DRAW: A New Data-gRouping-AWare Data Placement Scheme for Data Intensive Applications With Interest Locality

Jun Wang, Qiangju Xiao, Jiangling Yin, and Pengju Shang

Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32826 USA

Recent years have seen an increasing number of scientists employ data parallel computing frameworks such as MapReduce and Hadoop to run data intensive applications and conduct analysis. In these co-located compute and storage frameworks, a wise data placement scheme can significantly improve the performance. Existing data parallel frameworks, e.g., Hadoop, or Hadoop-based clouds, distribute the data using a random placement method for simplicity and load balance. However, we observe that many data intensive applications exhibit *interest locality* which only sweep part of a big data set. The data often accessed together result from their *grouping* semantics. Without taking data grouping into consideration, the random placement does not perform well and is way below the efficiency of optimal data distribution. In this paper, we develop a new Data-gRouping-AWare (DRAW) data placement scheme to address the above-mentioned problem. DRAW dynamically scrutinizes data access from system log files. It extracts optimal data groupings and re-organizes data layouts to achieve the maximum parallelism per group subjective to load balance. By experimenting two real-world MapReduce applications with different data placement schemes on a 40-node test bed, we conclude that DRAW increases the total number of local map tasks executed up to 59.8%, reduces the completion latency of the map phase up to 41.7%, and improves the overall performance by 36.4%, in comparison with Hadoop's default random placement.

Index Terms—**Data intensive, data layout, Hadoop, MapReduce.**

I. Introduction

The emerging myriad data intensive applications place a demand on high-performance computing resources with massive storage. Academic and industrial pioneers have been developing big data parallel computing frameworks and large-scale distributed file systems to facilitate the 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], [22]. 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 the future [10].

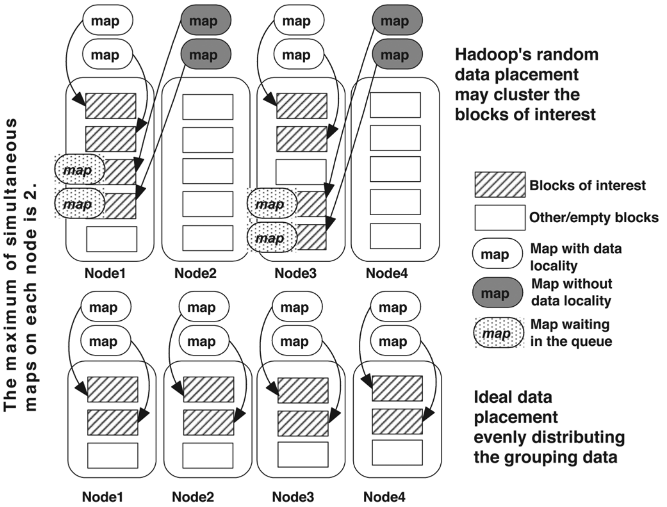

The maximum of simultaneous maps on each node is 2.
map
Node1
Node2
Node3
Node4
Hadoop's random data placement may cluster the blocks of interest
Blocks of interest
Other/empty blocks
Map with data locality
Map without data locality
Map waiting in the queue
Ideal data placement evenly distributing the grouping data

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

By using Hadoop's default random data placement strategy, the overall data distribution may be balanced,[1] but there is no guarantee that the data accessed as a group is evenly distributed. More specifically, a MapReduce job is split into many map tasks to process in parallel. Map tasks intend to be allocated to the nodes with the needed data locally being stored to achieve "compute-storage co-locality". Without evenly distributed grouping data, some map tasks are either scheduled on other nodes which remotely access the needed data, or they are scheduled on these data holding nodes but have to wait in the queue. These map tasks violate the data locality and severely drag down the MapReduce program performance [3]. Besides, dynamic data grouping is effective for exploiting the predictability of data access patterns and improving the performance of distributed file systems [10], [12], [16]. We show an example in Fig. 1: if the grouping data are

Manuscript received November 26, 2012; revised February 28, 2013; accepted March 01, 2013. Date of current version May 30, 2013. Corresponding author: J. Wang (e-mail: jun.wang@ucf.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2013.2251613

[1] If the initial data distribution is not balanced, Hadoop users can start a balancer (an utility in Hadoop), to rebalance the data among the nodes.

distributed by Hadoop's random strategy, the shaded map tasks with either remote data access or queueing delay are the performance barriers; whereas if these data are evenly distributed, the MapReduce program can avoid these barriers.

