Comput Syst Sci & Eng (2019) 1: 47–60

International Journal of
Computer Systems
Science & Engineering

CORE - An optimal data placement strategy in Hadoop for data intensive applications based on cohesion relation

Vengadeswaran* **and Balasundaram**

Department of Computer Applications, National Institute of Technology, Tiruchirappalli 620015, India

The tremendous growth of data being generated today is making storage and computing a mammoth task. With its distributed processing capability Hadoop gives an efficient solution for such large data. Hadoop's default data placement strategy places the data blocks randomly across the nodes without considering the execution parameters resulting in several lacunas such as increased execution time, query latency etc., Also, most of the data required for a task execution may not be locally available which creates data-locality problem. Hence we propose an innovative data placement strategy based on dependency of data blocks across the nodes. Our strategy dynamically analyses the history log and establishes relationship between various tasks and blocks required for each task through Block Dependency Graph (BDG). Then Our CORE-Algorithm re-organizes the HDFS layout by redistributing the data blocks to give an optimal data placement, resulting in improved performance for Big Data sets in distributed environment. This strategy is tested in 20-node cluster with different real-world MR applications. The results conclude that proposed strategy reduces the query execution time by 23%, improves the data locality by 50.7%, compared to default.

Keywords: Big Data, Distributed Storage, Parallel Processing, Hadoop, Data Placement, Cohesion

1. INTRODUCTION

In this data era, massive volumes of data are being generated every second in a variety of domains such as Geoscience, Social Web, Finance, e-Commerce, Health Care, Climate modelling, Physics, Astronomy, Government sectors etc. Big Data is the term applied to such large volume of data sets whose size is beyond the ability of the commonly used software tools to capture, manage, and process within a tolerable elapsed time [1]. Further Big data management possess many challenges due to big data diversity, big data reduction, big data integration and cleaning, big data indexing and query, and big data analysis and mining [2]. This situation has led to a rapidly increasing use of parallel and distributed environment framework like Hadoop, to analyze and gain insights from the data. The distributed processing of large data sets across clusters of computers using simple programming models has been facilitated through Apache Hadoop software library [3][4]. The inherent parallelization, synchronization and fault-tolerance offered by the model, makes it ideal for highly-parallel data-intensive applications [5]. Local computation and storage is achieved through the two major components of Hadoop namely Map Reduce (MR) and Hadoop Distributed File System (HDFS). The fundamental concept of HDFS and MR is to distribute data among many nodes and process in parallel.

HDFS [6][7] is a distributed file system designed to run on commodity hardware capable of storing large files across multiple machines. HDFS follows master slave architecture, consisting of one Name-Node and multiple Data-Nodes. Name-Node is the coordinator of HDFS which maintains the metadata i.e., size, location and replicas of the data blocks. Data-Nodes hold the actual storage of data. When you dump a file into the HDFS,

*Corresponding Author. E-mail: meetvengadesh@gmail.com

the files are broken into fixed size blocks, and blocks are stored on the various Data-Nodes in the Hadoop cluster. HDFS creates several replications of the data blocks and distributes them accordingly in the cluster in a way that will be reliable and can be retrieved faster. Data-Node reports the blocks stored in it, to Name-Node periodically thereby updating metadata. When you execute a query from a client, it will reach out to the Name-Node to get the file metadata information, and then it will reach out to the Data-Nodes to get the real data blocks. The most important aspect of Hadoop is that both HDFS and MR are designed with each other in mind and each are co-deployed such that there is a single cluster and thus it provides the ability to move computation to the data and not the other way around [8]. Thus, the storage system is not physically separate from the processing system, due to which the placement of data in Data-Nodes is crucial for efficient processing. Hence there is a huge need for optimal data placement in a HDFS as shown as in Fig. 1.

Data placement strategies mainly focus on two areas. 1.Placing data across the data centers for increasing the parallel execution [9][10][11] resulting in improved performance viz. reduced query latency, reduced query execution time, improved data locality and improved read write performance, throughput. 2.Placing Co-related data together for reducing the resource utilization [12][13][14] viz. reduced energy requirements for powering the computing equipment, minimizing the average query span, reduced network bandwidth, reduced carbon foot print, reduced operational cost. The type of strategy to be adopted depends on the nature of requirements as to whether the solution to the query is time dependent or cost dependent. We focus on data placement strategy for finding solution to queries which are required to be solved at the earliest possible time to enable quick decision making as well as deriving maximum utilization of resources. The real value of analyzing the Big Data is accelerating the time-to-answer, especially in case of streaming data where immediate response for taking better decision is very much desired.

Hadoop's default data placement strategy randomly places the data across the Data-Nodes without considering the storage capacity [15], but this can be overcome by executing the Load balancer utility. Load balancer [16] redistributes the data based on the storage capacity, but there is no guarantee that the data required for execution of any task is evenly distributed across the nodes to ensure Local map task execution. During parallel processing in a distributed environment, if the data required for a node to process is not available locally, then the Map task of that node will be idle or it will access the needed data remotely from some other node where the data is available, this will severely reduce the MR performance [17]. Hence the focus is on achieving maximum parallel execution through innovative data placement strategy. Several works have been done in the field of data placement in HDFS considering various metrics. In this paper, we focus on the inter and intra-dependency of data blocks in a node for a task. Existence of non-dependent data blocks in a node does not contribute towards the efficiency of map task, since they are not involved in any execution. So, we aim at evenly spreading out the dependent data blocks across the nodes which will result in maximum parallel execution. We term the concentration of such dependent data blocks in a node for a task as the Relation cohesion of the node for that task.

In our approach, Historical query workload logs are traced over a period of time and represented as a Block Dependency Graph, where nodes are data blocks and tasks are represented as edges incident on nodes. Cohesive Matrix is constructed from the graph for estimation of Relation and weighted Relation cohesion, which is used as input for the CORE algorithm as proposed. Our algorithm reorganizes the HDFS layout by redistributing the data blocks to give an optimal data placement, which has higher parallel execution resulting in improved performance. The result shows that CORE algorithm has efficiently reduced the query execution time and has improved local map task execution. Also a significant improvement over the read write performance is achieved. The rest of this paper is organized as follows: Section 2 describes the need and necessity of a new data placement algorithm, along with related works and problem definition. A motivating example is also explained in detail in this section. Our proposed CORE-Algorithm is explained in detail in section 3. Section 4 presents the experimental results and analysis; Finally, Section 5 concludes the paper with possible future research directions.

2. MOTIVATION AND PROBLEM DEFINITION

2.1 Cohesion and Coupling measurement

In our context, the term cohesion refers the tightness with which "related" data blocks are "grouped together" in a Data-Node. Coupling represents the amount of relationship, between the elements, belonging to different Data-Nodes illustrated in Fig.2. Let us consider a modular system S consisting of different modules M. Let $R^c(S)$ be the number of internal relations of system, $R^i(S)$ be the number of input relations, $R^0(S)$ be the number of output relations and $R(S)$ be the total number of relations in the system.

