Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 387 – 402
© TÜBİTAK
doi:10.3906/elk-2003-156

A novel data placement strategy to reduce data traffic during run-time

Sridevi SRIDHAR[*], **Rhymend Uthariaraj VAIDYANATHAN**
Ramanujan Computing Center, Information and Communication, Anna University, Chennai, India

Received: 26.03.2020 • **Accepted/Published Online:** 13.08.2020 • **Final Version:** 27.01.2021

Abstract: High impact scientific applications processed in distributed data centers often involve big data. To avoid the intolerable delays due to huge data movements across data centers during processing, the concept of moving tasks to data was introduced in the last decade. Even after the realization of this concept termed as data locality, the expected quality of service was not achieved. Later, data colocality was introduced where data groupings were identified and then data chunks were placed wisely. However, the aspect of the expected data traffic during run time is generally not considered while placing data. To identify the expected data traffic, the knowledge of the history of data movements is useful. In this work, this knowledge is utilized and an approach to intelligently select the nodes for placing data groups to ensure the least possible data movements is proposed. Systematic scrutiny of log files is conducted and a gain matrix is generated based on maximum likelihood estimation of data movements. Formally, the gain matrix is inversely proportional to the expected data traffic inside the data center. It reflects the performance gain obtained by assigning a block to a node with the lowest possible future data movements. To identify the optimal placement, a many-to-one assignment problem-based algorithm is presented. By experimental analysis, it is observed that the movement of data is significantly reduced by the proposed approach. It is also found that the performance has improved considerably.

Key words: Data management, data organization, cloud computing, parallel and distributed systems

1. Introduction

In this information age, various researchers and industrial pioneers have joined hands to create economic and social value from big data. An increasing demand for quick insights from raw data has urged the research community to focus on introducing fundamentally new approaches in terms of architecture, tools, and techniques. Proper data management is critical for efficiently processing large datasets. Improper data organization and management will lead to high data traffic in the processing infrastructure. This affects the performance that directly impacts the business. Thus, data traffic is a critical bottleneck that needs to be minimized. To handle this issue, data scientists introduced the concept of data locality. The idea was to move tasks to the content servers where processing happens, rather than bringing data to the processing nodes. It was a major paradigm shift that made it possible to process big data in virtualized servers.

Later, in practical scientific and engineering applications, an interesting property called ‘interest locality’ was identified. Interest locality revealed that data scientists usually sweep data sets only as specific parts and not as a whole. And, the frequency of accessing one particular group of datasets is more likely than others. According to this property, ‘data groupings’ played an important role while placing data in a server. This was termed as data colocality in the literature. In the past few years, data colocality based data placement

[*]Correspondence: sridevioct90@gmail.com

techniques have gained a lot of attention. Researchers across the globe applied multiple variants of the data colocality based data placement schemes and have shown performance improvements. However, the knowledge based on history of data movements across data centers is not explored yet.

In Google's PageRank, the performance of matrix-vector multiplication is critical. This algorithm is best solved using the MapReduce framework as the inputs are practically very large. In this algorithm, the vector is assumed to fit in the main memory of the node in which the large sparse matrix is split and processed. This necessitates that the vector needs to be moved to the mapper node to be multiplied with the matrix. This data movement happens for every input split that has been assigned a specific mapper leading to numerous data block movements. Such critical cases when data dependencies give rise to data movements need to be addressed. Many popular applications including genome analysis and indexing, weather simulations, climate, atmospheric modeling etc. have instances where data dependencies lead to data movements across data centers. Hence, a strategy that is based on the knowledge of data movements will be of great value.

In this work, considering historic data movement information, an attempt to model a decision-making system that decides on which node a particular data block has to reside is made. Hence, this work is termed as 'migration aware data placement' (MAD-placement) strategy. Here, 'migration' refers to the data movements during run-time. In this strategy, the 'gain matrix' is introduced to represent the gain obtained by assigning a data block to a particular node subject to the least number of data migrations. Utilizing the historic information, the gain matrix is generated based on maximum likelihood estimation during build-stage. Then the data groupings are identified based on the data colocality concept. These data groups are then placed in the nodes by systematically evaluating the gain matrix. Thus, this problem drills down to a many-to-one assignment problem which is handled using linear programming. This results in optimal data placement that ensures minimum data movement during run-time.

The rest of this paper is organized as mentioned below: Section 2 deals with the related work. The design of the proposed system is given in Section 3. It is followed by the grouped assignment based algorithm. The experimental setup and the analysis conducted is discussed in Section 4. The concluding remarks are given in Section 5.

2. Related work

Several researchers have exploited data grouping and semantics and have come up with wise strategies for data organization and placement. Being the era of big data, data placement strategies have evolved effectively over the time. It started by placing data in the nodes in a random fashion. Then, the effects of placing similar or frequently accessed data groups together were studied. This led to colocality, an idea of placing similar data blocks together in a single node. Data colocality suggests aggregation of data and placing in the appropriate nodes. The importance of aggregation of data for improving the overall performance is given in [1–3]. Generally, groups of data were aggregated and placed together. But, in the work by Wang, Xiao, Yin and Shang [4], to improve the parallelism, highly affinitive groups of data blocks were placed at different nodes. The affinity among the data blocks were identified using the log information usually available in the master node of the cluster. In the case of Apache Hadoop, the log information is available in the name node. Identifying the affinity using bond energy algorithm [5] shows how data block and task mappings are derived from logs and how they are converted into history of data access graphs (HDAG). To improve disk-write throughput and memory-write throughput, a two-level data layout approach is presented in the work by Hao, Jin and Yue [6]. Here, the groups of data are initially stored in a distributed memory file system (DMFS) on top of the Hadoop distributed file system (HDFS) before storing the data directly in the HDFS. Once the smaller data are merged to result in a larger file, they are flushed to the HDFS. This two-level data layout approach was implemented and tested by clustering data from various sensors. This rationale works well at a file-level. But, considering the data intensive computing frameworks, it is desirable to group the data at block-level rather than file-level.