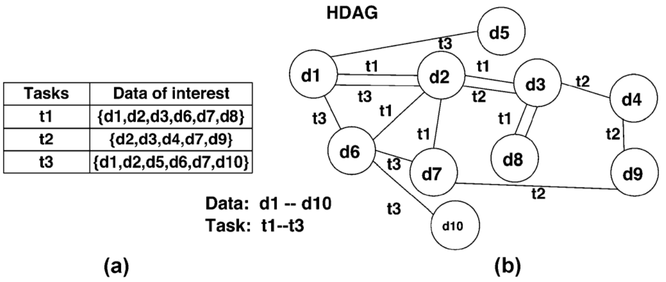
Now we briefly analyze the possibility for random data distribution to evenly distribute the data from the same group. Our observation shows this possibility is affected by three factors: 1) the number of replica for each data block in each rack (NR); 2) the maximum number of simultaneous map tasks on each node (NS); and 3) the data grouping access patterns. It is easy to conclude that the larger NR or NS is, the more likely that Hadoop's default random solution will achieve the optimal distribution: **a**) assume NR is extremely large, e.g., the number of replica for each data is same as the number of nodes in the cluster, because the random data distribution will not place the same data onto the same node, each node will hold one copy of the data therefore the maximized parallelism can be achieved; **b**) assume NS is extremely large, e.g., NS equals to the number of affinitive data, even if the random solution clusters all these data onto the same node, all the map tasks can run simultaneously hence the performance will not be degraded. In above two hypothetical cases, random data distribution is the ideal affinitive-data placement. However in practice, NR and NS are very small (3 replica for each data and $NS = 2$ by default) [3]: in 3-replica Hadoops, there are at most 2 replica ($NR = 2$) in the same rack, the other one is placed in another rack for rack-level fault tolerance; moreover, some single-replica Hadoops ($NR = 1$) are used for performance purpose [7], or in the Cloud platforms that provide sufficient data redundancy for reliability, e.g., Amazon's EC2 and S3 [1], [25]; NS is limited by the hardware capacity, currently $NS \leq 2$ in most of the clusters [5]. Moreover, the data grouping is not considered in default Hadoop, which results in a nonoptimal data placement strategy for the data-intensive applications having interest locality.

Therefore, we develop a new Data-gRouping-AWare data placement scheme (DRAW) that takes into account the data grouping effects to significantly improve the performance for data-intensive applications with interest locality. Without loss of generality, DRAW is designed and implemented as a Hadoop-version prototype. By experimenting 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, DRAW reduces the completion time of map phase by up to 41.7%, and the MapReduce task execution time by up to 36.4%.[2]

The rest of this paper is organized as follow: Section II describes the design of DRAW. Section III present experimental methodology and our results and analysis, respectively. Section IV presents related works. Finally, Section V concludes the paper.

II. Data-gRouping-AWare Data Placement

In this section, we design DRAW at rack-level, which optimizes the grouping-data distribution inside a rack. There are three parts in our design: a data access history graph (HDAG) to exploit system log files learning the data grouping information; a data grouping matrix (DGM) to quantify the grouping weights among the data and generate the optimized data groupings; an optimal data placement algorithm (ODPA) to form the optimal data placement.



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 the HDAG.