Then, Cohesion of the System $[CH(S)]$ is expressed as a ratio between the number of internal relations $R^c(S)$ and the total relations $R(S)$ by the formula.

$$CH(S) = \frac{\#R^c(S)}{\#R(S)} = \frac{\#R^c(S)}{\#R^c(S) + \#R^i(S) + \#R^0(S)}$$

Coupling of the System $[CP(S)]$ is the ratio between the number of external relations $R^i(S) + R^0(S)$ and the total relations of the system $R(S)$

$$CH(S) = \frac{\#R^i(S) + \#R^0(S)}{\#R(S)} = \frac{\#R^i(S) + \#R^0(S)}{\#R^c(S) + \#R^i(S) + \#R^0(S)}$$

It is normally assumed that the better the designer is able to encapsulate related program features together, the more reliable and maintainable is the system [20]. For a good software design "High Cohesion and Low Coupling" is required. But our work focuses on reducing the cohesion in order to achieve maximum parallel execution for improving the system performance. Cohesion as defined in this paper can be termed as the density of related data blocks, located in a node that are required for execution of a particular task. Since we are focusing mainly on reducing the concentration of dependent data blocks from the highly dense Data-Nodes, it is suffice that the cohesion alone is taken into account since Intra-dependency. This can also be

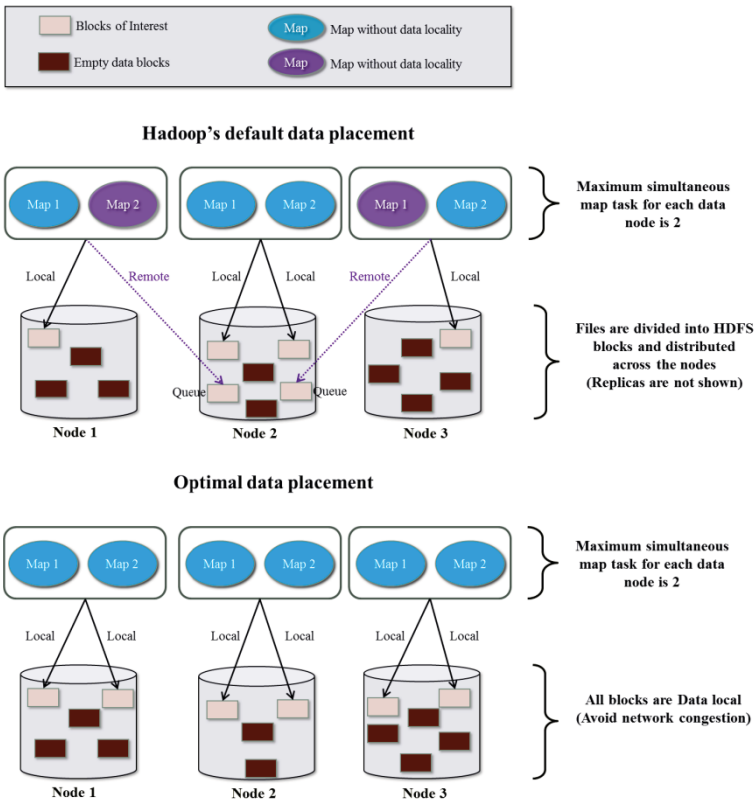


Figure 1 Need and necessity of an optimal data placement.

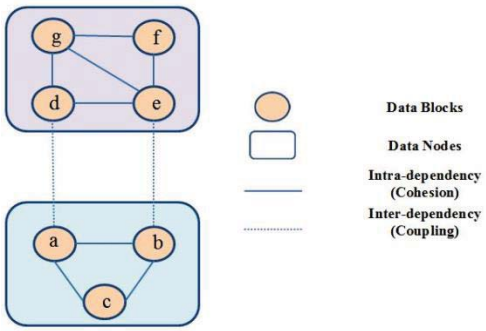


Figure 2 Inter and Intra dependency of data blocks across data nodes.

proved to satisfy the above four important properties as defined in above Table 1. The estimation of cohesion measurements is explained in detail in section 3.

2.2 Related Works

Several works have been done in the field of data placement for larger data in a distributed environment. Yuan et al. [21] give a matrix based k-means clustering strategy to address the issues in data placement in scientific cloud workflows. Accordingly the related datasets are placed in appropriate data centers based on dependencies during the runtime. Dependency matrix is constructed based on the access pattern on the datasets for the set of tasks. The datasets are partitioned by transferring the Dependency Matrix into Clustered Dependency Matrix using Bond Energy Algorithm (BEA). This strategy guarantees balanced distribution of data and reduced data movements. The drawback of this approach is the use of Bond Energy Algorithm to cluster the Dependency Matrix, since the time complexity of finding permutations of all rows every time for BEA is high. Jun Wang et al. (2014) [9] have proposed an optimal data placement strategy based on grouping semantics. This proposed strategy reduces the query execution time and improves the data locality compared to default strategy. It improves parallel execution of data sets having interest locality. But most of the real world applications are without interest locality and in such cases this strategy proves to be ineffective. Chia-Wei Lee et al. (2014) [10] proposed a strategy, that distributes the data blocks based on computing capacity of a Data-Node instead of storage capacity so that faster nodes are provided with more data blocks and can solve HPC application with reduced execution time. However there is no mechanism to ensure that the required data for execution are present in the faster (higher computing) nodes, thereby defeating the purpose of data placement.

Table 1 Properties of Cohesion of a Module and Modular System [18][19].

Cohesion of a [Module | Modular System] The cohesion of a %[module m = <Em, Rm> of a modular system MS|modular system MS] is a function [Cohension(m)|Cohension(MS)] characterized by the following properties.		
S. No.	***Property***	***Definition***
1	**Non-negativity & Normalization**	Cohension of a module and modular system belongs to a specified interval [Cohension(m) ∈ [0,Max]|Cohension(MS) ∈ [0,Max]] Cohension to be normalized so that the measure is independent of the size.
2	**Null Value**	Cohension of a modular system is null if its set of intramodule edges is empty. [Rm = ∅ ⇒ Cohension(m) = 0 | IR = ∅ ⇒ Cohension(MS) = 0] IR is the set of intra-module relationships.
3	**Monotonicity**	Adding intra-module relationships does not decrease [module|modular system] cohension.
4	**Cohensive Modules**	The cohension of a [module|modular system] obtained by putting together two unrelated modules is not greater than the [maximum cohension of the two original modules|the cohension of the original modular system].