The strategy by Mahdi Ebrahami, Aravind Mohan, Andrey Kashlev and Shiyong Lu [7] involved a metaheuristic-based optimization algorithm that places data optimally while minimizing scientific workflow communication among the nodes. A data interdependency matrix was generated to identify which pairs of data blocks were highly dependent. Then, every possible and legal data placement was evaluated and their heuristic values were computed. After applying several iterations of genetic algorithm operators, such as selection, crossover and mutation, the best placement was identified. A similar data placement scheme based on a genetic algorithm was presented by Xu, Xu and Wang [8]. However, in these metaheuristic-based strategies, the time taken to reach at a global optima after the several iterations involved is quite high.

Research by Zhang, Chen, Lou and Song [9] presented an operations research model that solves the data placement problem. A location aware placement strategy was modeled as a mixed integer programming problem and then solved. Another strategy by Runqun Xiong, Junzhou Luo and Dong [10] was to place data in a snake-like manner across the nodes according to what they term as 'hotness' of data. This work showed that the snake-like placement is energy-efficient and uses less storage. This snake like placement sometimes result in 'confliction'. A situation when the placement suffers a confliction and a queue mechanism for rescue of homeless blocks is given. Although the above technique is energy and space-efficient, it does not consider data locality and the overhead incurred due to the data movements during execution.

As it is well-known, there are two types of data placement strategies: the one which places data in bulk and the other one which adaptively places streaming data. One such adaptive data distribution strategy was presented by Juan M. Tirado, Daniel Higuero, Javier Garcia Blas, Florin Isaila, and Jesus Carretero [11]. Here, an adaptive prediction and data grouping technique was systematically studied and implemented. Several factors like, server utilization, load balance, and data locality were compared to show the efficiency of the adaptive framework. Workload heterogeneity is one such factor that is critical while dealing with adaptive frameworks. However, the framework does not consider this factor with respect to adaptive data grouping and system sizing. Also, the effect and cost incurred due to the data movements during execution is not studied in this work.

Based on the above research works, it is evident that the research and development and, the industry sector are constantly in search of an optimal data placement strategy. The existing data placement strategies basically involve huge computations as they apply metaheuristics. Also, critical scientific computations involve huge data movements which affects the system performance. Hence, a simple data placement strategy that strongly supports lesser data movements during run-time is valuable.

3. Migration aware data placement

Here, we propose a data placement strategy that identifies the best possible node for groups of data blocks based on run-time data movement history. The history of data access and the history of data movements are analyzed based on which the best possible data placement is identified. The proposed strategy is built as an overarch of a strategy which was termed as data grouping aware data (DRAW) placement scheme [4]. This strategy identifies the history of data access. Based on this history, the data groups with a high chance of being accessed together in the future are identified. Then, they are placed in different nodes to improve the parallelism. To achieve the complete success of implementing this existing data placement strategy, we propose to analyze the history of data movements. Based on this history information, data block groups that are most likely to be moved to a particular node for an upcoming user request are identified. This is captured as a gain matrix which gives the relation between a block and a node. When the matrix value is higher, a future migration is least likely to happen. Utilizing the gain matrix, for each of the data groups, an optimal node location is identified. To ensure optimal data placement, a many-to-one assignment problem termed as grouped assignment problem is presented.

The process of the migration aware data placement strategy consists of three major steps: gain matrix generation, data grouping, and optimal assignment. These steps are discussed in the following sections.

3.1. Gain matrix generation

In the first step, knowledge based on history of data movements is exploited and the gain matrix is generated. The values in the matrix represent the gain obtained by placing a data block in a node in terms of minimal data movements. The gain matrix is generated based on the idea that, if the data block remains to stay in a particular node during run-time, its gain relation with that node is higher. The gain of placing block i to node j is directly proportional to the history of access of block i placed at node j and accessed by the node j. And, the gain is proportional to the probability of not observing a future movement of block i from node j. The probability of no future movement is expressed as (1-maximum likelihood of future data movement). Hence, it is expressed as a function of the two terms as shown in (1).

$$G_{ij} = f\{acc_{ij} * (1 - \hat{\boldsymbol{\theta}}_{MLE_{ij}})\} \tag{1}$$

where G_{ij} denotes the gain value of block i with respect to node j. acc_{ij} is the history of access frequency of data block i at node j by node j within a time slot Δt expressed in (2).

$$acc_{ij} = (n_{ij}/total) \tag{2}$$

where, for a given duration Δt, n_{ij} denotes the number of accesses of the data block i at node j by the node j and *total* denotes the total number of accesses of the data block i. $\hat{\boldsymbol{\theta}}_{MLE_{ij}}$ gives the maximum likelihood estimate [12] of future movement of block i which is expressed based on history of data movements as in (3).

$$\hat{\boldsymbol{\theta}}_{MLE_{ij}} = arg\,max_{\theta\in\Theta}\{(f(acc_{ij}|\theta))\} \tag{3}$$

The movement probability of i^{th} block is expressed as f($acc_{ij}|\theta$) where θ denotes the vector of the corresponding block movements during run-time obtained from the HDFS logs. The term f($acc_{ij}|\theta$) represents the probability of observing the system logs and representing it as a function of θ. Specifically, in restricted parameter space, $\theta = \{\theta : \theta \in R^k, h(\theta) = 0\}$ where $h(\theta) = [h_1(\theta), h_2(\theta), \ldots, h_r(\theta)]$ which is a vector valued function mapping R^k to R^r. The procedure to maximize the above function by maximizing the products obtained by the function is tedious. Hence, we use the sample analogue of the log likelihood. As logarithm is an increasing function, it is widely used as an equivalent to maximizing the likelihood function. It is represented below:

$$\hat{\boldsymbol{\theta}}_{MLE_{ij}} = \sum_{i=1}^{n}\{log\,(f(acc_{ij}|\theta))\} \tag{4}$$