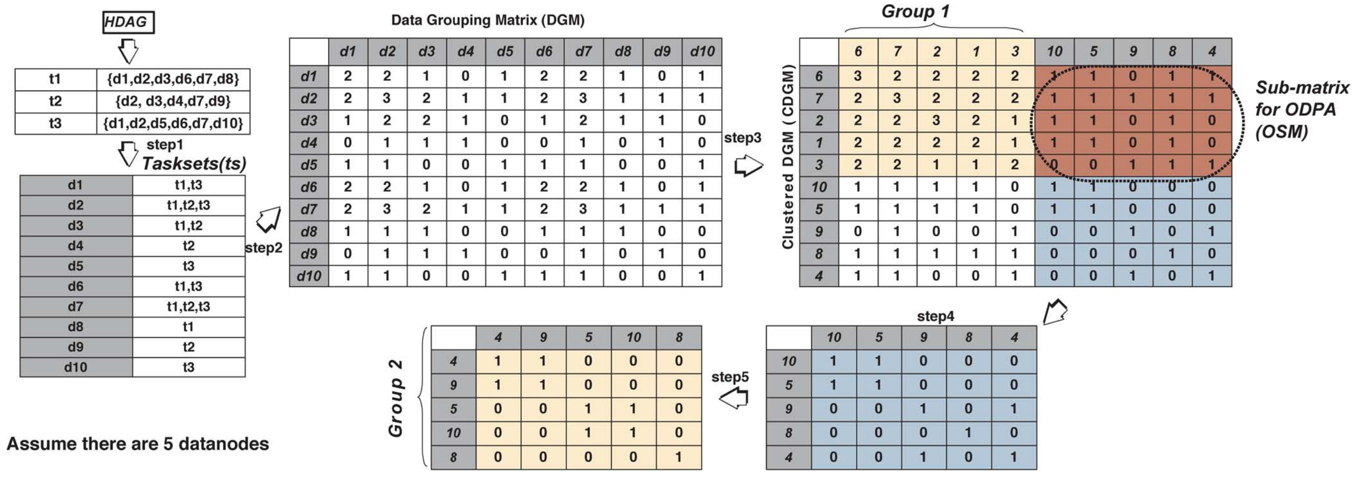
A. History Data Access Graph (HDAG)

HDAG is a graph describing the access patterns among the files, which can be learned from the history of data accesses. In each Hadoop cluster rack, the *NameNode* maintains system logs recording every system operation, including the files which have been accessed. A naive solution can be: monitor the files which are being accessed; every two continuously accessed files will be categorized in the same group. This solution is simple for implementation because it only needs a traversal of the *NameNode* log files. However in practical situations there are two problems: first, the log files could be huge which may result in unacceptable traversal latency; second, the continuously accessed files are not necessarily related, e.g., the last file accessed by task x and the first file accessed by task $x+1$. Therefore, we need to define checkpoint to indicate how far the HDAG will traverse back in the *NameNode* logs; and we also need to exploit the mappings between tasks and files to accurately learn the file access patterns. Note that in Hadoop clusters, files are split into blocks which is the basic data distribution unit; hence we need to translate the grouping information at file level into 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 MapReduce tasks, $t1$ accesses $d1, d2, d3, d6, d7, d8$, here d is block; $t2$ accesses $d2, d3, d4, d7, d9$; and $t3$ accesses $d1, d2, d5, d6, d7, d10$. The accessing information initially generated from the log files is shown as Fig. 2(a). Thereafter 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.

B. DGM

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 build the DGM as shown in Fig. 3 (step1 and step2), where each element $\mathrm{DGM}_{i,j} = \mathrm{grouping}_{i,j}$ is the grouping weight between data i and j. Every $\mathrm{DGM}_{i,j}$ can be calculated by counting the tasks in common between task sets of ts_i and ts_j. The elements in the diagonal of the DGM show the number of jobs that have used this data. In DRAW, DGM is a n by n matrix, where n is the number of existing blocks. As we stated before, one data belonging to group A may belong

[2] These numbers can be affected by the number of launched reduce tasks, the required data size, etc.


