6,d7,d8 here d is block d2,d3,d4,d7,d9: t3 access d1,d2,d5,d6,d7,d10.



Tasks	Data of interest
t1	{d1,d2,d3,d6,d7,d8}
t2	{d2,d3,d4,d7,d9}
t3	{d1,d2,d5,d6,d7,d10}

Fig.2. Example showing HDAG.

The accessing information initially generated from the log files is shown as Fig. 2(a). There after we can easily translate the table into the HDAG shown as Fig. 2(b). This translation step makes it easier to generate the grouping Matrix for the next step and also here with these HDAG we can know which task is accessing which data block .It will use full to replicate frequently accessed data block.



HDAG
step1
Tasksets(ts)

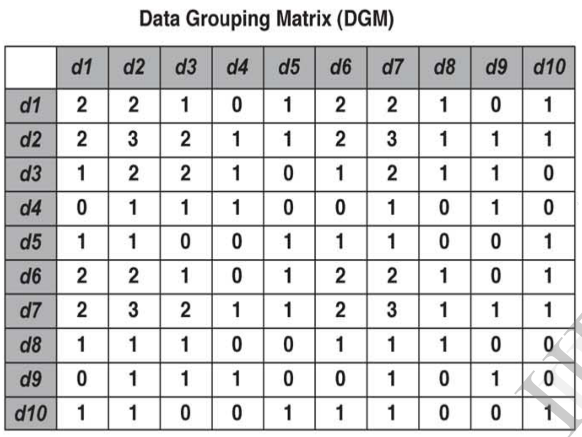
t1	{d1,d2,d3,d6,d7,d8}
t2	{d2, d3,d4,d7,d9}
t3	{d1,d2,d5,d6,d7,d10}

d1	t1,t3
d2	t1,t2,t3
d3	t1,t2
d4	t2
d5	t3
d6	t1,t3
d7	t1,t2,t3
d8	t1
d9	t2
d10	t3

Fig.2.(c) example showing the table of accessed blocks.

B.DGM (Data Grouping Matrix)

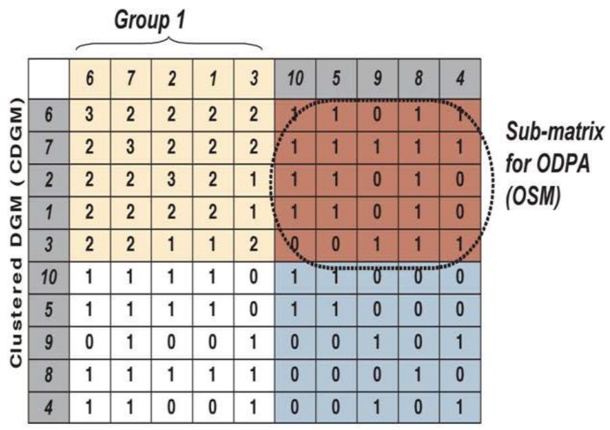
Based on HDAG, we can generate a DGM showing the relation between every two data blocks. Given the same example as shown in Fig. 2, we can construct the DGM as shown in Fig. 2(d), where each element DGM i,j=grouping i,j can be calculated by counting the tasks in common between task sets of *t*si and *t*sj. The elements in the diagonal of the DGM show the number of tasks that have used this data. In DRAW, DGM is a by matrix, where is the number of existing blocks. DGM will show the relationship among the data groups


Data Grouping Matrix (DGM)

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
d1	2	2	1	0	1	2	2	1	0	1
d2	2	3	2	1	1	2	3	1	1	1
d3	1	2	2	1	0	1	2	1	1	0
d4	0	1	1	1	0	0	1	0	1	0
d5	1	1	0	0	1	1	1	0	0	1
d6	2	2	1	0	1	2	2	1	0	1
d7	2	3	2	1	1	2	3	1	1	1
d8	1	1	1	0	0	1	1	1	0	0
d9	0	1	1	1	0	0	1	0	1	0
d10	1	1	0	0	1	1	1	0	0	1

Fig 2(d) example for Data Grouping Matrix formed from HDAG

at the same time; the grouping weight in the DGM denotes "how likely" one data should be grouped with another data. After knowing the DGM in Fig.2(c), we should form a cluster. For that Specifically, Bond Energy Algorithm (BEA) is used to transform the DGM to the clustered data grouping matrix (CDGM). Since a weighted matrix clustering problem is N-P hard, the time complexity to obtain the optimized solution is *O(n^n)*. The BEA algorithm saves the computing cost by finding the suboptimal solution in time *O(n^2)*[13]; it has been widely utilized in distributed database systems for the vertical partition of large tables [18] and matrix clustering work [13]. The BEA (bond energy algorithm) algorithm clusters highly related data and helps the data to evenly distribute among nodes.