Consider the fact that we are interested to find the maximum likelihood estimate of a block movement during run-time. Hence, our aim is to identify p, which is the probability of movement of i^{th} data block. Suppose, 80 tasks were executed in total within the duration Δt. As each map task processes one data block, there are 80 data blocks of interest. For the available sample, consider that we categorize blocks with respect to a particular node whether it was moved as 'Y' and not moved as 'N'. the sample would simply be $[X_1 = Y, X_2 = Y, X_3 = N, \ldots, X_{80} = Y]$. From this, the count of number of movements 'Y' is observed. This implies that a probability of no future movements is $1 - p$ (where p is $\hat{\boldsymbol{\theta}}_{MLE_{ij}}$ as above). Suppose there were 51 movements and 29 in situ task executions, and suppose that the data block i with the probability of data movement from node 1, 2 and 3 are observed as p = 1/3, 1/2 and 2/3 respectively, maximum likelihood estimate can be found as given below: By using the probability mass function of the binomial distribution, with sample size as 80,

$$p[X = \text{`}Y'|p = 1/3] = \binom{80}{51}(1/3)^{51}(1-1/3)^{29} \approx 0.000001 \tag{5}$$

$$p[X = \text{`}Y'|p = 1/2] = \binom{80}{51}(1/2)^{51}(1-1/2)^{29} \approx 0.04316 \tag{6}$$

$$p[X = \text{`}Y'|p = 2/3] = \binom{80}{51}(2/3)^{51}(1-2/3)^{29} \approx 0.07950 \tag{7}$$

Therefore, the likelihood of movements is maximum when p = 2 / 3. i.e. when the data block i is placed at node 3, the likelihood of future data movement is very high. This restricts the gain value of block i with respect to node 3 when substituted in (1). Also, it is worth noting that, like in other similar applications, if the observed parameter follows uniform distribution, the maximum likelihood estimate coincides with Bayes estimator.

3.2. Data grouping

The next step is to identify the data grouping information. Data groups are found as per the steps detailed in the DRAW strategy. The various matrices computed one after the other to identify the data grouping are discussed briefly as given below:

(i) **Attribute usage matrix:**
Attribute usage matrix is the matrix revealing the relationship between the two entities. Two entities considered here are the tasks and the data blocks. From HDAG, information about the relationship between data blocks and tasks are available in graphical form. Attribute usage matrix depicts the relation between tasks and data blocks in a matrix format. If q is the task that is being considered, and i is the data block referenced by q, then the usage matrix would have either a value 0 or 1. If q references the data block, 1 is the value of the mapping, 0 if not referenced. Mathematically, this can be represented as follows:

$$use(q,i) = \begin{cases} 1, & \text{if } q \text{ uses } i \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

(ii) **Attribute affinity matrix:**
For the calculation of the attribute affinity matrix, there are two inputs needed. One is the attribute usage

matrix. Other one is the application frequencies. Application frequencies denote the number of times a similar task is assigned to a particular VM or a site. From the HDAG, the data block task mapping table is prepared. Attribute affinity matrix can be calculated using the formula: ref is the number of access to data blocks block1 and block2 together for the execution of the task q. acc_j is the access frequency measure of both the data blocks at node j.

$$aff(b_1,b_2) = \sum_{\forall nodes j} ref(b_1,b_2) * acc_{(b_1,b_2)j} \tag{9}$$

(iii) **Data grouping matrix (DGM):**
Data grouping matrix (DGM) is a matrix indicating the relation between two data blocks. This relation between data blocks is obtained from the HDAG. Task to data blocks mapping is converted to a table with every data block and the task that uses this data block. So DGM would be an n*n matrix where n is the number of data blocks that are being considered. The data grouping matrix indicates how likely that one data block may be grouped to another data block.

(iv) **Clustered data grouping matrix (CDGM):**
Clustered data grouping matrix (CDGM) is a matrix derived from DGM. Bond energy algorithm (BEA) is used with DGM as the input to derive CDGM. BEA clusters highly associated data and it saves time by finding a suboptimal solution in less time. With this clustering, it is possible to derive a conclusion indicating which data should be placed together. When data blocks are more than the number of nodes, the process is repeated and a large number of clustering steps are involved.

(v) **Optimal Submatrix (OSM):**
Finally, high affinitive submatrices are identified as optimal submatrix (OSM) from the CDGM. These submatrices reveal the data groupings.

3.3. Optimal assignment

In this final step, the optimal locations for the data blocks are identified. By utilizing the gain matrix and the data groupings obtained using the previous steps, an optimal assignment is found. To find this, a suitable assignment problem as per the linear programming approach is identified and applied. Generally, a classical assignment problem deals with one-to-one mapping of tasks to resources while the objective is to minimize the total cost (or maximize the total gain) of assignment. The mathematical model for classical cost minimization problem is well-known:

$$Minimize \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} * X_{ij} \tag{10}$$

Subject to:

$$\sum_{i=1}^{n} X_{ij} = 1,\ j = 1,\ldots,n,$$

$$\sum_{j=1}^{n} X_{ij} = 1,\ i = 1,\ldots,n,$$

$$X_{ij} = 0\, or\, 1$$

In the above equation, X_{ij} is assigned to 1 if and only if task i is mapped to resource j, else, assigned as 0. Here, C_{ij} denotes the cost of assigning task i to resource j which is to be minimized. The constraints ensure that one-to-one cardinality is maintained. Many works with minor modifications to the basic assignment problem are discussed elaborately in the work by Pentico [13].

In this work, a suitable many-to-one assignment problem is identified and is termed as Grouped Assignment problem (GrAP). GrAP considers an m × n pregrouped gain matrix G = $[g_{ij}]$ and an integer k. k is the cardinality measure that is either m or n, whichever is minimum. The idea is to assign k grouped columns to k rows such that the sum of the corresponding gain is a maximum. Thereby, this problem becomes a many-to-one scenario where a group of highly affinitive data blocks are mapped to a particular node. Motivation for using the cardinality measure k is obtained from the work by Dell'Amico and Martello [14]. Formally,

$$Maximize \sum_{i=1}^{n}\sum_{j=1}^{m} G_{ij} * X_{ij} \tag{11}$$

Subject to:

$$\sum_{i=1}^{n} X_{ij} <= 1,\ j = 1, \ldots, m,$$

$$\sum_{j=1}^{m} X_{ij} <= 1,\ i = 1, \ldots, n,$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} X_{ij} = k$$

In (11), X_{ij} is assigned to 1 if and only if data block i is assigned to VM j that has maximum gain value. The implementation of the optimal assignment problem based on GrAP is given as Algorithm 1.

