International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 3, Issue 4, July 2014, 1-10
© IASET

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

AN OPTIMAL DATA PLACEMENT FRAMEWORK TO INCREASE PERFORMANCE BOF MAPREDUCE FOR DATA-INTENSIVE APPLICATIONS WITH INTEREST LOCALITY

SOLMAZ VAGHRI[1] & MOHAN K. G[2]

[1]M.Tech Student, Department of CSE, Acharya Institute of Technology, Affiliated to VTU, Bangalore, Karnataka, India

[2]Professor, Department of CSE, Acharya Institute of Technology, Affiliated to VTU, Bangalore, Karnataka, India

ABSTRACT

Emerging many numbers of data-intensive applications that needs to access ever-increasing data sets ranging from gigabytes to terabytes or even petabytes, place a demand on employing parallel processing techniques to optimize performance and reduce the decision time. Of late parallel computing frameworks such as MapReduce and it’s open source implementation Apache Hadoop has been used to run large scaledata-intensive applications and conduct analysis, but data locality have not been taken into account in Hadoop and MapReduce and they use random data distribution method for load balancing. Practically in many data-intensive applications data groups often accessed to gather and only subset of a whole data set are frequently used. Ignoring data grouping issue and random data placement noticeably reduce the performance of MapReduce and Hadoop. This paper presents architecture and implementation status of a an optimal data placement framework that dynamically analyzes data accesses from system log files and create optimal data groupings and distribute the data evenly to achieve maximum parallelism per data group and significantly improves the overall performance of MapReduce for data-intensive applications.

KEYWORDS: Data-Intensive, Data Placement, Hadoop, Map Reduce, Parallel Processing

INTRODUCTION

Many Scientific and Engineering applications have become data-intensive, hence accessing large amounts of data which demand on high performance computing approaches for efficiently processing and analysing large scale data sets in less amount of time. Parallel processing frameworks and large scale distributed file systems can be used to facilitate high-performance runs of data-intensive applications. Parallel computing of data-intensive applications, partitions data into several smaller segments that can be process independently in parallel using the same executable application program on an appropriate computing platform and then reassembles the results to create the complete output data [3].

The fundamental challenges for computing in data-intensive applications are managing and processing ever expanding data sets, significantly combining associated data to support practical, timely applications. In this regard several solutions have been employed including the Google’s MapReduce and Hadoop that is open-source implementation of MapReduce [4]. In Hadoop, data stored in HDFS files whichis a distributed file system that provides high throughput access to data and map and reduce functions operate on it. The large files store on HDFS as a series of blocks distributed over a cluster of data nodes; for the purpose of fault tolerance HDFS can be configured to replicate data. HDFS tries to balance load by placing blocks randomly; it does not take into account any data characteristics[5].

In particular, HDFS does not provide any means to locate related data on the same set of nodes [5]. In MapReduce framework two important procedures are Map that performs filtering and sorting and Reduce that performs a summary operation; in this approach jobs process in parallel by splitting into many map tasks running on multiple machines in large scale cluster by assuming the data locally being stored. This can simplify the complexity of running large data processing functions among several nodes in a cluster; then automatically handles the gathering of results and return a single result or set.

The data locality for current Hadoop and MapReduce Implementations is important performance factor; they use random data distribution policy based on disk space availability to balance the load, which is very efficient when in the terms of both computing and disk capacity nodes are identical for heterogeneous environment [6]. We empirically observed that many data-intensive applications that take benefits from MapReduce and Hadoop systems for data processing have *interest data locality*; they use only subsets of big data set or one subset has been accessed more frequently than others in such applications [6]. For example, in the climate modelling and forecasting domain [11], some scientists are only interested in some specific time periods. As another example in the bioinformatics domain, X and Y chromosomes of human that are related to the offspring's gender are often analysed together in generic research rather than all the 24 human chromosomes [12]. In summary, the most affinitive data have high possibility to be processed as a group by specific domain applications. Here we can define the "*data grouping*" to represent the possibility of two or more data blocks to be accessed as a group in Hadoop and MapReduce. In the other hand possibility for two data blocks that accessed together for many times is high, to be accessed as a group in later [7].