An optimal data placement, by co-locating co-related data items together in order to reduce resource consumption, query span (minimum number of machines required to process query) is suggested by Ashwin Kumar et al.(2013) [14]. Their work mainly focuses on reducing cost, but keeping the available resources without utilization is not a viable solution, since the real value of analyzing the Big Data is accelerating the time-to-answer, for taking better decision. Lili Sun et al. (2013) [22] suggested a strategy, by taking into account the disk space utilization and computing capacity of each node, to give an efficient load balancing. Though efficient load balancing is achieved with minimum movement of data across the nodes, it does not ensure data locality, which in turn may reduce local map execution.

Also several research papers are available in the literature to identify the metrics to measure inter and intra dependency through software measurements such as cohesion and coupling. Lionel Briand et al. [18] have suggested a mathematical model which defines several measurement concepts (size, length, cohesion, and coupling) which can be used for software design abstractions. However, specific measurement frameworks for particular product abstractions e.g., Control Flow Graphs, Data Dependency Graphs are not defined. Edward B. Allen et al. [19] give a new approach for measuring the Inter and Intra modular relations which exhibits finer discrimination than counting based measurement. But the usefulness of cohesion is not validated. Mirjana et al. (2014) [23] present a method to establish a set of relationship between particular software metrics and corresponding measures from complex networks theory. Accordingly a complex network measure and its related software metrics measure are considered for defining a relation. The defined relation is tested for establishing formalized metrics.

2.3 Motivating example

In order to prove the proposed work mathematically, the principle was experimented with several miniature examples. One of the motivating examples satisfying the requirements is explained in Fig. 3. Accordingly it comprises of a cluster with four nodes (DN1, DN2, DN3, and DN4) wherein three different tasks (T1, T2, and T3) over 24 data blocks (B1 – B24) are executed.

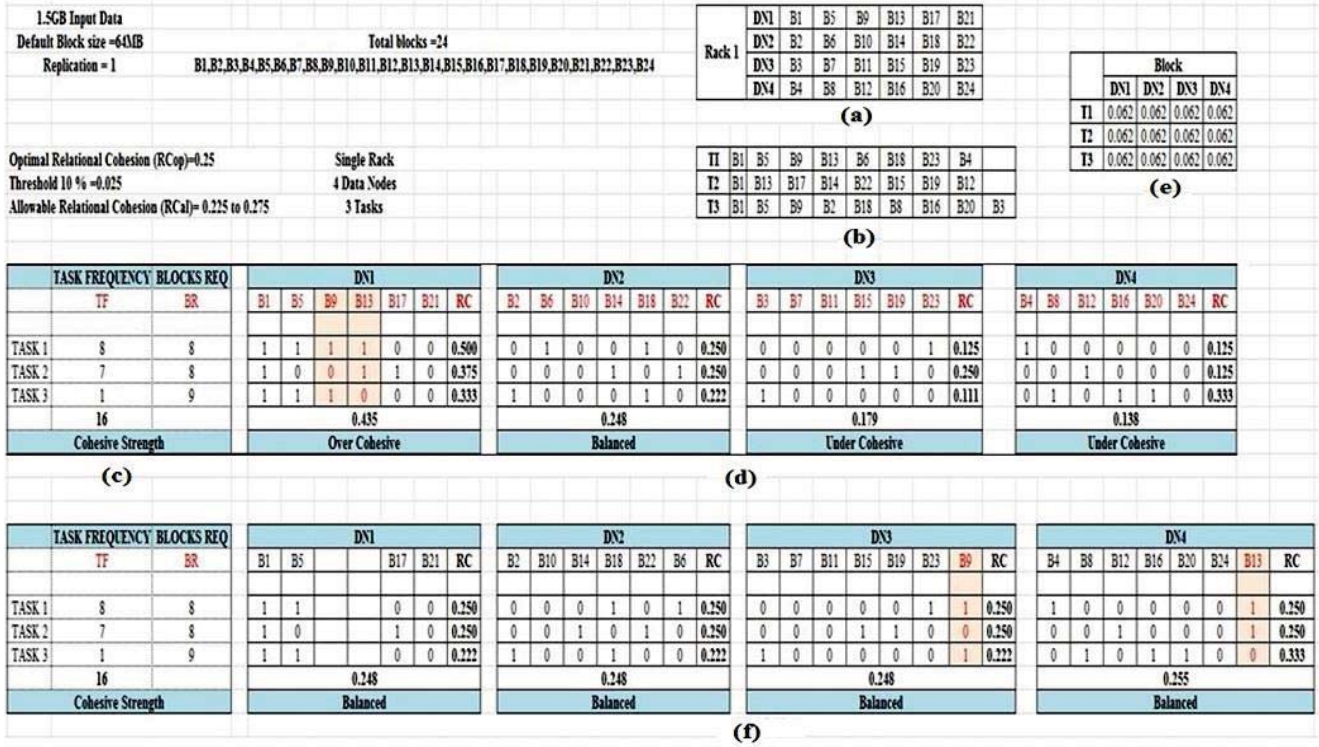
2.4 Problem definition

For a given set of data blocks B to be processed in a cluster having N number of Data-Nodes, as per default, the blocks will be randomly distributed among the N nodes. Let the blocks required for the execution of a particular task T' be B' where $B' \subseteq B$. Even if B is evenly distributed across the cluster, it does not guarantee that blocks required for any particular task is evenly distributed, resulting in reduced system performance. Further ensuring even distribution of B' alone will not be an optimal solution since the task being executed in a system will not be unique. Hence there is a need for an Optimal Data Placement Strategy.

3. CORE-OPTIMAL DATA PLACEMENT STRATEGY

3.1 Proposed Strategy

In this paper, CORE, an optimal data placement strategy is pro-



a) Location of data blocks in data nodes; (b) Blocks required for each task; (c) Task frequency table; (d) Estimation and categorization of data nodes according to initial data layout; (e) Block cohesive value table; (f) Estimation and categorization of data nodes after to final reorganized data layout;

Figure 3 Example showcasing the effectiveness of CORE algorithm.

posed by ensuring even distribution of related data blocks across the nodes to improve the efficiency of the system in a distributed environment. Accordingly, data blocks are placed in such a way that, the concentration of dependent data blocks in a node is balanced so as to improve the degree of parallel execution. The proposed strategy is elaborated in detail. The flow diagram of the entire work is shown in Fig. 4. This section consists of five parts, First part, *User History Log* exploits the system log files and Name-Node meta information to construct Task frequency table and network topology; Second part, *Block Dependency graph (BDG)* depicts the relationship among the dependent blocks; In third part, Cohesive Matrix is constructed to learn about the internal cohesion in the Data-Nodes; The *estimation of cohesion strength* for a Data-Node and modular system is done in the fourth part; The *CORE* which proposes an optimal data placement algorithm is done in the final part.