3.4. System architecture

The above three steps form the basis of designing the architecture of the proposed system. The system architecture for predicting data movements and placing data appropriately is given in Figure 1. It consists of the physical components and the virtual components. The physical components available in the data centers such as servers and storage devices are virtualized and then multiple virtual machines (VM) are configured. Virtual machines act as the servers that hold multiple processing elements. The virtual machine manager (VMM) is responsible for a multitude of data center-wide decisions such as data distribution, task scheduling, and load balancing. One of the virtual Machines is termed as the master node and the rest are considered to be the slave nodes. The master node acts as the cluster head and is configured to maintain data distribution information and job tracking information. The slave nodes are the nodes where the data blocks are available. The jobs are subdivided into tasks and scheduled to the data nodes for processing. During build time, the framework accepts random data placement and identifies data grouping and movement information. These observations are translated into data access graphs which are vital for successful prediction and placement. The prediction component has two interacting modules: i) gain matrix prediction module, to systematically scrutinize log information and compute the gain matrix for a given data distribution; ii) the change detector module, that monitors the accuracy of gain prediction and changes the likelihood estimation parameters based on service-level

Algorithm 1: Optimal assignment based on GrAP.

```
   Input: DblkPair[p][2] (p - no. of pairs)
   Gain Matrix GM[nBlk][nslave]
   (nSlave- no. of slave nodes; nblk - no. of data blocks of interest)
   Output: Data Placement matrix DP[p][nslave]
 1 Read GM
   /* Generate IntSet[p:no. of pairs][nSlaves:no of slave nodes]          */
   /* For each col in GM                                                  */
 2 for each col in GM[nBlk][nslave] do
 3     for each row in DblkPair[p][2] do
 4         if DblkPair[p][0]==GM.ColIndex then
               /* Build comma separated groups                            */
 5             $IntSet[nSlave][p]$.append(,)
 6             $IntSet[nSlave][p]$.append($GM[nSlave][nblk]$)
   /* Apply GrAP                                                          */
 7 for each row $R_i$ in IntSet do
 8     for each column $C_j$ from IntSet do
 9         $IntSet[nSlave][p]$.add(M[,][$C_i$])
           /* add all comma separated items in col $C_i$                  */
10 Solve using Hungarian method
11 Return DP matrix
```

agreements. The placement component consists of three modules: i) the data grouping module employs DRAW based data grouping method based on which the highly affinitive data block pairs are identified; ii) the candidate node identification module checks for nodes that are most suitable for a given set of data block pairs. Such potential nodes are marked as candidate nodes; iii) finally, for a given data block pair, the grouped assignment module, evaluates and finds the best node from the candidate set of nodes. The framework is connected as a loop so that the updated configuration is reflected in future placements.

3.4.1. Prediction component

(i) **Gain matrix prediction:**

The gain matrix is derived out of the logs maintained by the master node. It represents the relation between the data blocks and the virtual machine. Logs are analyzed and a high value of gain is given to the data block and VM pair that has been migrated less.

(ii) **Change detector module:**

This module constantly monitors changes in the workload patterns. These changes are to be refitted or updated in the system so that the changes are accounted for. The accuracy of the predictions severely affects the performance of the system. Hence, when the changes are not incorporated, it results in inaccurate gain matrices and hence a nonoptimal data placement. In this module, minor workload fluctuations are neglected whereas, major variations in the patterns are taken note of. These variations, when found to be persistent, are utilized for model redefinition. The change detection algorithm that is given in [11], is suitable for the proposed framework.

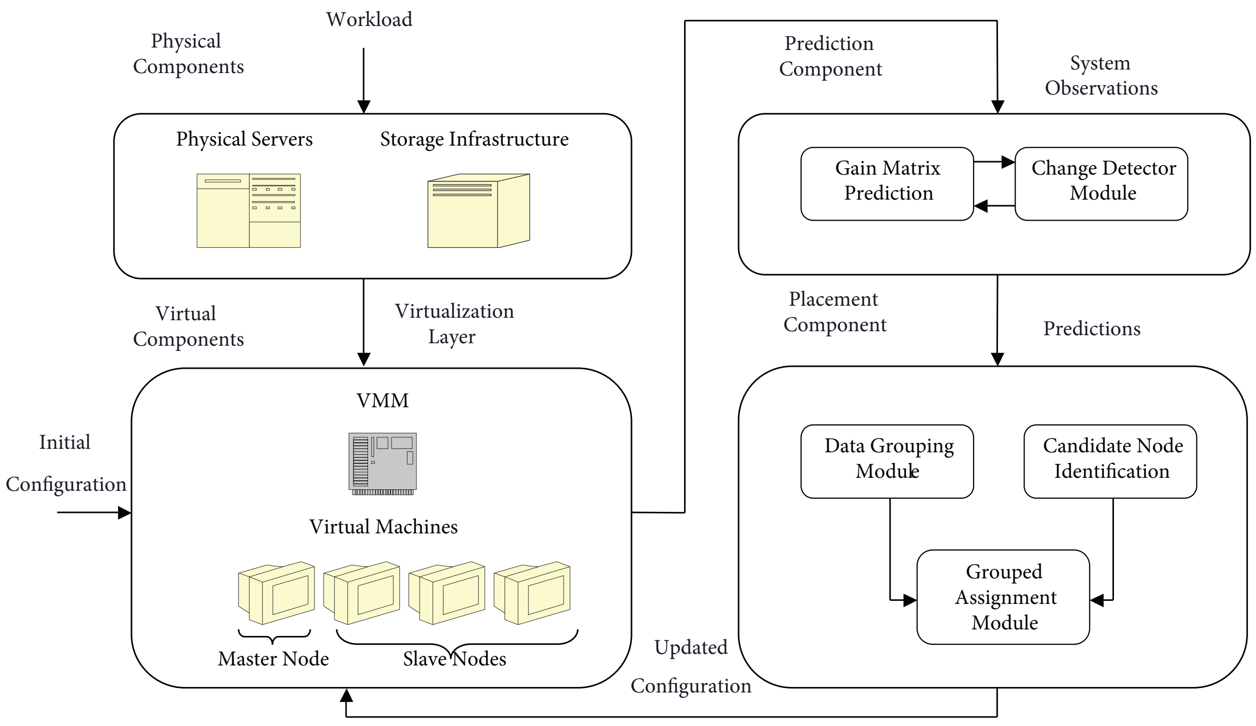

Figure 1. System architecture for predicting the movements and placing the datablocks.