HDAG
step1
Taskets(ts)
step2
Data Grouping Matrix (DGM)
step3
Group 1
Clustered DGM (CDGM)
Sub-matrix for ODPA (OSM)
step4
step5
Group 2
Assume there are 5 datanodes

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

Tasksets(ts)

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

Clustered DGM (CDGM)

	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

step4

	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	0
10	0	0	1	1	0
8	0	0	0	0	1

Fig. 3. Example showing the grouping matrix and the overall flow to cluster data based on their grouping weights.

to group B at the same time; the grouping weight in the DGM denotes "how likely" one data should be grouped with another data.

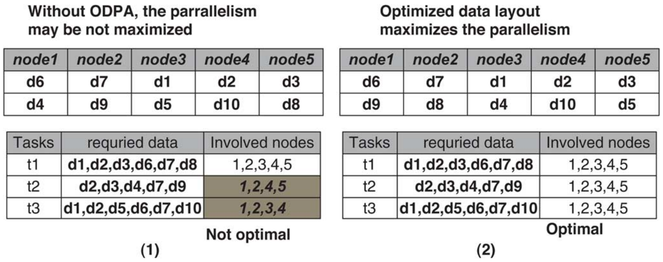
After knowing the DGM in Fig. 3, we use a matrix clustering algorithm to group the highly related data in step3. Specifically, Bond Energy Algorithm (BEA) is used to transform the DGM to the clustered data grouping matrix (CDGM). Since a weighted matrix clustering problem is N-P hard, the time complexity to obtain the optimized solution is $O(n^n)$, where n is the dimension. 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 algorithm clusters the highly associated data together indicating which data should be evenly distributed. After group 1 is generated, repeat the steps in step 4 and step 5 to generate the group 2. In this case, group 1 and group 2 represent most related data sets. Assuming there are 5 *DataNodes* in the Hadoop cluster, the CDGM in Fig. 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 Fig. 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, more clustering steps are needed.

C. ODPA

Knowing the data groups alone is not enough to achieve the optimal data placement. Given the same example from Fig. 3, random placing of each group, as shown in Fig. 4 (1), task 2 and task 3 can only run on 4 nodes rather than 5, which is not optimal.

Algorithm 1 ODPA algorithm

```
Input: The sub-matrix (OSM) as shown in Fig. 3: $M[n][n]$;
where $n$ is the number of data nodes;
Output: A matrix indicating the optimal data placement:
$DP[2][n]$;
Steps:
for each row from $M[n][n]$ do
    $R = $ the index of current row;
    Find the minimum value $V$ in this row;
    Put this value and its corresponding column index $C$ into
    a set MinSet;
    $MinSet = C1, V1, C2, V2,$; // there may be more than
    one minimum value
    if there is only one tuple $(C1, V1)$ in $MinSet$ then
        //The data referred by C1 should be placed with the
        data referred by R on the same node;
        $DP[0][R] = R$;
        $DP[1][R] = C1$;
        Mark column $C1$ is invalid (already assigned);
        Continue;
    end if
    for each column $C_i$ from $MinSet$ do
        Calculate $Sum[i] = sum(M[\star][C_i])$;// all the
        items in $C_i$ column
    end for
    Choose the largest value from $Sum$ array;
    $C = $ the index of the chosen Sum item;
    $DP[0][R] = R$;
    $DP[1][R] = C$;
    Mark column $C$ is invalid (already assigned);
end for
```



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

(1)

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

(2)

Fig. 4. Without ODPA, the layout generated from CDGM may be still non-optimal.

This is because the above data grouping only considers the horizontal relationships among the data in DGM, and so it is also 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 propose an algorithm named ODPA to complete our DRAW design, as described in Algorithm 1. 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 Fig. 3 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. 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. After doing the same processes to the rows with index 1, 2, 3, we have a $DP = \{\{6, 9\}, \{7, 8\}, \{1, 4\}, \{2, 5\}, \{3, 10\}\}$, indicating the data should be placed as shown in Fig. 4 (2). Clearly, all the tasks can achieve the optimal parallelism (5) when running on the optimal data layout. With the help of ODPA, DRAW can achieve the two goals: maximize the parallel distribution of the grouping data, and balance the overall storage loads.