3.2 User History Log

Log files are typically large in size and contain lot of resources about data, which have to be processed to get useful information. Analyzing the characteristics of cluster for various workloads is the key for making optimal placement decisions. Usually Log files are semi structured. All Map reduce applications executed in cluster, save the task execution details as a log file, which consists of two files

(i) *Job Configuration XML file:* it contains the job configuration as specified when the job is launched,

(ii) *Job Status file* which contains task ID, status, start and end up time etc. for each job executed in the machine. Using this as input, the log files are processed to construct Task frequency table containing the list of different task executed Ti, frequency of occurrence Tf and the required blocks Br for each of the task is shown in Fig. 3(c). Name-Node contains meta data from which the network topology is constructed to identify the different Data-Nodes present in the cluster and the data blocks present in each of the Data-Node (Fig.3(a)). Meta data can be traced in *dfs.namenode.name.dir* configuration property located in hdfs-site.xml. From the above information, the Block Dependency Graph is constructed.

3.3 Block Dependency Graph

The computations of parallel processing can be solved efficiently when the dependency of task on the blocks are mapped through a Block Dependency Graph (BDG). It is an undirected graph which depicts inter and intra dependency of data blocks for each task executed.

BDG of a system S can be constructed as a 3-tuple BDG =<B, R, N> where B represents the set of Blocks, R is a binary relation on B ($R \subseteq B \times B$) representing the relationships between Set of blocks and N is a collection of Nodes of S such that $\forall$ b $\in$ B ($\nexists$ n $\in$ N (n = <Bn, Rn> and e $\in$ Bn)) and $\forall$ n1, n2 $\in$ N (n1 = <Bn1, Rn1> and n2 = <Bn2, Rn2> and Bn1 $\cap$ Bn2 = $\emptyset$)

Fig. 5 is BDG in which three tasks t1, t2 and t3 are executed;

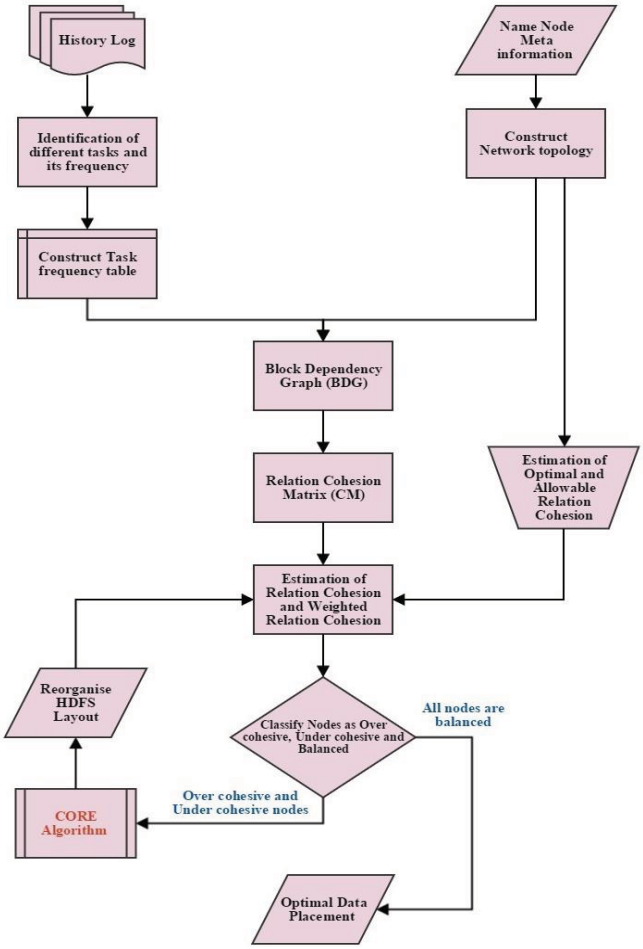


Figure 4 Detailed flow diagram for the proposed work.

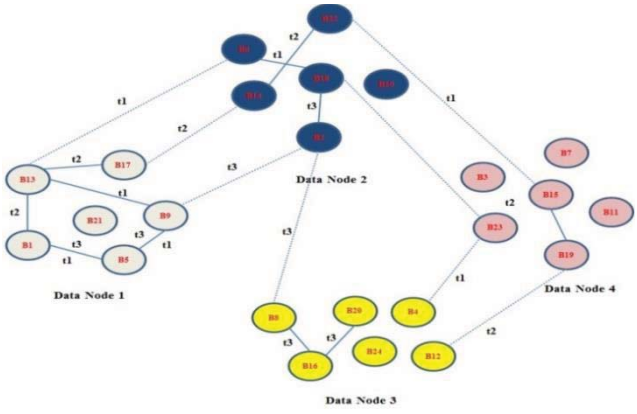


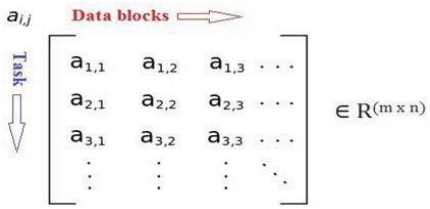
Figure 5 Block Dependency Graph.

t1 requires blocks B1, B5, B9, B13, B6, B18, B23 and B4. t2 requires blocks B1, B13, B17, B14, B22, B15, B19, and B12. t3 requires blocks B1, B5, B9, B2, B18, B8, B16, B20 and B3. From the graph, the edges within a Data-Node connecting the blocks exhibit the degree of intra-dependency of blocks for a task. The edge (task) connecting blocks located in two different nodes is a measure of inter-dependency of blocks for a task. Since this paper focusses on improving the parallel execution by reducing the internal cohesion, the intra-dependency among the blocks of a node alone is taken into consideration.

3.4 Cohesive Matrix

From the Block Dependency Graph (BDG) and the information available from Task Frequency (Tf) table, a Cohesive Matrix (CM) is constructed. CM is a matrix which is used to define

the relation between two objects. The Cohesive Matrix A=[aij] of BDG is the Incidence matrix [24] of size m × n where m is the tasks being executed and n is the data blocks represented as a subset of node in which they are located, such that Cohesive Matrix $A = [a_{ij}]$ $1 \leq i \leq m,\ 1 \leq j \leq n$ with



$$a_{ij} = \left\{ \begin{array}{l} 1 \text{ if the } i^{\text{th}} \text{ edge } m_i \text{ is incident on } j^{\text{th}} \text{ node } n_j \\ 0 \text{ otherwise.} \end{array} \right\}$$

It is a binary matrix or a (0, 1)-matrix as it contains only two types of elements 0 or 1. From the Matrix following properties are inferred.

1. The number of ones in each column (block) equals the degree (i.e. the number of varying edges incident on the node) of the corresponding node.

2. A column with all zeros represents an isolated node (unutilized block).

3. The sum of entries in a row Tm indicates the number of blocks required for that task.

4. The sum of entries in a column Bn indicates the number of tasks for which the block is required.

The incidence structure will be space efficient if there are many more nodes than edges. Fig. 3(d) depicts the presence of a data block in the node relating to a task in binary form. If a task requires a particular data present in the Data-Node for its execution, then its corresponding value will be one else zero. This will be an indicative of the total concentration of the blocks required for a particular task in every Data-Node.