Group 1
Clustered DGM (CDGM)
Sub-matrix for ODPA (OSM)

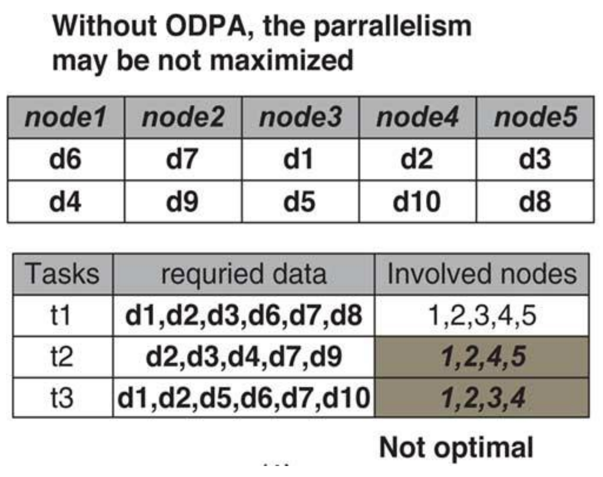
	6	7	2	1	3	10	5	9	8	4
6	3	2	2	2	2	1	1	0	1	1
7	2	3	2	2	2	1	1	1	1	1
2	2	2	3	2	1	1	1	0	1	0
1	2	2	2	2	1	1	1	0	1	0
3	2	2	1	1	2	0	0	1	1	1
10	1	1	1	1	0	1	1	0	0	0
5	1	1	1	1	0	1	1	0	0	0
9	0	1	0	0	1	0	0	1	0	1
8	1	1	1	1	1	0	0	0	1	0
4	1	1	0	0	1	0	0	1	0	1

Fig .2(e) example showing for CDGM(cluster data grouping matrix)

After generating CDGM, we can take OSM as group 2. In this case, group 1 and group 2 represent most related data sets. Assuming there are 5 *Data Nodes* in the Hadoop cluster, the CDGM in Fig.2 (e) shows data {6, 7, 2, 1, 3} as group 1 and {4, 9, 5, 10, 8} as group 2 should be evenly distributed when placed on the 5 nodes. But in Fig2 (f) total we have only 10 pieces of data in our example, after knowing that {6, 7, 2, 1, 3} should be placed as a group (horizontally), it is natural to treat the left data {4, 9, 5, 10, 8} as another group. Hence OSM are not necessary for our case, but when placing group 2 we can see that it s not evenly distribute you can see in fig 2(f) so for this we need another step to do i.e., ODPA.

C. OPDA (Optimal Data Placement Algorithm)

Knowing the data groups alone is not enough to achieve the optimal data placement. Given the same example from Fig.2 (f), random placing of each group, as shown in Fig. 2 (f), task 2 and task 3 can only run on 4 nodes rather than 5, which is not optimal. This is because the above data grouping only considers the horizontal relationships among the data in DGM, So it is also necessary to make sure those blocks are on the same node that have minimal chance to be in the same group (vertical relationships). In order to obtain this information, we can use an algorithm named ODPA.



Without ODPA, the parrallelism may be not maximized

node1	node2	node3	node4	node5
d6	d7	d1	d2	d3
d4	d9	d5	d10	d8

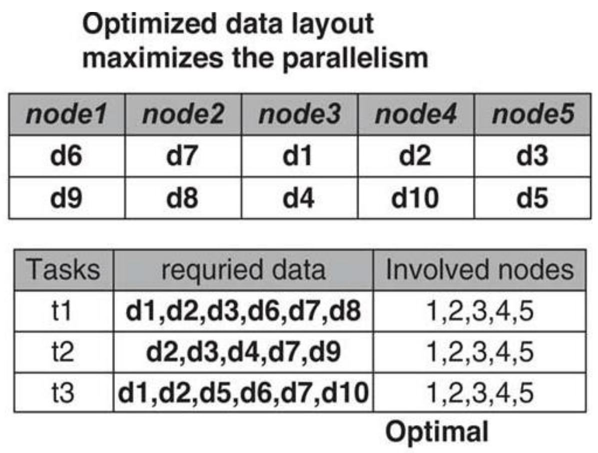
Tasks	requried data	Involved nodes
t1	d1,d2,d3,d6,d7,d8	1,2,3,4,5
t2	d2,d3,d4,d7,d9	1,2,4,5
t3	d1,d2,d5,d6,d7,d10	1,2,3,4