D. Exceptions

The cases without interest locality: DRAW is designed for the applications showing interest locality. However there are some real world applications do not have interest locality. In this case, 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. Because all the data are equally popular, theoretically random data distribution can evenly balance them onto the nodes. In this case, DRAW has the same performance as Hadoops random data distribution strategy.

The cases with special interest locality: The purpose of DRAW is to optimize the performance for the common applications which follow or not totally deviated from the previous interest locality. However in practice, some applications may have unpredicted access patterns that DRAW did not studied yet. These uncommon queries may suffer from bad performance because DRAW cannot guarantee these accessing data are well distributed. But this pattern will be considered into DRAWs future data organization in case it happened more times.

E. Multireplica Per Rack

In previous analysis, we assume that there is only one copy of each data existing in each rack. This assumption is derived from the practical Hadoop configurations, e.g., Hadoop with single-replica for each data [7], [25], Hadoop with three replicas for each data but put into three different racks [8], etc. However, there are some Hadoop clusters that keep two or even three copies of the same data in the same rack [3] to provide better read performance. As we stated in Section I, the more replicas for each data in the same rack, the more optimal data distribution the random strategy can achieve (given that any two replica cannot stay in the same node). In order to prove our DRAW is still necessary for multiple replica Hadoops, we launch intensive experiments as sensitivity study in Section III-D.

III. Experimental Results and Analysis

In this section, we present four sets of results: the unbalanced data distribution caused by Hadoop's default random data placement; comparison of the traces of the MapReduce programs on the randomly placed data, and the DRAW's reorganized data; and the sensitivity study used to measure the impact of the NR (number of replica for each data block in Hadoop) on DRAW. In the experiments, we developed a program performing data reorganization according to the ODPA output with a preconfigured frequency or we could launch the program manually if necessary.

A. Test Bed

Our test bed consists of 40 heterogenous nodes on a single rack with Hadoop 0.20.1 installed. The cluster and node configurations are shown in the Table I. One node is configured as the *NameNode* and *JobTracker*, whereas the other 39 nodes are *DataNodes* and *TaskTrackers*.

B. Data Distribution

Intuitively, the data distribution may be related to the way the data is uploaded. There are two ways for the users to upload data: bulkily upload all the data at once; or upload the data based on their categories, e.g., species or particles in our cases. The second way considers the human-readable data grouping information (in our case, data belonging to the same species or particles are assumed to be highly related) rather than the blindly uploading as in the first method. We upload the data to our test bed by using these two data uploading methods, 20 times for each. The overall data distributions are similar in these runs.

First, after bulk uploading the genome data of six species (a subset of our 40 GB genome data), the data distribution (from

TABLE I
CASS CLUSTER CONFIGURATION

14 Compute/Data Nodes and 1 Head Node	
Make& 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
25 Compute/Data Nodes	
Make& Model	Sun V20z
CPU	2x AMD Opteron 242 @ 1.6 GHz
RAM	2GB - registered DDR1/333 SDRAM
Internal HD	1x 146GB Ultra320 SCSI HD
Network Connection	1x 10/100/1000 Ethernet connection
Operating System	Rocks 5.0 (Cent OS 5.1), Kernel:2.6.18-53.1.14.e15
Cluster Network	
Switch Make & Model	Nortel BayStack 5510-48T Gigabit Switch