Each data group is quantified by a weight which is nothing but the number of the times that the data already have been accessed as a group. In MapReduce program without evenly distributing grouping data, some map tasks might be scheduled on the nodes that can remotely access the needed data, or they are scheduled on the data holding nodes but have to wait in the queue. These map tasks are not match with the locality of data and significantly reduce the MapReduce program performance. Figure 1 shows this scenario; the grouping data are distributed by random distribution strategy, the shaded map tasks have remote data access or queuing delay that are the performance barriers; whereas if these data are evenly distributed, the MapReduce program can avoid these problem [7].

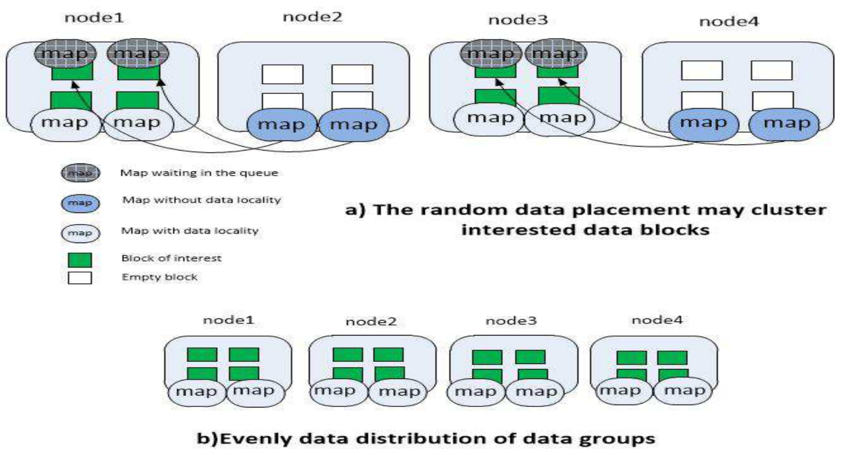

node1
node2
node3
node4
map
Map waiting in the queue
Map without data locality
Map with data locality
Block of interest
Empty block
a) The random data placement may cluster interested data blocks
b)Evenly data distribution of data groups

Figure 1: Simple Case Showing the Efficiency of Data Placement for MapReduce

Evenly data distribution across the nodes by random data placement is affected by several factors including, the number of replica for each data block in each rack in Hadoop cluster; the large number of replica the more map tasks can be run simultaneously and maximum parallelism can be achieved.

Another important factor is number of simultaneous tasks which are equal to number of similar data; in this case very large number leads to achieving maximum parallelism because each node can have one copy of data in the cluster. As last factor we can consider the data grouping access pattern that affects evenly distribution of random data placement strategy.

However in practice, the default replication factor is 3 which means that a block is stored on three separate data nodes; and default number of simultaneous tasks are 2 which is very small [7]. Moreover, the data grouping has not been considered in default Hadoop and MapReduce, results in a nonoptimal data placement strategy for the data-intensive applications having interest data locality. In this study, we develop an Optimal Data Placement Framework (OPDF) to address above mentioned problem. Generally OPDF is designed to dynamically scrutinize system log file and extracts optimal data grouping and re-balance data to achieve maximum parallelism per group; the framework performs data reorganization before MapReduce program and boosts its performance by rebalancing the load across the nodes.

MOTIVATION

Data is an important parameter to take any business decisions and to carry out scientific research. Since Massive amounts of data are generating every day in variety of domains; this lead to ever increasing use of parallel computing techniques, thus optimal data placement is critical in order to meet their performance requirements. Of late data parallel processing techniques like Hadoop and MapReduce that employ random data distribution policy for load balancing have been used, for data analysis and gain insights from it.In this approach by developing a framework which uses the local interest by scanning the system logs, we canspeed the process of analysis.