Not optimal

Fig 2(f) example random placing of each group

1. ODPA is based on sub matrix for ODPA (OSM) from CDGM. OSM indicates the dependencies among the data already placed and the ones being placed. For example, the OSM in Fig. 2(e) denotes the vertical relations between two different groups (group1:6, 7, 2, 1, 3 and group2:4, 9, 5, 10, 8). Take the OSM from Fig. 2(e) as an example, The ODPA algorithm starts from the first row in OSM, whose row index is 6. Because there is only one minimum value 0 in column 9, we assign DP [6]={6,9} we assign , which means data 6 and 9 should be placed on the same data node because 9 is the least relevant data to 6. When checking row 7, there are five equal minimum values, which means any of these five data are equally related on data 7. To choose the optimal candidate among these five candidates, we need to examine their dependencies to other already placed data, which is performed by the *for* loop calculating the for these five columns In our case, *sum* [8]=5 is the largest value; by placing 8 with 7 on the same node, we can, to the maximum extent, reduce the possibility of assigning it onto another related data block. Hence, a new tuple {7, 8} is added to DP. After doing the same processes to the rows with index 1,2,3 we have DP={{6,9}{7,8}{1,4}{2,5}{3,10}} indicating the data should be placed as shown in Fig.2 (g).Clearly, all the tasks can achieve the optimal. With the help of ODPA, DRAW can achieve the two goals:

1. Maximize the parallel distribution of the grouping data
2. Balance the overall storage loads.



Optimized data layout maximizes the parallelism

node1	node2	node3	node4	node5
d6	d7	d1	d2	d3
d9	d8	d4	d10	d5

Tasks	requried data	Involved nodes
t1	d1,d2,d3,d6,d7,d8	1,2,3,4,5
t2	d2,d3,d4,d7,d9	1,2,3,4,5
t3	d1,d2,d5,d6,d7,d10	1,2,3,4,5

Optimal

Fig.2(g) example for after applying OPDA.

2.2. Data rebalancing scheme

DRAW is designed for the applications showing interest locality, in addition to these our Data Rebalancing Scheme will be implemented .To implement these we need two things.

1. Information of frequently accessed data blocks by the users.
2. Blocks information in which nodes they resides.

By using file system check (FSCK) in hadoop, We can know the information of which block is resides in which *data node* and also by using HDAG we can analyze the name node log files to get the information of frequently accessed data blocks. In Fig.2(c) clearly its showing d1, d2, d3, d6, d7 are the blocks which are accessed more than 2 times, so these data blocks should be replicated and placed in a node which is not having the same block by checking with FSCK .So if more than two tasks came for same block it won't wait for certain amount of time . Why because already we replicated those blocks in different nodes, so they will access those replicated blocks by this waiting time of Map tasks will be reduced. After doing

DRAW we should apply Data Rebalancing Scheme. Hence performance of Mapreduce increased.

3. System Architecture

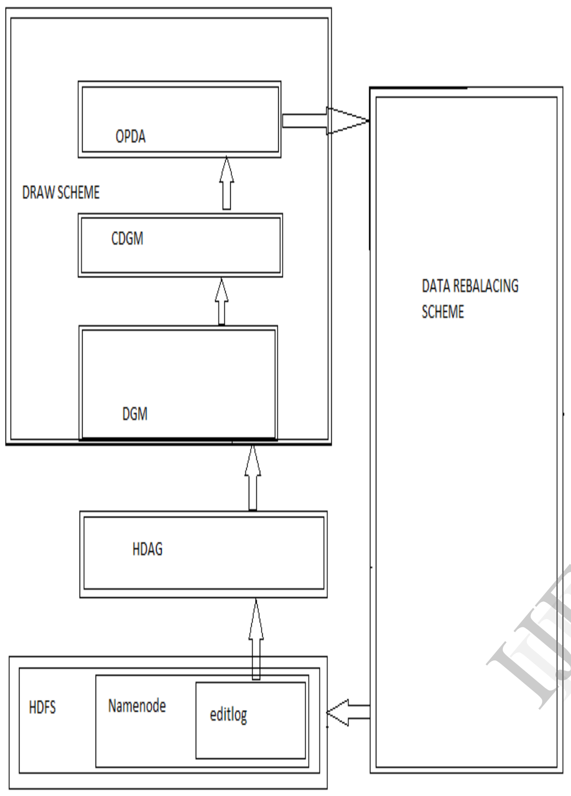


Fig 3(a) System Architecture

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance. HDFS will consist of two components name node and *Data Node*. Name Node is responsible for a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of Data Nodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data Nodes. The Name Node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes. The Data Nodes are responsible for serving read and write requests from the file system's clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the Name Node.