3.4.2. Placement component

This component dynamically identifies data grouping and placement information for a time frame Δt based on the discrete results obtained from the prediction component. The data grouping information is identified using matrix-based mathematical computations. A candidate set of suitable VMs that could hold the data groups is found. Then, the optimal assignment algorithm based on GrAP is executed. This assigns data groups to nodes in a many-to-one fashion with the least possible future data movements.

(i) **Data grouping module:**
This module is designed as per the existing DRAW strategy. The log information from the master node reveals history of data access which forms the basis of developing history of data access graphs (HDAG). A set of matrix computations as discussed in Section 3.2 is done in this module.

(ii) **Candidate node identification:**
This module keeps track of nodes that are most suitable for a given set of data block pairs. Such potential nodes are marked as candidate nodes. The candidate set of nodes are separately maintained for a given data block. This acts like a cache that is looked up to quickly check if the node will be the best location for the data block. If there is not any candidate node for a given block, the complete node list is traversed to identify the best match. The implementation of this module is purely to speed up the lookup process.

(iii) **Grouped assignment module:**
After data block pairing, the gain matrix is checked for maximum value for both data blocks. Maximum among the sum of the gain matrix values for both data blocks gives the VM id where the data blocks have to be placed. The data blocks are then placed in the corresponding VM (as per Section 3.3).

4. Performance analysis

For performance analysis of the proposed system, a series of simulations followed by the execution of two real-time applications are conducted. For simulation purposes, the prediction and placement components are developed in the CloudSim framework. The workload with task lengths measured in terms of million instructions (MI) is generated following Poisson arrival distribution as in [15]. The performance metrics such as the response time and the migration counts are considered. The system performance with a varied number of processing nodes in an elastic architecture is evaluated. Also, the sensitivity of the system to a varied number of input data blocks is studied.

4.1. Simulation environment

The simulation parameters are tabulated in Table 1. To compare the performance of the proposed MADP, the existing DRAW system is implemented and the metrics are noted. The analysis details are explained in detail in the following sections.

Table 1. Parameters of simulation.

Cloud entity	Quantity
Data center	1
Hosts in DC	500
Host RAM capacity	16/32 GB
VM processing capacity	174/247/355 MIPS
VM	10 to 200 in steps of 10
VM RAM capacity	1920 MB
VM manager	Xen
Processing elements (PE)	4/8
PE processing capacity	90/120/150/225 MIPS
Task length/instructions	500000 to 200000000 MI
Data blocks	100 to 500 in steps of 50
Data nodes	1 VM per worker node
Namenode	1 VM for master node

4.1.1. Response time: performance in an elastic architecture

In Figure 2a, the number of VMs is varied from 10 to 200 in steps of 10 whose processing capacities are specified using million instructions per second (MIPS). The number of data blocks and the number of cloudlets (tasks) are maintained as 300 and 200 respectively. These constants are arrived based on multiple simulation runs conducted and based on general high performance computing (HPC) application requirements. When using DRAW, the input is provided without the gain matrix values. For MADP, the input is provided with the computed gain matrix values. Each cloudlet gets submitted to the data center broker. The service start time of the cloudlet is noted. After completion of the task, the finish time is noted. The time taken to service the cloudlet gives the response time. The average response time of the system is then identified by taking the statistical average of all the response times for a given set of cloudlets. As seen in Figure 2a, a hike in the response time of MADP was seen at a 70–90 range of VMs. Then, MADP attains a steady state after 120 VMs and is in decreasing order of response time. As we know, in cloud data centers, the number of VMs are typically very large. Hence, decreasing order of response time for a large number of VMs is valuable.

4.1.2. Response time: sensitivity to the number of data blocks

In Figure 2b, the number of data blocks is varied from 100 to 500 in steps of 50. The number of virtual machines and the number of tasks are maintained at 60 and 100, respectively. It is inferred that the average response time of MADP is significantly improved w.r.t that of DRAW in many instances. To account for instances when MADP is not at par with DRAW, it is further analyzed in terms of the number of migrations which is discussed in the following subsections 4.1.3 and 4.1.4.