SYSTEM ARCHITECTURE

In OPDF, the technique of identifying frequently accessed groups based on the weights of data groups involves the following steps [7]:

- History data access graph to scan system event log and obtains the data grouping information.
- A data grouping matrix to generate the optimized data groupings and assign the grouping weights to data groups.
- An optimal data placement algorithm to create the optimal data placement.

History Data Access Graph

History Data Access Graph (HDAG) can be obtained by the history of data accesses and describes which blocks of data accessed among the files. The system logs records every system operation and shows which files have been accessed. By monitoring the files access pattern, every two frequently accessed files can be categorized in one group. Although we need a simple traversal of system log file to learn data grouping information, in practice we will face long traversal latency since the log files could be huge. To overcome this problem, we define checkpoints to indicate how far the HDAG needs to traverse back in the logs.

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

(a)

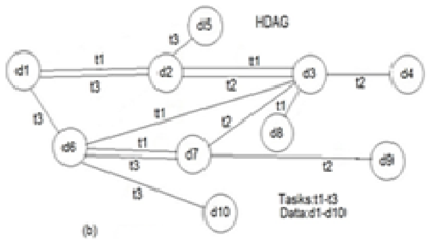


Figure 2: Example Showing the HDAG

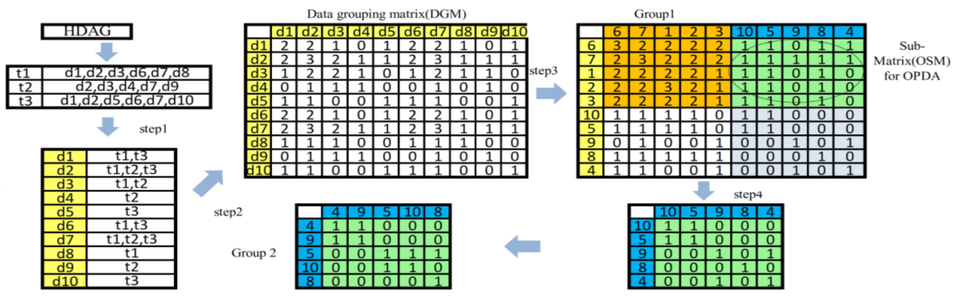
Figure 2 shows an example of HDAG: By considering three MapReduce tasks as shown in above table t1 accesses d1, d2, d3, d6, d7, d8, here d is data block. t2 accesses d2, d3, d4, d7, d9 and t3 accesses d1, d2, d5, d6, d7 d8. [7] The accessing information initially generated from the log files is shown as Figure 2(a) Therefor we can easily translate this information into the HDAG shown as Figure 2(b) that will be used to generate DGM in the next step.

Data Grouping Matrix

The Data grouping matrix (DGM) shows the relation between every two data blocks and can be generated based on HDAG. Based on the same example as shown in Figure 2, we can build the DGM as shown in Figure 3 (step1 and step2). DGM is N by N matrix where N is number of data blocks in HDAG and each element of DGM represents grouping weight between two data block. Every DGM$_{i,j}$ can be calculated by counting the tasks in common between task sets for blocks i and j. The elements in the diagonal show the number of jobs that have used the data block. Each data block belonging to group A may belong to group B at the same time therefore the grouping weight in the DGM shows “how likely” one data should be grouped with another data[7].

After obtaining the DGM in Figure 3, we need to use matrix clustering techniques to group the most related data in step3. Specifically, Bond Energy Algorithm (BEA) [9] is used to create clustered data grouping matrix (CDGM) from DGM. We can apply BEA for the purpose of grouping various types of machinery by their functions; it has been widely used in distributed database systems for the vertical partition of large tables and matrix clustering work.