The Name Node uses a transaction log called the Edit Log to persistently record every change that occurs to file system Meta data. So with the help of edit log analysis by *HDAG* and *FSCK* frequently accessed block names & also the information about which block in which node can be extracted these will be stored and compared. According to this replication matrix will be formed.

This replication matrix table will be as input to the cluster data grouping matrix and it will form cluster according to the weight of each block. To calculate these weights bond energy algorithm should be used. Still there is no optimal solution is obtained, so optimal data placement algorithm will be used to distribute those blocks and finally frequently accessed blocks should be replicated by using Data Rebalancing scheme which will increase the performance more than previous one.

4. Algorithms

4.1. Bond Energy Algorithm

1. Set i=1. Arbitrarily select any row from DGM and place it.
2. Place each of the remaining n-i rows in each of the i+1 positions (i.e. above and below the previously placed i rows) and determine the

row bond energy for each placement using the formula

$$\sum_{i-1}^{i+1}\sum_{j-1}^{m} a_{ij}\left(a_{i-1,j}+a_{i+1},j\right)$$

3. Select the row that increases the bond energy the most and place it in the corresponding position.
4. Set $i = i+1$. If $i < n$, go to step 2; otherwise go to step 4.
5. Set j=1. Arbitrarily select any column and place it.
6. Place each of the remaining m-j rows in each of the j+1 position (i.e. to the left and right of the previously placed j columns) and determine the column bond energy for each placement using the formula.

$$\sum_{i=1}^{n}\sum_{j=1}^{j+1} a_{ij}\left(a_{i,j-1}+a_{i,j+1}\right)$$

7. Set $j=j+1$. If $j < m$, go to step 5; otherwise stop.

4.2 Optimal Data Placement Algorithm

Input: The sub-matrix (OSM) as shown in Fig. 2(d): M[n][n] where is the number of data nodes;
Output: A matrix indicating the optimal data Placement: DP**[2][n];**
Steps:

```
1. for each row from M[n] [n]do
        R= index of current row
     Put this value and its corresponding column index
              into a set MinSet.

2. There may be more than one minimum      value.

Min set = (c1, v1) (C2, V2)
If there are only one set (c1, v1)
The data referred by C1 should be placed with the Data
referred by R on the same node;
3. Mark c1 column is invalid (already assigned)
              DP [0] [R] =R
              DP [1] [R] =C1

4. For each column Ci from Min set do

     Calculate sum[i] = sum (M(*)Ci) all the items
     in column Ci

5. Choose the largest value from sum array
        C= index of the chosen sum item
        DP [0] [R] =R
        DP [1] [R] =C
        Mark column c is invalid (already
        assigned);
```

Conclusion:

The default random data placement in a Hadoop cluster does not take into account data grouping semantics. This could cluster many grouped data into a small number of nodes, which limits the data parallelism degree and results in performance bottleneck. In order to solve the problem, a new DRAW scheme is developed. DRAW captures runtime data grouping patterns and distributes the grouped data as evenly as possible. There are three phases in DRAW: learning data grouping information from system logs, clustering the data-grouping matrix, and reorganizing the grouping data. In addition to this we proposed a Data Rebalancing scheme which will replicate frequently accessed blocks so that performance can be increased. We also theoretically prove that the inefficiency of hadoops random data placement algorithm. DRAW& Data rebalancing schemes can significantly improve the throughput of local map task execution and reduce the execution time of map phase. The overall Mapreduce job response time is reduced compare to previously used.