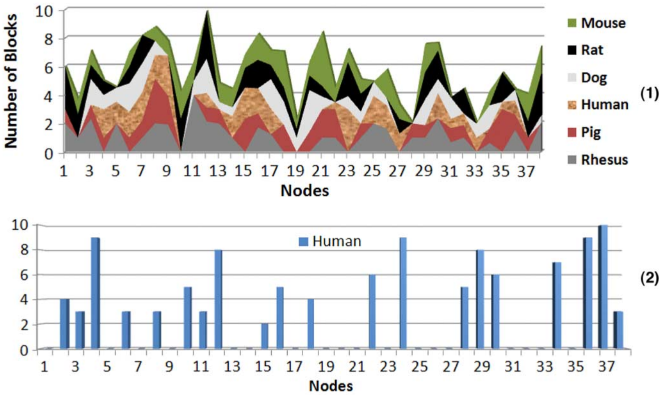


Fig. 5. Data layout after bulk uploading six species' genome data, and the human's genome data layout.

a randomly picked run) is shown in Fig. 5 (1). Given a research group only interested in human [14], [22], the requiring data is clustered as shown in Fig. 5 (2). The human data is distributed on half (51.3%) of the cluster, meaning the parallelism for the future MapReduce job is not optimal.

When using the category-based uploading method, we surprisingly find that the overall data distribution is similar to what is shown in Fig. 5. To highlight the unbalanced distribution of the related data, we quantify the degree of unbalance with $1-(\# \, of \, nodes \, having \, the \, data/\# \, of \, nodes)$. With 20 runs using the species-based data uploading method, on average, the data of a specific species is distributed over only 53.2% nodes of the cluster. The conclusion shows that even when the data is uploaded based on the initial data grouping information, the Hadoop's random data placement is not able to achieve the maximal parallelism for the associate data.

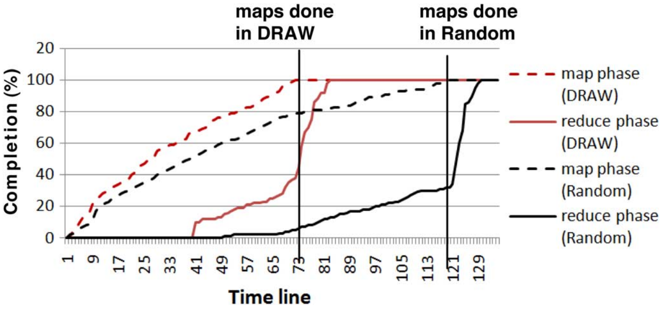


Fig. 6. Running of Genome indexing MapReduce program on human genome data.

TABLE II
COMPARISON OF TWO RUNS OF GENOME INDEXING APPLICATION

	Total maps	Local maps	Ratio
On DRAW	399	302	76.1%
On Random	399	189	47.1%

C. Performance Improvement of MapReduce Programs

1) Genome Indexing: Based on the DRAW [2] reorganized 40 GB genome data downloaded from [4], we run the Bowtie indexing [6] MapReduce program to index the human's chromosomes. Fig. 6 shows the traces of two runs on DRAW's reorganized data and Hadoop's randomly placed data, respectively. The number of reducers is set as large as possible so that the reduce phase will not be the performance bottleneck. In our case, we use 39 reducers. The map phase running on DRAW's data is finished 41.7% earlier than the one running on randomly placed data, and the job's overall execution time is also improved by 36.4% when using DRAW's data. The comparison is shown in Table II. The MapReduce job running on the DRAW's reorganized data has 76.1% maps which benefit from having data locality, compared with 47.1% from the randomly placed data; the

TABLE III
COMPARISON OF EXPERIMENTAL NHD (% OF NODES HOLDING DATA) AND DRAW'S IDEAL NHD

	NR=1			NR=2			NR=3		
	Blks	Experimental NHD	DRAW NHD	Blks	E NHD	D NHD	Blks	E NHD	D NHD
Stickleback	44	44.7%	100%	82	63.2%	100%	122	81.6%	100%
Opossum	48	47.4%	100%	100	73.7%	100%	150	86.8%	100%
Chicken	61	73.7%	100%	122	97.4%	100%	174	89.5%	100%
C.briggsae	13	26.3%	***34.2%***	23	42.1%	***60.5%***	34	68.4%	***89.5%***