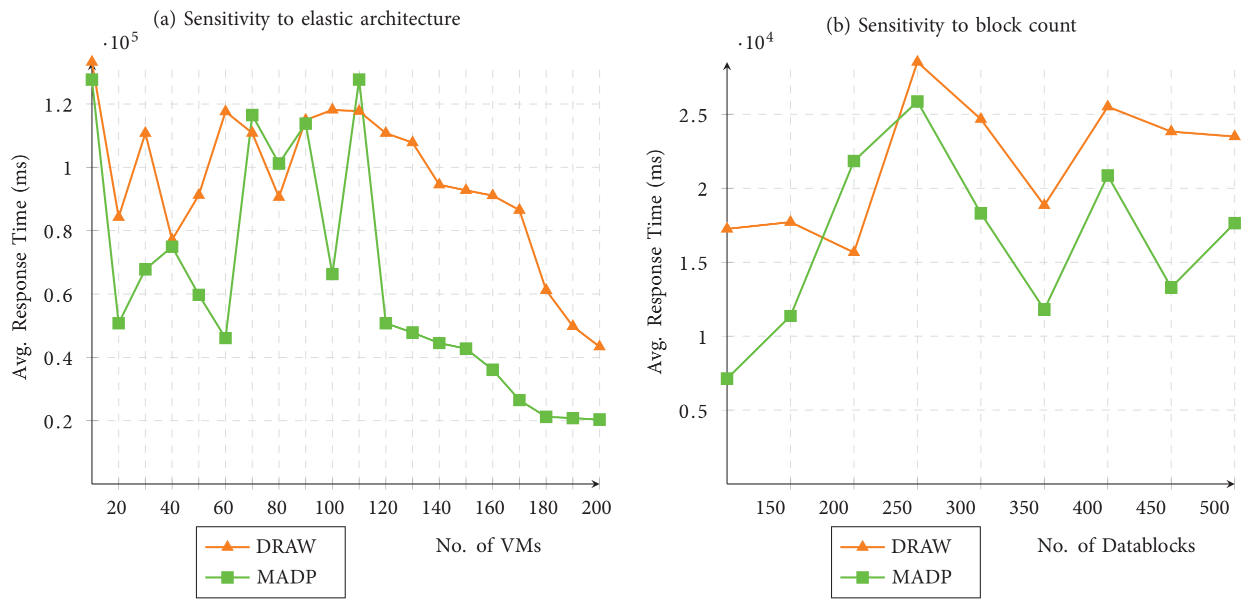

Figure 2. Response time comparison (a) when nDatablocks = 300, nTasks = 200, and the nVMs = {10, 20, 30,...,200}, (b) when nVMs = 60, nTasks = 600, and the nDatablocks = {100, 150, 200,...,500}.

4.1.3. Migration analysis: performance in an elastic architecture

It is important to quantify the performance improvement in HPC systems using a benchmarked index. Data availability is one such key metric [16]. Intuitively, data availability is indirectly proportional to the average number of data migrations when HPC applications are executed in cloud systems. Hence, we delve into analyzing the average number of migrations that happen during task execution. In Figure 3a, the numbers of VMs are varied from 10 to 100 in steps of 10 while the number of data blocks and tasks are maintained at 300 and 200, respectively. Migration is the difference between the number of data blocks required by the cloudlet and the number of data blocks available in the VM to which the task is assigned. The average number of migrations is noted and the graph is plotted. It is inferred from the graph that it tends to significantly decrease as the number of VMs utilized is greater than 120. Interestingly, this confers with the inference from Figure 2a.

4.1.4. Migration analysis: sensitivity to the number of data blocks

In Figure 3b, the number of data blocks is varied from 100 to 500 in steps of 50 while the available number of VMs were considered to be 60 and 100 tasks were initiated. The average number of migrations were then noted.

It is evident that, for MADP, the number of migrations is considerably lesser than that of DRAW placement. A trend analysis is conducted on the graph given in Figure 3b and the resultant linear trends for MADP and DRAW are obtained. R^2 corresponding to the DRAW curve is found to be 0.9924. Whereas, for MADP, R^2 is found to be 0.9935. It is well-known that the higher the R-squared value, the better the model fits the data. So, it is inferred that the linear trend best fits the MADP strategy and the forecast function will yield a similar trend for a higher number of data blocks. $y = 25.211x + 27.976$ and $y = 22.769x + 27.511$ are the line equations corresponding to the trendline of the DRAW curve and MADP, respectively. Further, the variance analysis on the number of data migrations is conducted on the available data from Figure 3b and the resultant variance percentages show that there is a minimum of 2.10% improvement due to MADP strategy when compared with DRAW. An improvement of up to 18.96% is obtained in terms of the number of data migrations during run-time. From the above graphs, it can be inferred that MADP performs well in many practically relevant instances. Further, to execute real-world applications and identify the performance improvements, an Apache Hadoop cluster is built.

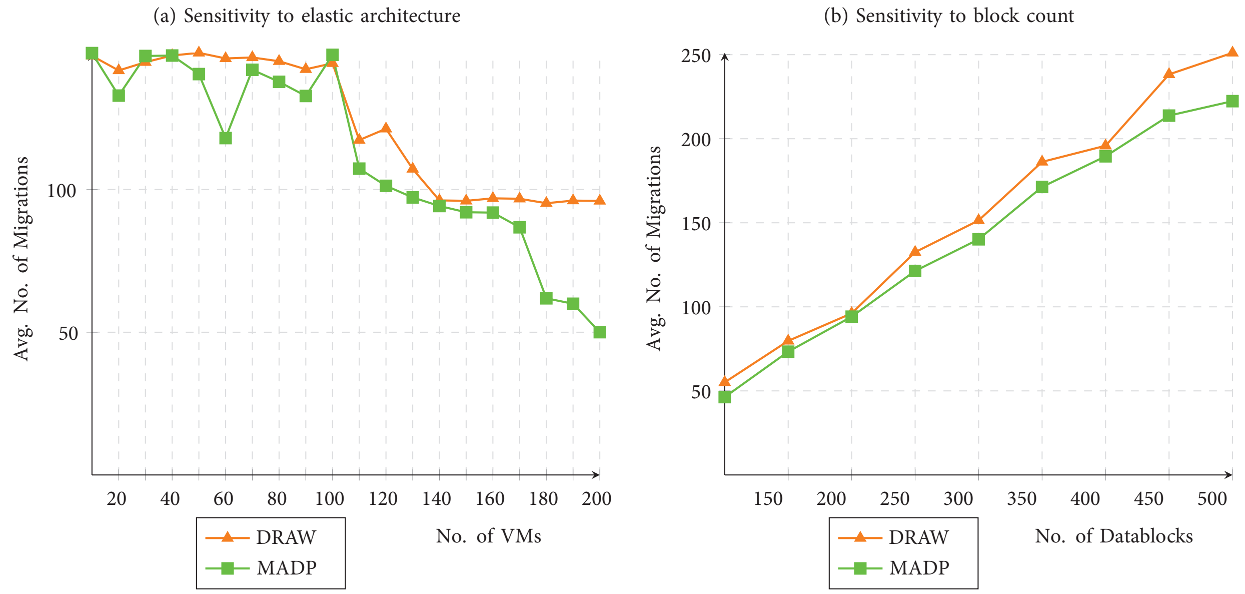

Figure 3. Migrations analysis (a) when nDatablocks = 300, nTasks = 200, and the nVMs = {10, 20, 30,...,200}, (b) when nVMs = 60, nTasks = 600, and the nDatablocks = {100, 150, 200,...,500}.