REFERENCES:

[1][Online].Available:http://aws.amazon.com/s3/
[2][Online].Available:http://bowtiebio.sourceforge.net/index.shtml
[3][Online].Available:http://developer.yahoo.com/hadoop/tutorial/module1.html
[4][Online].Available:http://genome.ucsc.edu/
[5][Online].Available:http://hadoop.apache.org/common/docs/r0.18.3/ hdfs_design.html
[6][Online].Available: http://michael.dipperstein.com/bwt/
[7][Online].Available:http://sector.sourceforge.net/benchmark.html
[8][Online].Available:https://issues.apache.org/jira/browse/hadoop-2559
[9][Online].Available:http://t8web.lanl.gov/people/heitmann/arxiv/
[10] A. Amer,D.D. E. Long, and R. C.Burns, "Group-based management of distributed file caches," in *Proc. 22nd Int.Conf. Distrib. Compute. Syst. (ICDCS'02)*,Washington, DC, USA, 2002, p. 525, IEEE Computer Soc.

[11] A. Dumitriu,"X and y (number 5)," in *Proc. ACM SIGGRAPH 2004 Art Gallery SIGGRAPH'04*, New York, NY, USA, 2004, p. 28, ACM.
[12]G.Ganger and M. Frans Kaashoek, "Embedded I nodes and explicit grouping: Exploiting disk bandwidth for small files," in *Proc. 1997 USENIX Technol. Conf.*, 1997, pp. 1–17.
[13] N. Gorla and K. Zhang, "Deriving program physical structures using bond energy algorithm," in *Proc. 6th Asia Pacific Software Eng.Conf. APSEC'99*,Washington, DC, USA, 1999, p. 359, IEEE Computer Soc.
[14] Y. Hahn and B. Lee, "Identification of nine human-specific frame shift mutations by comparative analysis of the human and the chimpanzeegenomesequences"*Bioinformatics*, vol. 21, pp. 186–194, Jan. 2005.
[15]X. Jiong, Y. Shu, R. Xiaojun, D. Zhiyang, T. Yun, J. Majors, A. Manzanares, and q.xiao , "improving Mapreduce performance of through data placement in heterogeneous hadoop clusters," april 2010.
[16] G. H.Kuenning andG. J. Popek, "Automated hoarding for mobile computers," in *Proc. 16th ACM Symp. Operat. Syst. Principles, SOSP'97*, NewYork, 1997, pp. 264–275, ACM.
[17] J. G. Liu, M. Ghanem, V. Curcin, C. Haselwimmer, Y. Guo,G.Morgan, and K. Mish, "Achievements and experiences from a grid-based earthquake analysis and modelling study," in *Proc. 2nd IEEE Int. Conf. e-Science and Grid Computing, E-SCIENCE'06*, Washington, DC, USA, 2006, p. 35-, IEEE Computer Soc.
[18] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems* , 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.
[19] M. Rodriguez-Martinez, J. Seguel, and M. Greer, "Open source cloud computing tools: A case study with a weather application," in *Proc. 2010 IEEE 3rd Int. conf.cloud comput, cloud'10* Washington, DC, USA, 2010, pp. 443–449, IEEE Computer Soc.
[20]M.C.Schatz,"Cloudburst," *Bioinformatics*, vol. 25, pp. 1363–1369, Jun. 2009.
[21] Jun Wang, Qiangju Xiao, Jiangling Yin, and Pengju Shang ," DRAW: A New Data-gRouping-AWare Data Placement Scheme for DataIntensive Applications With Interest Locality," *IEEE* trancactions in magnetics, Vol. 49, NO. 6, June 2013
[22] M.Specht, R.Lebrun, and C. P.E.Zollikofer, "Visualizing shape transformation between chimpanzee and human braincases," *Vis. Comput.*, vol. 23, pp. 743–751, Aug. 2007.
[23] S. Tripathi and R. S. Govindaraju, "Change detection in rainfall and temperature patterns over india," in *Proc. 3rd Int. Workshop on Knowledge Discover. Sens. Data, SensorKDD'09*, NewYork, NY, USA, 2009, pp. 133–141, ACM.
[24] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Gener. Comput. Syst.*, vol. 26, pp. 1200–1214, Oct. 2010.
[25] B. Zhang, N. Zhang, H. Li, F. Liu, and K. Miao, "An efficient cloud computing-based architecture for freight system application in china railway," in *Proc. 1st Int. Conf. Cloud Comput. CloudCom'09*, Berlin, Germany, 2009, pp. 359–368, Springer-Verlag.
[26] S. Sehrish, G. Mackey, J. Wang, and J. Bent, "Mrap: A novel mapreduce- based framework to support HPC analytics applications with access patterns," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput. HPDC'10*, New York, NY, USA, 2010, pp. 107–118, ACM.