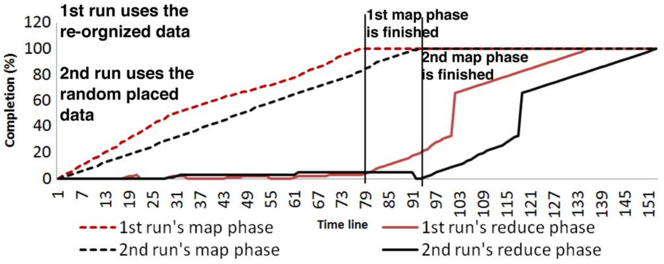


Fig. 7. Running of Mass Analyzer on astrophysics data; the size of interested data for each run is relative small (8 blocks on average).

number of local map tasks is increased by $(320 - 189)/189 = 59.8\%$.

Note that there are still 23.9% maps which are working without having data locality even after the DRAW's data reorganization. There are two reasons: first, the data grouping information the BEA algorithm used is generated from all previous MapReduce programs rather any specific one, and the ODPA follows ***High-Weight-First-Placed*** strategy, which means the data with higher (accumulative) grouping weights will be granted higher priority to be evenly distributed. In other words, the distribution of the nonhottest data is only optimized but may be not 100% perfect for the corresponding MapReduce programs. Second, the matrix clustering is an NP-hard problem, hence the clustered grouping matrix generated from BEA algorithm, whose time complexity is $O(n^2)$ rather than $O(n^n)$, is a pseudo-optimal solution. Adoption of BEA algorithm is a reasonable tradeoff between efficiency and accuracy. However, since the hottest data will be granted the highest priority to be clustered, the applications interested in these data can achieve the ideal parallelism.

2) Mass Analyzer on Astrophysics Data: In this section, we do experiments on smaller data sets: each particle's data is exactly 512 M, which will be split into only 8 blocks.

Our Mass Analyzer on the astrophysics data tries to calculate the average mass of each area. The results are shown in Fig. 7. DRAW reduces the map phase by 18.2%, and the overall performance of the MapReduce program is improved by only 11.2%. It is obvious that the impact of DRAW is linearly related to the size of the required data by the MapReduce program. The less data is being accessed, the more close that random data placement can achieve maximized parallelism. For example, given 40 nodes in the cluster and 2 maximum simultaneous map tasks on each node, the 8 blocks of each astrophysics data file are more likely to be balanced when compared to the 48 blocks of an mammal's genome data. Hence the conclusion is DRAW works better for the MapReduce programs accessing large-scale data (larger than 3 GB for our hardware configuration).

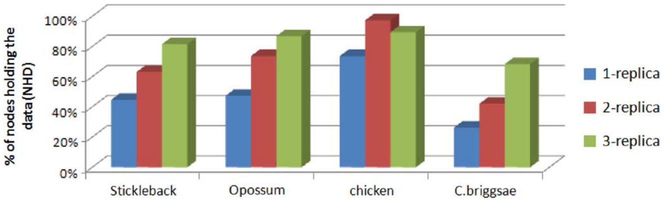


Fig. 8. Data distributions (NHD) of four species, on 1-replica, 2-replica, and 3-replica Hadoop.

D. Sensitivity Study: Number of Replica (NR)

The NR for each data block in Hadoop cluster is configurable. For data distribution, the more replica that exist for each block, the higher possibility that the grouping data can be evenly distributed. Hence, the efficiency of DRAW on the MapReduce programs is inverse proportional to NR in the Hadoop.