3.5 Estimating cohesive strength for Data-Node and Modular system

The concentration of data blocks in a node Ni that is required for particular task T_i execution with reference to the total blocks required for that task is termed as Relation cohesion of that node $Rc_{<}$Ti,Ni>

$$Rc < Ti,Ni >= \frac{\text{No. of blocks executed in Node Ni for Task Ti}}{\text{Total no. of blocks required Br for that task Ti}}$$

This Relation cohesion for each task in a particular node is standardized with reference to the frequency of each task and the weighted Relation cohesion Rcwa is calculated for every Data-Node.

$$Rc_{wa} = \frac{\sum_{i=1}^{n} (Rc_{<T_i,N_i>} * T_{f_i})}{\sum_{i=1}^{n} T_{f_i}}$$

For ideal parallel execution, the data blocks required for the task have to be evenly distributed across the cluster of Data-Nodes. Hence, the optimal Relation cohesion Rc_{op} desired for a cluster for every node can be estimated from the Cluster configuration.

$$Rc_{op} = \frac{1}{n}$$

Where n is the number of nodes in the cluster.

An Ideal situation may not be practically possible while redistributing the data blocks and hence in order to have an inbuilt flexible redistribution, a configurable threshold limit (default 10%) is fixed and allowable Relation cohesion Rc_{al} for each node is designed as

$$Rc_{al} = Rc_{op} \pm \text{Threshold}$$

Rc_{al} will be the parameter used for categorizing the nodes as over-cohesive or under-cohesive which requires to be balanced. For balancing, the blocks have to be migrated from over cohesive to under cohesive nodes. Every addition or deletion of a related data block in a node (Block involved in any of the task execution) will result in a change in the Rc_{wa} of that node. This change in Rc_{wa} will be the Block cohesive value B_{cv}which is used to estimate the number of blocks that has to be moved out from an over cohesive node and the number of blocks that can be received by an under cohesive node in order to be balanced (Fig. 3(e)).

3.6 CORE- Proposed Data Placement Algorithm:

Our work focuses on balancing the cohesive strength of the nodes by moving the dependent data blocks from higher cohesive strength nodes to lower cohesive strength nodes. It ensures even distribution of required data blocks across available Data-Nodes.

Input: Cohesive Matrix (Default data layout), R_c, Rc_{wa}, Rc_{op}, B_{cv}.

Output: Optimal data layout

- Task Frequency be T_f,
- Blocks Required for the Task be B_r,
- Optimal Relation Cohesion estimated from the Cluster configuration be Rc_{op}
- Relation cohesion of each data node for each task be Rc_n
- Allowable Relation Cohesion be Rc_{al}
- Weighted average of Relation Cohesion for each node Rc_{wa}
- Block cohesive value B_{cv} = Change in Rc_{wa} for every addition or deletion of a block.

Begin
For every Data Node DN_i Compare Rc_{wa} with Rc_{al}

Begin
For every Data Node $\mathbf{DN_i}$ Compare $\mathbf{Rc_{wa}}$ with $\mathbf{Rc_{al}}$
If $(Rc_{op} - Threshold) \leq RC_{wa} \leq (Rc_{op} + Threshold)$
DN_i is balanced, ***//Put this value in Bal-set with Rc_{wa}***
Bal-set [ ] = $[DN_i, Rc_{wa(i)}]$
Else If $(Rc_{wa} > (Rc_{op} + Threshold))$
DN_i is Over Cohesive,
Calculate the number of blocks that can be moved from Over cohesive Node (NB_i)
$NBi = \frac{Rcwa(i) - Rcop}{Bcv}$
OC-set [ ] = $[DN_i, Rc_{wa(i)}, NB_i]$ ***// Put these values in OC-set with Rc_{wa} and NB_i***
Else
DN_i is Under Cohesive,
Calculate the number of blocks that can received by the Under cohesive node (NB_i)
$NBi = \frac{Rcop - Rcwa(i)}{Bcv}$
UC-set [ ] = [DN_i, $Rc_{wa(i)}$, NB_i] ***//Put these values in UC-set with Rc_{wa} and NB_i***
End If
Do until (UC-set [ ] and OC-set [ ] empty) ***// Until cluster cohesively balanced***
Source Node SN = Max Rc_{wa} (OC-set [ ])
Receiving Node RN = Min Rc_{wa} (UC-set [ ])
Number of Data blocks to be moved is B_i = Min (NB_i(SN),NB_i(RN))
From the Table1 relating to Source Node (SN) **//Choose the Victim Block**
For each Row M[n][n] from OC-set [ ] do
Find Row having Max Sum,
Victim blocks list [ ] = Blocks corresponding to Row having Max Sum
For each Victim blocks list [ ]
Do until B_i choose the Victim Blocks
Find difference in RC of SN and RN corresponding to each Task
Delete blocks relate to task having Min diff. from Victim blocks list [ ]
End Until.
End For
Victim block Vb = Remaining blocks in Victim blocks list [ ]
End For
Move the Vb from SN to RN ***// Repeat the iteration, cluster cohesively balanced***
End Until ***// Repeat the iteration, until UC-set [ ] and OC-set [ ] empty***
End For

According to our optimal data placement algorithm, the higher cohesive strength of a Data-Node exhibits the existence of more number of Inter dependent blocks in the Data-Node for a particular task. The algorithm starts with the initial estimation of Relation cohesion R_c for each task in a Data-Node and weighted Relation cohesion Rc_{wa} of each Data-Node based on the frequency of task. Each Data-Node is categorized into over cohesive, under cohesive, and balanced based on Rc_{wa} and Rc_{al}. For our example Rc_{op} is 0.25 and hence Rc_{al} will be 0.225 to 0.275. From Fig. 3(d), according to default data block placement DN1 is over cohesive, DN2 is balanced and DN3, DN4 are under cohesive. The default block value B_{CV} (change in R_c for addition or deletion of any data block in the cluster) is estimated as 0.062 as shown in Fig. 3(e).