According to the frequencies of access of data blocks, we adapt the BEA to decide the components placement. We use DGM which is N by N symmetric matrix, whose rows and columns identify the N data block and each array element is the frequency access between the two blocks at the corresponding row and column.



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

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

Group1

	6	7	1	2	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
1	2	2	2	2	1	1	1	0	1	0
2	2	2	3	2	1	1	1	0	1	0
3	2	2	2	2	1	1	1	0	1	0
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

	10	5	9	8	4
10	1	1	0	0	0
5	1	1	0	0	0
9	0	0	1	0	1
8	0	0	0	1	0
4	0	0	1	0	1

Group 2

	4	9	5	10	8
4	1	1	0	0	0
9	1	1	0	0	0
5	0	0	1	1	1
10	0	0	1	1	0
8	0	0	0	0	1

Figure 3: Example Showing the Grouping Matrix and the Overall Flow to Cluster Data Based on their Grouping Weight

Since we have a symmetric matrix, we need only to find a row (or column) permutation that creates the strongest `bond energy' by driving the larger array elements together. This is achieved by calculating the measure of effectiveness (ME) for each permutation by [9]:

$$ME=\sum_i^N \sum_j^N Fi,j(Fi,j-1+Fi,j+1)$$

Where $F_{i,j}$ is the frequency calling between blocks i and j. The maximum ME for permutation represents a desirable component placement. Since our matrix is symmetric, the algorithm can operate on either rows or columns and generate the same result as described below. Given N data blocks, assuming to operate on rows:

- Create an N by N symmetric matrix *InitMatrix= [$F_{i,\ j}$]* such that $F_{i,\ j}= F_{j,\ i}$. is the frequency calling between components i and j (i ≠j), and $F_{i,i}$is an integer larger than the maximal value of the matrix elements.
- Set i = 1; Select a row arbitrarily (say, the first row) from Init Matrix to be placed in a new matrix.
- Place individually each of the remaining N –i rows in each of i +1 possible positions, and compute each row's contribution to the ME. Place the row in the position that gives the largest incremental ME. Set i=i+1 and repeat this step if i< N.
- The rows of the resulting new matrix (here CDGM) give the relative positions of the program components; the BEA algorithm clusters the most affinitive data together indicating which data should be evenly distributed [9].

In our example, after group 1 is generated we can repeat the steps in step 4 and step 5 to generate the group 2. In this case, group 1 and group 2 represent highly related data sets. Assuming there are 5 Data Nodes in the cluster, the CDGM in Figure 3 indicates data {6, 7, 2, 1, 3} (group 1) and {4, 9, 5, 10, 8} (group 2) should be evenly distributed when placed on the 5 nodes. Note that 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, step 4 and step 5 in Figure 3 are not necessary for our case, but when the number of remaining data (after recognizing the first group) is larger than the number of nodes [7].

Optimal Data Placement Algorithm

We cannot achieve the optimal data placement and maximum level of parallelism, knowing the data groups solely. Here we use OPDA [7] in order to optimally placing data among the nodes in Cluster.

```
Algorithm: OPDA M[n][n]
//Input – Sub-matrix OSM, which include the groups and their weights
//Output – Matrix indicating optimal group placement.
For each row from M[n][n] do
        R = index of the current row;
Find the minimum value V in this row;
Put this value and its corresponding column index C into a set Min Set; MinSet = C1, V1, C2, V2;
If there is only one tuple (C1, V1) in MinSet then
        DP[0][R]=R;
        DP[1][R]=C1;
```