In order to quantify the impact of NR on our design, we bulkily upload the 40 G genome data to our test bed configured with $NR = 1$, $NR = 2$ and $NR = 3$, respectively. Fig. 8 shows the data distributions for four species: Stickleback, Opossum, Chicken from vertebrates, and C.briggsae from nematodes. The "% of nodes holding the data (NHD)" is directly related to the parallelism that the program accessing corresponding species can use. The results prove that, in most cases,[3] NR is linearly related to the parallelism of data distribution; which means a higher degree of replica in Hadoop can mitigate the problem of unbalanced grouping-data distribution. For example, the Stickleback data is only distributed on 44.7% of the nodes in 1-replica Hadoop; when using 3-replica Hadoop, 81.5% of the nodes can provide Stickleback data.

Now we study the efficiency of DRAW for multiple replica Hadoop systems. We still use the above data. Table III shows the comparison of the experimental NHD and DRAW's ideal NHD. The NHD difference indicates the possible improvement DRAW can achieve. Note that for the three vertebrates, the number of blocks for each species is larger than the number of nodes in our test bed, hence ideally, DRAW can distributes the grouping data on all the nodes, with 100% NHD; for the C.briggsae whose number of blocks is smaller than 40, the ideal DRAW's NHD is calculated as $\#\_of\_Blks/\#\_of\_nodes$, which is shown in bold font in Table III. Our experimental results show that, for the 2-replica Hadoop, DRAW may improve the data distribution parallelism by 27.2% on average; for the 3-replica Hadoop, DRAW is expected to improve the parallelism by 17.6% (without considering the exception of Chicken data) on average.

[3] There is one exception for Chicken: the data is more evenly distributed in 2-replica case than 3-replica.

IV. Related Work

Several previous works exploited the grouping-like data semantics and organized data layout in some specific ways to support high-performance data accesses. Ahmed *et al.* [10] exploited the file-grouping relation to better manage the distributed file caches. The rationale is that group accessed files are likely to be group accessed again. This idea works well in cache management. However in data intensive computing, such grouping behavior is at chunk level rather than file level.

Yuan [24] develops a data dependency-based data placement for the scientific cloud work flows, which clusters the relative data as intensively as possible to effectively reduce data movement during the workflow's execution. However, for data parallel frameworks such as MapReduce which relies on the co-located compute and data locality [3], the relevant data needs to be distributed as evenly as possible.

Xie *et al.* [15] takes data locality into account for launching speculative MapReduce tasks in heterogeneous environments. They focus on balancing data processing load based on network topology and disk space utilization of a cluster. In contrast, DRAW focuses on data redistribution based on data semantics; they are two complimentary works.

MRAP [21] develops a set of MapReduce APIs for data providers who may be aware of the subsequent access patterns of the data after being uploaded. By specifying the access patterns, the data are distributed in a corresponding way such that the best data access performance can be achieved. However, it requires application developers to specify the data access patterns beforehand. Our DRAW captures grouping patterns by runtime tools.

V. Conclusion

The default random data placement in a MapReduce/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. We also theoretically prove that the inefficiency of Hadoop's random placement method. Our experimental results show that for two representative MapReduce applications—Genome Indexing and Astrophysics, DRAW can significantly improve the throughput of local map task execution by up to 59.8%, and reduce the execution time of map phase by up to 41.7%. The overall MapReduce job response time is reduced by 36.4%.

Acknowledgment

This work was supported in part by the U.S. National Science Foundation under Grant CCF-0811413 and Grant CNS-1115665, and by the National Science Foundation Early Career Award 0953946.

References

[1] [Online]. Available: http://aws.amazon.com/s3/

[2] [Online]. Available: http://bowtie-bio.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. Comput. 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 inodes 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 frameshift mutations by comparative analysis of the human and the chimpanzee genome sequences," *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 through data placement in heterogeneous hadoop clusters," Apr. 2010.

[16] G. H. Kuenning and G. J. Popek, "Automated hoarding for mobile computers," in *Proc. 16th ACM Symp. Operat. Syst. Principles, SOSP'97*, New York, 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] 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.

[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 Discov. Sens. Data, SensorKDD'09*, New York, 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.