For the first iteration DN1 will be the Source Node [SN], DN4 will be the Receiving Node [RN]. Since DN1 has the highest Rc_{wa} in the list of over cohesive nodes and DN4 has the least Rc_{wa} in the list of under cohesive nodes. The number of blocks that can be moved out from Data-Node1 to balance it will be 2; the number of blocks that DN4 can receive for balancing is 1 (refer Fig. 3). Hence the number of blocks that can be moved in this iteration be 1 (Min (2, 1)). The victim block to be moved out is to be identified from the Source Node DN1. From Fig. 3(d) relating to SN i.e. DN1, The row having maximum sum (indicates the task having maximum cohesion) is identified. In our example row 1 relating to task 1 has maximum sum of 4. Then the victim block list [ ] that contains all blocks associated with the task corresponding to max sum of the row i.e. B1, B5,

B9, B13 is created. Then the difference in Rc between the SN and RN for each of the task is found. Tabulate the values as $<T_i, Rc_{diff}>$. The task which has the lowest difference will be identified (T3) and the blocks associated with that task (T3) in DN1 are deleted from the victim block list (B1, B5, B9). Remove the task and difference from the table. The iterations have to be continued until the required number of blocks to be moved NB_i is only available in the victim block list. In our example B13 is victim block for the first iteration. Victim block [B13] is moved from SN [DN1] to RN [DN4]. After the movement of data blocks from SN to RN, there will be a change in R_c and Rc_{wa} in the Cohesive Matrix. The algorithm continues the process of categorizing the nodes and follows all the above steps. The process is continued until over-cohesive set (OC-Set[ ]) and under-cohesive set (UC-Set[ ]) become empty. This will end up with all nodes becoming cohesively balanced with reference to Relation cohesion (R_c) as in Fig. 3(f).

The maximum number of simultaneous map tasks on each node is limited by the hardware capacity; currently it is ≤ 2 in most of the clusters. Considering simultaneous map task each Data-Node is 2, the utilization % of local map task at each Data-Node for every task is calculated. The average utilization % for every task is calculated as UT_i.

$$UT_i = \frac{\sum_{i=1}^{n} (\text{utilization \% at each node for the task})}{\text{Total no. of nodes}}$$

Then average utilization for the cluster is also calculated by giving weightage to frequency of each task.

$$\text{Average utilization of map task \%} = \frac{\sum_{i=1}^{n} (UT_i * T_{f_i})}{\sum_{i=1}^{n} T_{f_i}} = \frac{((75*8)+(87.5*7)+(87.5*1))}{16} = 81.25$$

CORE algorithm iteratively changes the default block placement in each Data-Node to an optimized location which in turn is proved to be more effective in local task execution. Fig. 6 and Fig. 7 are local map task for initial and final data layout. The results tabulated in Fig. 7 which shows that local task execution percentage is increased by 18.75%. CORE algorithm does not guarantee 100% local map task execution every time since there is possibility of variation depending on the size of dataset, data block size, satisfying the load balancing and rack awareness but CORE will always produce an improved result over the default data placement strategy which is tested with several examples.

4. EXPERIMENTAL RESULTS AND ANALYSIS

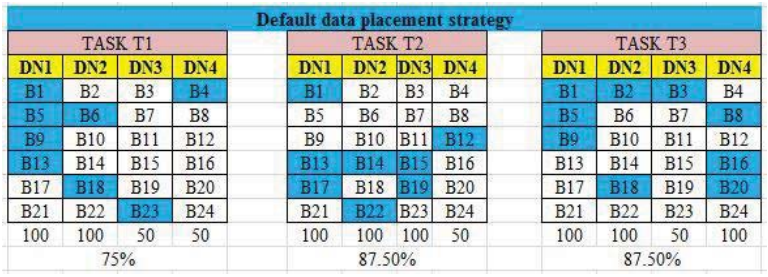
Our proposed CORE algorithm is tested in 20 node cluster placed in a single rack with Hadoop-1.2.1 installed in every node. One node is configured as the Name-Node and the remaining 19 nodes are Data-Nodes. The Data-Nodes are provided with different configuration so as to have a heterogeneous environment. The detailed cluster configuration is shown in Table 2. The implementation of CORE algorithm will dynamically reorganize the HDFS data layout to present an optimal placement for execution; the program is launched as a utility to be executed manually as and when required.

The dataset used in our experiments is a collection of daily weather measurements collected by National Climatic Data Centre (NCDC) and it is a public data set available for download from Amazon s3 [25]. The data are collected every hour for about 18 metrological elements (e.g. Temperature, Wind speed, Humidity etc.) from over 9000 weather stations located globally for the period from 1929 to 2009. The data is strictly ASCII, with a mixture of character data, real values, and integer values. Data files are organized according to date and location of weather station. The records for every year is present in a directory in which there will separate gzipped file (.gz) for the data relating to each weather station. The size of the dataset is about 20 GB, distributed as blocks across the 19 Data-Nodes for our experiment.

The dataset required for analysis will be uploaded in a bulk at an instance. The default strategy will randomly distribute the dataset as even sized blocks across the available Data-Nodes. This strategy does not consider the nature of queries likely to be executed in the system. Even though the dataset that is available is unique, the nature of queries executed will exhibit some interest localities which cover only a part of big data. The interest locality may be different for different domain analyst based on the Geographical location, Metrological element, etc. For example, Metrological scientists belonging to a country will be interested in the data relating to their country alone and in such case the queries executed will have a common dependency among the data related to that country. Similarly specific domain analyst working on forecasting of rainfall will have an interest domain with reference to rainfall particulars which sweeps only a small part in this huge dataset. Since such interest locality are not taken into consideration, there is a likelihood of required dependent blocks for execution of any interest based query to be concentrated within a few nodes alone thereby initiating non local map task, resulting in poor performance. CORE measures density of such dependent blocks within a node and aim at evenly distributing related data blocks for queries across nodes.

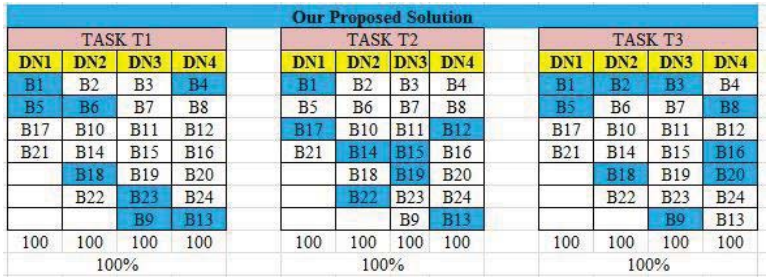
The experiment was conducted by executing various tasks on weather dataset. The tasks T1 to T6 were chosen in such a way that it has specific dependent blocks over the entire 320 blocks (20GB). For example, finding the minimum temperature during a particular period of years, finding the maximum rainfall in a certain region, finding the day of maximum temperature for a particular station etc. The results are listed in Table 3, 4 and 5 which shows percentage improvement of local map execution for data placement based on CORE algorithm against the default random placement strategy.