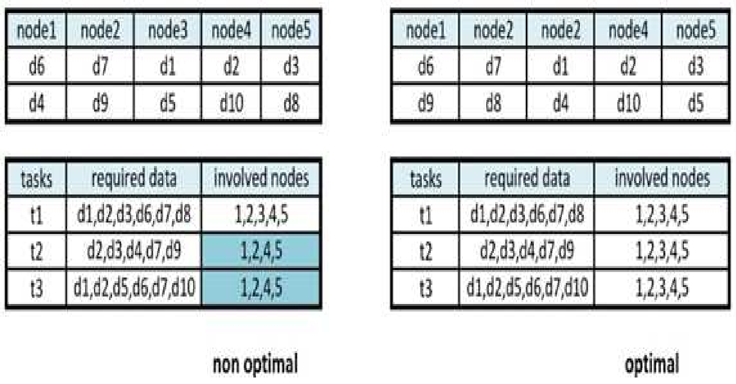
```
Mark C1 as invalid, Continue;
end if
**for** each column Ci from MinSet **do**

        Calculate Sum[i] = sum (M[*][Ci])
        Items in Ci column;
**end for**
Choose the largest value from array;
        C= index of the chosen Sum item;
        DP [0][R]=R;
        DP [1][R]=C;
Mark column is invalid (already assigned);
**end for**
```

Given the same example from Figure 3, random placement of each group, as shown in Figure 4 (1); task 2 and task 3 can only run on 4 nodes rather than 5 which is not optimal[7]. By considering the horizontal relationships among the data in DGM, this cannot be optimal. So it is necessary to make sure the blocks on the same node have minimal chance to be in the same group (vertical relationships).

In order to obtain this information, we make use of an ODPA algorithm to complete our OPDF design. ODPA is based on submatrix for ODPA (OSM) from CDGM. OSM indicates the dependencies among the data already placed and the ones being placed. For example, the OSM in Figure 3 denotes the vertical relations between two different groups (group 1:6, 7, 2, 1, 3 and group 2:4, 9, 5, 10, 8). Take the OSM from Figure 3 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}, 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 sum 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.



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

tasks	required 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	1,2,4,5

non optimal

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

tasks	required 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

Figure 4: Shows without OPDA, Non-Optimal Data the Layout Generated in CDGM

After doing the same processes to the rows with index 1, 2, 3, we have a DP= {(6,9) (7,8) (1,4) (3,5) (2,10)}, indicating the data should be placed as shown in Figure 4 (2). Clearly, all the tasks can achieve the optimal parallelism in all 5 data nodes when running on the optimal data layout. With the help of ODPA, OPDF can achieve the two goals: maximize the parallel distribution of the grouping data, and balance the overall storage loads [7].

The OPDF is designed to increase MapReduce performance for the applications showing interest locality. In case of applications that do not have interest locality, all the data on the cluster belongs to the same group. Therefore the data grouping matrix contains the same grouping weight for each pair of data (except for the diagonal numbers); the BEA algorithm will not cluster the matrix, all the data blocks will stay on the nodes and distributed as the default random data distribution[7][8]. Because all the data are equally popular, theoretically random data distribution can evenly balance them onto the nodes. In this case, OPDF has the same performance as random data distribution strategy[7].

IMPLEMENTATION STATUS

In this section we present comparison of the impact of random data placement on the MapReduce programs and proposed framework's reorganized data. We empirically developed a program performing data reorganization according to the ODPA; In order to verify the system's feasibility, we constructed test bed nodes by employing an open source cloud; using CPU > 2 GHZ, HDD >300GB and RAM >2GB for configuration, as well as Apache Tomcat 7.0 as server and MySql 5.5 database.

The data partitioned into less than 64MB blocks and upload to the framework; the system exploited log file and conducted analysis on data. Following figure shows the traces of two runs on OPDF's reorganized data and MapReduce randomly placed data, respectively.