4.2. Hadoop environment

Commodity clusters are widely used for large scale scientific computations. In this work, Hadoop environment is created and scientific applications are executed in the cluster to evaluate the performance of the proposed scheme. Apache Hadoop is installed in the given set of nodes and thereby a storage and processing cluster is created. Public datasets such as literary works and weather forecasting and modeling datasets are used. The datasets were initially uploaded in the cluster using a bulk upload procedure. The applications to analyze literary work and model weather data were developed. These applications were executed on the cluster and the name node logs were analyzed. The logs revealed an HDAG formation which in turn was used to identify data block groups and also the gain matrix. The applications were executed to note down the vital metrics of the map-reduce tasks. Then, data blocks were wiped off the cluster. With the knowledge of HDAG and data movement history, the MAD-placement was employed to upload the datasets into the cluster. Then, the applications were executed to note down the metrics for the MADP strategy.

The complete details and analyses are given below: The Hadoop cluster was built with a single master node and 4 data nodes with Hadoop 1.x version installed. The cluster and node configurations are shown in the Table 2. The master node was configured to be the NameNode and the JobTracker, while the 4 slave nodes were configured as the DataNodes and TaskTrackers. The storage capacities of the data nodes were configured keeping in mind the huge dataset that is to be processed. Two widely used analytics applications such as literary data analytics and weather data analytics were implemented. Literary data sets [1] and weather data sets [2] were obtained from online repositories. Based on standard recommendations, the average block size was configured as 128 MB and the replication factor was configured as 3 in the single rack testbed. On the initial bulk upload of the two data sets, the data blocks were distributed randomly. At this stage, the workload is assigned and the applications were executed in the cluster. From the system logs, the gain matrix and the data groupings were generated, based on which the optimal data placements were identified.

Table 2. Apache Hadoop cluster setup.

Master node - NameNode/JobTracker		Slave nodes - DataNode/TaskTracker	
Model	Dell PowerEdge	Model	Dell PowerEdge
CPU cores	8	CPU cores	8
RAM speed	7.7 GBps	RAM Speed	3.7 GBps
Hard disk drive	688.5 GB	Hard disk drive	666.4 GB
Network interface	Intel Pro NIC	Network interface	Intel Pro NIC
Operating system	Ubuntu 10.4	Operating system	Ubuntu 10.4
Cluster network			
Switch model - BayStack Gigabit Switch			

4.2.1. Completion time analysis

Map-reduce task execution analysis of weather and literary data analytics application is shown in the Table 3. It is inferred that in both the applications, the number of map-reduce tasks initiated in the MADP based Hadoop framework is lesser when compared to that of DRAW. Further analyses showed improvements in the completion time of map-reduce executions. The graph of percentage of completion of map and reduce tasks for weather and literary applications are shown in Figures 4a and 4b, respectively.

Table 3. Comparison of map-reduce executions.

Analytics application	Placement strategy	Mappers	Reducers	Total tasks
(a) Weather data analytics	On DRAW	62	23	85
	On MADP	55	16	71
(b) Literary data analytics	On DRAW	52	25	77
	On MADP	55	14	69

[1]Free eBooks (2015). Project Gutenberg [online]. Website https://www.gutenberg.org/ [accessed 04 June 2020].

[2]NOAA (2015). National Centers for Environmental Information [online]. Website https://www.ncdc.noaa.gov/ [accessed 04 June 2020]

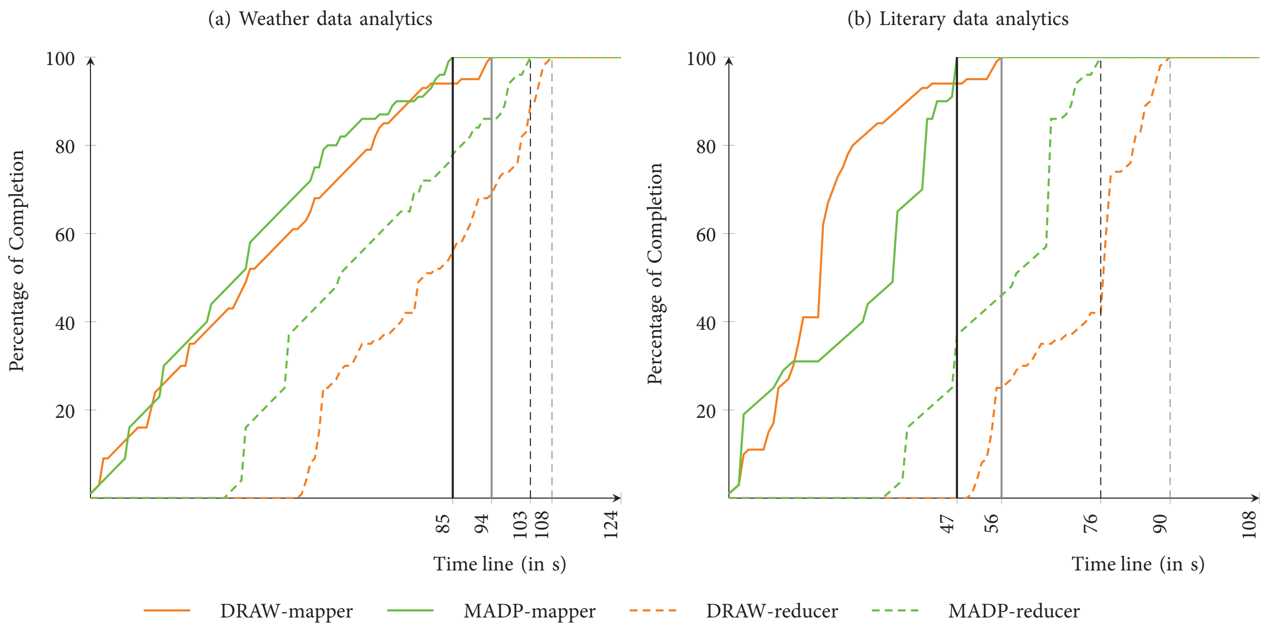

Figure 4. Completion time analysis of map-reduce applications on DRAW and MADP based Hadoop frameworks.