The improvement in reducing the total execution time is also listed. From Table 3 out of the 328 maps required for execution, default random placement strategy has 130 maps executed locally (i.e. 39.6%) whereas as per CORE the local maps executed is 196 (i.e. 59.8%). Hence the overall improvement in local map task execution will be 50.7% ((196–130)/130). The CORE algorithm was tested with various tasks and it has always



Default data placement strategy											
TASK T1				TASK T2				TASK T3			
DN1	DN2	DN3	DN4	DN1	DN2	DN3	DN4	DN1	DN2	DN3	DN4
B1	B2	B3	B4	B1	B2	B3	B4	B1	B2	B3	B4
B5	B6	B7	B8	B5	B6	B7	B8	B5	B6	B7	B8
B9	B10	B11	B12	B9	B10	B11	B12	B9	B10	B11	B12
B13	B14	B15	B16	B13	B14	B15	B16	B13	B14	B15	B16
B17	B18	B19	B20	B17	B18	B19	B20	B17	B18	B19	B20
B21	B22	B23	B24	B21	B22	B23	B24	B21	B22	B23	B24
100	100	50	50	100	100	100	50	100	100	50	100
75%				87.50%				87.50%			

Figure 6 Local map task execution for default data placement.



Our Proposed Solution											
TASK T1				TASK T2				TASK T3			
DN1	DN2	DN3	DN4	DN1	DN2	DN3	DN4	DN1	DN2	DN3	DN4
B1	B2	B3	B4	B1	B2	B3	B4	B1	B2	B3	B4
B5	B6	B7	B8	B5	B6	B7	B8	B5	B6	B7	B8
B17	B10	B11	B12	B17	B10	B11	B12	B17	B10	B11	B12
B21	B14	B15	B16	B21	B14	B15	B16	B21	B14	B15	B16
	B18	B19	B20		B18	B19	B20		B18	B19	B20
	B22	B23	B24		B22	B23	B24		B22	B23	B24
		B9	B13			B9	B13			B9	B13
100	100	100	100	100	100	100	100	100	100	100	100
100%				100%				100%			

Figure 7 Local map task execution for proposed data placement.

Table 2 Cluster Configuration.

20 Node Cluster			
Property	*Name Node-1*	*Data Node-19*	
	NN-1	DN-10	DN-9
vCPU	8	4	2
RAM	8 GB	8 GB	4 GB
Hard Disk	300 Gb	300 GB	150 GB
Processor	Intel(R) Core i7-3770 CPU @ 3.40 GHZ		
OS	Ubuntu Server 14.04 LTS		
Hadoop	Hadoop 2.7.2 (Stable Version)		

Table 3 Performance improvement in Local map task and Execution time - For varying tasks size.

Dataset	**Size**	**Blocks**	**Task**	**Total maps**	**Default**		**CORE (Proposed)**		**Improvement over default**	
					Local maps (%)	Exe. time (secs)	Local maps (%)	Exe. time (secs)	Local maps (%)	Exe. time (secs)
Weather dataset (NCDC)	**20 GB**	**320**	**T1**	**32**	50	1036	87.5	677	37.5	359
			T2	**80**	35	2851	42.5	2421	7.5	430
			T3	**66**	30.3	2419	54.5	1773	24.4	646
			T4	**50**	52	1598	72	1182	21	416
			T5	**40**	45	1339	65	1066	19	273
			T6	**60**	36.7	2116	60	1614	24.5	502
The results are taken after executing Load balancer utility, a Hadoop daemon										

shown an improvement over the default data placement. The results for only 6 sample cases have been listed in Table 3. When tested in worst case where any interest locality does not exist or all data blocks are required to be accessed for execution of the task, CORE has shown the same efficiency as default.

To strengthen the effectiveness of CORE algorithm, and to make a study of its behavior at various cluster size, the experiment was conducted for a sample task by varying the number of nodes. The task was executed on the same dataset and results are tabulated in Table 4. With the increase in Data-Nodes, the availability of maximum number of simultaneous map task will increase, which will naturally increase the degree of parallel execution. Our experiment gives an interesting result, that CORE algorithm is more efficient for bigger clusters. i.e., the incremental improvement of efficiency is high when the number of nodes in cluster is more. The results shown in graph (Fig. 8.) prove that the efficiency of the CORE algorithm increases with the increase in size of cluster. Our proposed algorithm also significantly reduces the Map and Reduce completion time as shown as in Fig. 11.

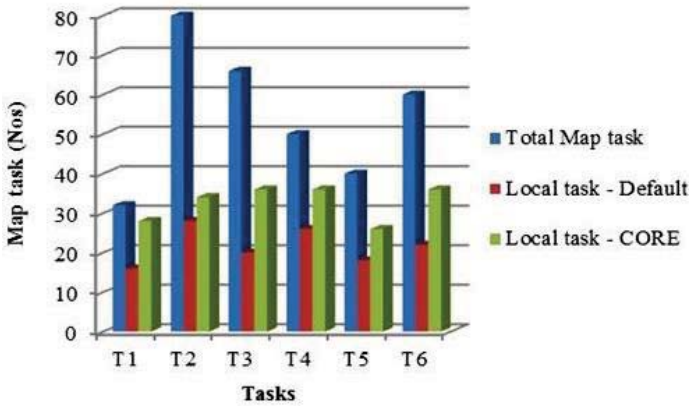


Figure 8 Graph showing the performance improvement in local map task for default and CORE.

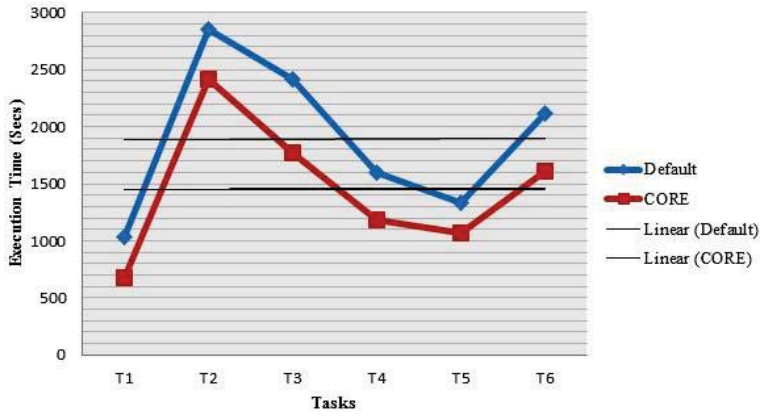


Figure 9 Graph showing the performance improvement in execution time for default and CORE.

Table 4 Overall comparison of default and CORE data placement strategy.

	Total maps (nos)	Local maps (nos)	Local maps (%)	Exe. time (secs)
Default	**328**	130	39.6	11359
CORE (Proposed)	**328**	196	59.8	8733

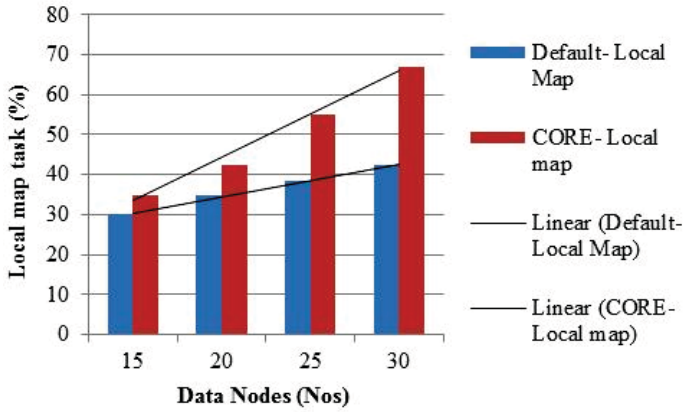


Figure 10 Performance improvement in local map for default and CORE âŁ“ For varying cluster size.