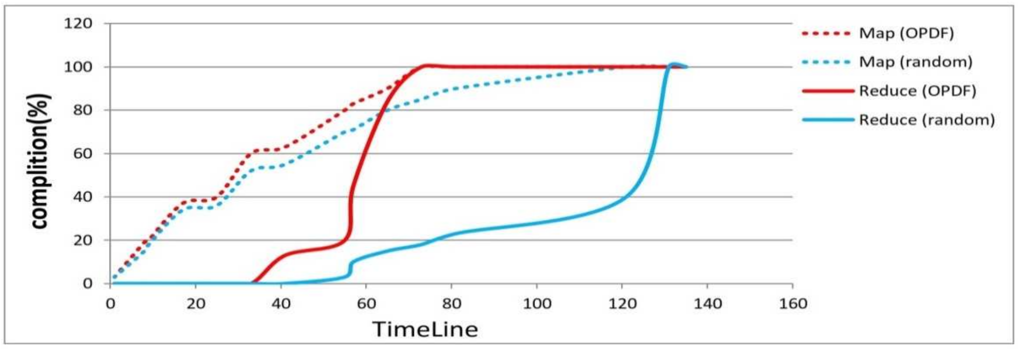


Figure 5: The MapReduce Program Statistic for OPDF and Random Placement Method

The number of reducers is set large to avoid performance bottleneck. The Map when using OPDF has finished 42.2% earlier than random method and the overall performance of system improved by 46.6% in comparison with random data placement method.

CONCLUSIONS AND FUTURE WORK

The default random data placement in a MapReduce/Hadoop framework does not take into account data grouping semantics. This could create clustering of many grouped data into a small number of nodes, which leads to limited level of parallelism and performance bottleneck. In order to solve the problem, an optimal data placement framework is developed. OPDF captures runtime data grouping patterns and distributes the grouped data as evenly as possible. There are three phases in OPDF: learning data grouping information from system logs, clustering the data-grouping matrix, and reorganizing the grouping data.

The proposed placement algorithm in the OPDF architecture uses the idea of grouping weights for analysis. The algorithm follows "highest weight first" strategy as rule of thumb for data placement. However for operating system scheduling, algorithms that following "lowest weight first", "first come first serve" and "shortest analyzed group first" can be used to decrease the number of large data chunks sent for analysis ; thereby increasing the performance of the analysis and using the system resources to the fullest and generate business specific data for decision making at faster rates.

REFERENCES

1. [Online]. Available:http://developer.yahoo.com/hadoop/tutorial/module1.html
2. A Amer, D. D. E. Long, and R. C. Burns, "Group-based management of distributed file caches," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst. (ICDCS'02)*, Washington, DC, USA, 2002, p. 525, IEEE Computer Soc.
3. L.S. Nyland, J.F. Prins, A. Goldberg, and P.H., Mills, "A Design Methodology for Data-Parallel Applications", IEEE Transactions on Software Engineering, Vol. 26, No. 4, 2000, pp. 293-314.
4. [Online]. Available: http://en.wikipedia.org/wiki/Data-intensive_computing
5. M. Y. Eltabakh_, Y. Tian_, F. Ozcan, R. Gemulla, A. Krettek, J. McPherson," Co Hadoop: Flexible Data Placement and Its Exploitation in Hadoop", proceeding of VLDB Endowment(PVLDB), Vol 4, pp 575-585, 2011
6. J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin," Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters" in Proc. 19th Int'l Heterogeneity in Computing Workshop, Atlanta, Georgia, April 2010.
7. J. Wang, Q. Xiao, J. Yin, and P. Shang," DRAW: A New Data-g Rouping-A Ware Data Placement Scheme for Data-intensive Applications With Interest Locality", IEEE transaction on magnetics, vol. 49, no 6, june 2013.
8. K. Zhang, N. Gorla," Locality metrics and program physical structures" Elsevier Science, May 2000, pp: 159-166, vol 54.
9. 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.

10. D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Gener. Comput. Syst.*, vol. 26, pp. 12001214, Oct 2010.

11. 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.

12. Y. Hahn and B. Lee, "Identification of nine human-specific frame shift mutations by comparative analysis of the human and the chimpanzee genome sequences," *Bioinformatics*, vol. 21, pp. 186–194, Jan. 2005.