As time progresses, the percentage of completion of the map and reduce tasks are noted from the JobTracker module. The solid lines denote map tasks and dashed ones denote reduce tasks. A speed up in the execution of applications in MADP based framework is visible when compared to that of the DRAW framework. There is an improvement of 10.58% concerning mapper and 4.85% concerning reducer when weather analytics application is run on reorganized data as per MADP as compared with DRAW. For literary analytics, an improvement of 16.07% and 15.55% in the map and in the reduce phase was observed. This improvement is achieved by applying the knowledge of data movements and by optimally placing the data blocks. Based on these analyses, for larger datasets and larger processing infrastructures, it is expected that the benefit of applying the proposed approach will be high. It is noted that the initial random bulk upload of data incurs overhead. This overhead is acceptable as this methodology learns adaptively and improves over time.

5. Concluding remarks

In this work, a novel data placement strategy that aims to reduce the data movements during run-time is presented. This strategy termed as the MAD-placement is based on maximum likelihood estimation that predicts the likelihood of a data block being moved during application execution. MADP is demonstrated successfully by following a three-step process. First, MADP generates the gain matrix that is based on the likelihood estimation. Then, it captures run-time data block groupings from the system logs. Finally, by applying the many-to-one assignment problem, the data blocks are optimally placed in the data nodes. The system is evaluated under varying conditions and application scenarios. It is inferred that the proposed system performs well in terms of the response time and the number of block migrations. Currently, the load balance among the nodes on implementing the proposed strategy and also the possible effect of varying the replica factors is being analyzed. In the future, the proposed work can be extended by placing the data blocks not as pairs but in groups of more than two blocks per group. Further, the proposed strategy can be used beyond the initial application area, and it can be applied to estimate task migrations and thereby schedule tasks to resources with the least possible future migrations.

Acknowledgment & Conflict of interest

This research work of the first author was supported with a merit-based fellowship scheme entitled, 'Visvesvaraya PhD Scheme for Electronics and IT' (Department of Electronics and Information Technology-DeitY) by the Ministry of Electronics and Information Technology, India. The authors are grateful for the feedback from Dr. V. Sankaranarayanan, Former Professor and Director, Ramanujan Computing Centre, Anna University; Former Special Officer, Science and Technology, Govt. of Tamil Nadu and Dr. S. Jayaprakash, former professor, IIT Madras.
The authors have no conflict of interest to disclose.

References

[1] Kevin B, Satoshi M. Co-locating graph analytics and HPC applications. In: IEEE 2017 International Conference on Cluster Computing; Honolulu, HI, USA; 2017. pp. 659-660.

[2] Ke H, Li P, Guo S, Stojmenovic I. Aggregation on the fly: reducing traffic for big data in the cloud. IEEE Network 2015; 29 (5): 17-23. doi: 10.1109/MNET.2015.7293300

[3] Leskovec J, Rajaraman A, Ullman JD. Mining of Massive Datasets. Palo Alto, CA, USA: Cambridge University Press, 2014.

[4] 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. doi: 10.1109/TMAG.2013.2251613

[5] Gorla N, Zhang K. Deriving program physical structures using bond energy algorithm. In: ASPEC 1999 Proceedings Sixth Asia Pacific Software Engineering Conference; Takamatsu, Kagawa, Japan; 1999. pp. 359-366.

[6] Hao X, Jin P, Yue L. Efficient storage of multi-sensor object-tracking data. IEEE Transactions on Parallel and Distributed Systems 2016; 27 (10): 2881-2894. doi: 10.1109/TPDS.2015.2511735

[7] Mahdi E, Aravind M, Andrey K, Shiyong L. BDAP : a big data placement strategy for cloud based scientific workflows. In: 2015 IEEE First International Conference on Big Data Computing Service and Applications; Redwood City, CA, USA; 2015. pp. 105-114.

[8] Xu Q, Xu Z, Wang T. A data-placement strategy based on genetic algorithm in cloud computing. International Journal of Intelligence Science 2015; 5 (3): 145-157. doi: 10.4236/ijis.2015.53013

[9] Zhang J, Chen J, Luo J, Song A. Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers. Tsinghua Science and Technology 2016; 21 (5): 471-481. doi: 10.1109/TST.2016.7590316

[10] Runqun X, Junzhou L, Fang D. SLDP : a novel data placement strategy for large-scale heterogeneous hadoop cluster. In: Second International Conference on Advanced Cloud and Big Data; Huangshan, Anhui, China; 2014. pp. 9-17.

[11] Juan T, Daniel H, Javier B, Florin I, Jesus C. CONDESA : a framework for controlling data distribution on elastic server architectures. IEEE Transactions on Parallel and Distributed Systems 2014; 25 (8): 2010-2018. doi: 10.1109/TPDS.2013.197

[12] Myung J. Tutorial on maximum likelihood estimation. Journal of Mathematical Psychology 2003; 47: 90-100. doi: 10.1016/S0022-2496(02)00028-7

[13] David P. Assignment problems : a golden anniversary survey. European Journal of Operational Research 2007; 176: 774-793. doi: 10.1016/j.ejor.2005.09.014

[14] Mauro D, Silvano M. The k-cardinality assignment problem. Discrete Applied Mathematics 1997; 76: 103-121. doi: PZZSO166-218X(97)00120-5

[15] Atmaca T, Begin T, Brandwajn A, Castel-Taleb H. Performance evaluation of cloud computing centers with general arrivals and service. IEEE Transactions on Parallel and Distributed Systems 2016; 27 (8): 2341-2348. doi: 10.1109/TPDS.2015.2499749

[16] Yonghong H, Xuebin C, Debbi C, David K, David Y. A quantitative index for measuring the development of supercomputing. Concurrency and Computation: Practice and Experience 2015; 27 (17): 4685-4703. doi: 10.1002/cpe.3451