5. CONCLUSIONS AND FUTURE WORK

Hadoop's default data placement strategy places the data blocks evenly across the Data-Nodes. Even though, blocks are evenly distributed across the cluster, it does not guarantee that blocks required for execution is evenly distributed, which severely drag down the system performance during Map reduce task. So, we

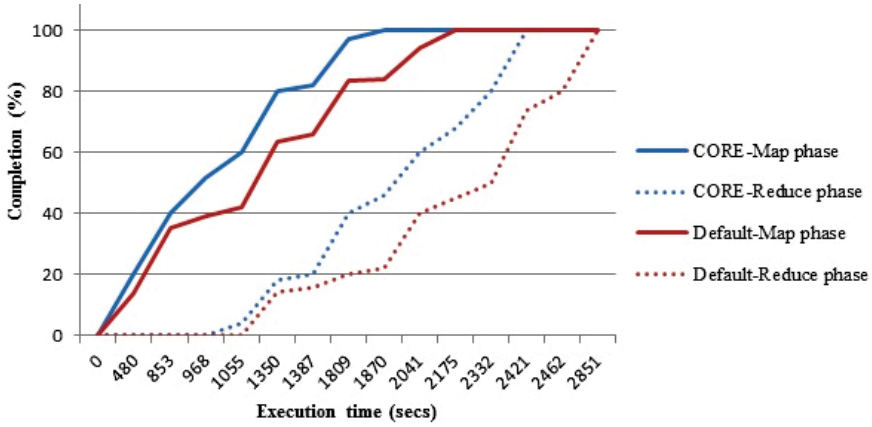


Figure 11 Map and Reduce completion % for default and CORE while executing task T2.

have proposed an innovative data placement strategy which will distribute the related data blocks i.e. blocks required for execution, evenly across the Data-Nodes to ensure maximum parallel execution. This has been experimentally tested in a 20 node cluster using different map reduce applications. The result has strengthened our proposed algorithm and has proved to be more efficient for massive datasets by reducing query execution time by 23% and significantly improves the data locality by 50.7% compared to Hadoop's default data placement strategy.

Even though the results are most optimistic, our experiment has been conducted in a single rack topology without any replicas. We know that further data copies (replicas) will certainly improve the performance and reduce the overheads, but its behavior in cross rack environment has to be further studied for improved performance. Also due to migration of data blocks, ultimately cluster may become imbalance. Hence a new and efficient load balancing with optimal data placement is being focused considering the dependency of data bocks.

Acknowledgments

The research work reported in this paper is supported by Department of Electronics & Information Technology (DeitY), a division of Ministry of Communications and IT, Government of India., under Visvesvaraya PhD scheme for Electronics & IT.

REFERENCES

1. [Online]. Available:https://en.wikipedia.org/wiki/Big_data
2. Chen J, Chen Y, Du X, Li C, Lu J, Zhao S, Zhou X. Big data challenge: a data management perspective. Frontiers of Computer Science, 2013, 7(2): 157–164.
3. [Online]. Available:https://hadoop.apache.org/
4. [Online]. Available:https://developer.yahoo.com/hadoop/tutorial/
5. Fadika Z, Govindaraju M, Canon R, Ramakrishnan L. Evaluating hadoop for data-intensive scientific operations. In: IEEE 5th International Conference on Cloud Computing (CLOUD). 2012, 67—74.
6. [Online]. Available:https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
7. Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). 2010, 1—10.
8. [Online]. Available:http://docs.hortonworks.com/HDP2Alpha/index.htm#About_Hortonworks_Data_Platform/Understanding_Hadoop_Ecosystem.html
9. Wang J, Xiao Q, Yin J, Shang P. Draw: A new data-grouping-aware data placement scheme for data intensive applications with interest locality. IEEE Transactions on Magnetics, 2013, 49(6): 2514—2520.
10. Lee C W, Hsieh K Y, Hsieh S Y, Hsiao H C. A dynamic data placement strategy for hadoop in heterogeneous environments. Big Data Research, 2014, 1: 14—22.
11. Liu L. Computing infrastructure for big data processing. Frontiers of Computer Science, 2013, 7(2): 165—170.
12. Eltabakh M Y, Tian Y, Özcan F, Gemulla R, Krettek A, McPherson J. Cohadoop: flexible data placement and its exploitation in hadoop. Proceedings of the VLDB Endowment, 2011, 4(9): 575—585.
13. Maheshwari N, Nanduri R, Varma V. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. Future Generation Computer Systems, 2012, 28(1): 119—127.
14. Kumar K A, Deshpande A, Khuller S. Data placement and replica selection for improving co-location in distributed environments. arXiv preprint arXiv:1302.4168, 2013.
15. White T. Hadoop: The definitive guide. "O'Reilly Media, Inc.", 2012
16. [Online]. Available:https://issues.apache.org/jira/browse/HADOOP-1652
17. Xie J, Yin S, Ruan X, Ding Z, Tian Y, Majors J, Manzanares A, Qin X. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In: IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW). 2010, 1—9.
18. Briand L C, Morasca S, Basili V R. Property-based software engineering measurement. IEEE Transactions on Software Engineering, 1996, 22(1): 68—86.
19. Allen E B, Khoshgoftaar T M. Measuring coupling and cohesion: An information-theory approach. In: Proceedings of the Sixth International Software Metrics Symposium. 1999, 119—127.
20. Salgado R, Diaz O, Marrero M, Mendez I, Lara S. An object-oriented metric to measure the degree of dependency due to unused interfaces. In: Computational Science and Its Applications ?? ICCSA 2004, volume 3046 of *Lecture Notes in Computer Science*, 808—817. Springer Berlin Heidelberg, 2004.

21. Yuan D, Yang Y, Liu X, Chen J. A data placement strategy in scientific cloud workflows. Future Generation Computer Systems, 2010, 26(8): 1200—1214.
22. Sun L, Yang Y, Xiong Z, Zhao X. Intelligent block placement strategy in heterogeneous hadoop clusters. Journal of Convergence Information Technology, 2013, 8(8).
23. Rakic G, Savi' c M, Budimac Z, Ivanovi' c M. Toward the formalization of software measurement by involving network theory. In: 13th Serbian Mathematical Congress. 2014.
24. [Online]. Available:http://mathworld.wolfram.com/Incidence Matrix.html
25. [Online]. Available:http://aws.amazon.com/datasets/daily-global-weather-measurements-1929-2009-ncdc