
A novel entropy-based dynamic data placement strategy for data intensive applications in Hadoop clusters

K. Hemant Kumar Reddy*

Computer Science and Engineering,
National Institute of Science and Technology,
Berhampur, India

Email: khemant.reddy@gmail.com

*Corresponding author

Vishal Pandey

David Eccles School of Business,
University of Utah,
Utah, USA
Email: vishalpandey92@live.com

Diptendu Sinha Roy

National Institute of Technology,
Bijni Complex, Laitumkhrah,
Shillong 793003, Meghalaya, India
Email: diptendu.sr@nitm.ac.in

Abstract: In the last decade, efficient data analysis of data-intensive applications has become an important research issue. Hadoop is the most widely used platform for data intensive application. However, majority of data placement strategies attempt placing related-data close to each other for faster access without considering new datasets, generated or for different MapReduce jobs. This paper deals with improving the map-reduce performance over multi-cluster datasets by means of a novel-entropy-based data placement strategy (EDPS) in three-phases. K-means clustering strategy is employed to extract dependencies among different datasets and group them into data-groups. Then these data-groups are placed in different datacenters while considering heterogeneity. Finally, an entropy-based grouping of the newly generated datasets where these datasets are grouped with most similar existing cluster based on their relative entropy. The experimental results show efficacy of the proposed three-fold dynamic grouping and data placement policy, which significantly reduces the time and improve Hadoop performance

Keywords: dynamic data placement strategy; Hadoop clusters; MapReduce; k-means clustering; entropy.

Reference to this paper should be made as follows: Reddy, K.H.K., Pandey, V. and Roy, D.S. (2019) ‘A novel entropy-based dynamic data placement strategy for data intensive applications in Hadoop clusters’, *Int. J. Big Data Intelligence*, Vol. 6, No. 1, pp.20–37.

Biographical notes: K. Hemant Kumar Reddy received his MTech and PhD degrees from the Berhampur University in 2008 and 2014 respectively. He is currently with the National Institute of Science and Technology, Berhampur, India as an Assistant Professor. His research interests include distributed, grid and cloud computing, service oriented architectures, and optimisation in engineering.

Vishal Pandey received his BTech in Information Technology from the Maharaja Agrasen Institute of Technology, Guru Gobind Singh Indraprastha University, India in 2016 and currently pursuing his Master’s of Science in Information Systems from the David Eccles School of Business, University of Utah. His current research interests include big data analysis, data processing over geo-distributed data centres, image processing, data mining through visual analytics and geographical information systems. He was the recipient of the Morgan Stanley Fellowship for Academic and Research Excellence at the University of Utah.

Diptendu Sinha Roy received his BTech degree from the Kalyani University in 2003 and subsequently his MTech and PhD degrees from the Birla Institute of Technology, Mesra, India in 2005 and 2010 respectively. He is currently with the National Institute Technology, Meghalaya, India as an Associate Professor. His research interests include distributed and grid computing, software reliability, optimisation in engineering. He also works towards design and analysis of distributed infrastructure of power systems.

1 Introduction

Recent times have seen the proliferation of data intensive applications such as scientific workflows, search engines, bioinformatics, online portals, etc. This explosion of data from a variety of sources has led to the requirement of highly efficient big data analysis tools. In 2004, Google proposed the popular MapReduce framework for parallel processing of big data in high performance clusters that could process approximately twenty petabytes of data per day (Dean and Ghemawat, 2008) that was inconceivable a decade ago. Apache Hadoop (Appache Software Foundation, 2016), Google's open-source variety of MapReduce, is extensively employed to provision application demanding short response times. Hadoop has been employed to process huge volumes of data in parallel by leading internet service providers for their services like Amazon (2016) and Facebook (Borthakur et al., 2011), in addition to Yahoo! and Google.

MapReduce framework cut downs the obscurity of processing data distributed among multiple clusters by allowing even neophyte users to create map reduce functions by providing easy to use APIs for parallel data processing. MapReduce itself handles intricacies of such operations over many clusters, even with heterogeneous environments (Lee et al., 2014). The MapReduce paradigm exhibits improved performance and added advantages in comparison to traditional parallel computing techniques. First, MapReduce models are highly scalable in that even with increased number of cluster nodes and data sizes, the system still works efficiently without much intervention. Second, the MapReduce model is highly fault tolerant in that it automatically manages faults and failures as and when system components go down. Additionally, the MapReduce paradigm is remarkably simple to work with.

In the Hadoop architecture, three aspects, namely, assigning tasks, placing data among nodes in a cluster and run-time data movements are crucial (Reddy et al., 2015). The main contribution of this paper lies in intelligent data placement that can affect Hadoop's performance significantly. In MapReduce approach, computations and data when collocated in the same node can bring out significant performance benefits (Reddy et al., 2015). However, for a variety of reasons such as restrictive data volume, proprietary considerations moving data among geographically dispersed datasets can be prohibitively costly (Zaharia et al., 2008). Therefore we need efficient data placement policies to reduce cost by minimising runtime data movements and data overheads.

In default Hadoop environment, Hadoop distributed file systems (HDFS), the data are partitioned into numerous blocks of same size and are distributed to each node equally to balance the load of each node. Such a data placement strategy can achieve load balance, which is highly efficient and practical for a homogeneous cluster node environment. However, in a practical scenario, heterogeneity of nodes is more the norm than an exception. If MapReduce applications are executed on such a scenario with Hadoop default strategy with all nodes having a balanced processing responsibility, a node having higher computing capacity with local data blocks completes executing tasks assigned to it faster than others. Due to adaptive load balancing strategy, such a node processes tasks of nonlocal data blocks. This incurs additional overhead to move data blocks from a slower node to a faster node for execution. To this end, we envision a dynamic data placement policy to reduce the runtime data movement overhead.

There have been some previous attempts to improve the performance of MapReduce applications by making the system aware of the heterogeneity in the data centres (Lee et al., 2014; Reddy et al., Reddy et al., 2015; Zaharia et al., 2008; Wang and Li, 2016) and by adopting a data placement strategy which places highly cohesive datasets together in a single data centre to reduce data overheads (Yuan et al., 2010). Although these attempts improved the performance of MapReduce applications to some extents, but they had a major drawback of not accounting for the data placement for the newly generated datasets and for the same reason there implementation has been very restricted.

In this paper we propose a three phase dynamic data placement strategy named to reduce the runtime data movements overhead in MapReduce applications. In the first phase, we cluster those data sets which are highly depended on each other. Dependencies are evaluated based on correlations among various tasks in terms of their data block requirements by employing the bond energy algorithm (BEA) (Xie et al., 2010). In the second phase, we place these groups of datasets in different data centres considering the heterogeneity of the cloud to balance data loads among different data centres to reduce data overheads cost during task execution. Finally, the third step aims at proper placement of newly generated or arrival data. This is achieved by using the concept of entropy, which defines the unrelatedness and disorder in any group or cluster. We calculate the entropy metric in different data groups related to the newly generated data when the newly generated data is placed in that data group and thereby we attempt to minimise the entropy. In the proposed three phase strategy

solution, the first two steps are extensions of the works of Lee et al. (2014), Yuan et al. (2010) and Reddy et al. (2015). for efficient data placement of existing datasets whereas the third step accounting for the placement of newly generated datasets is the novelty of this research work. Such a multi-step data placement strategy as proposed in this research work is the first attempt in current researches to improve the performance of map reduce applications in Hadoop ecosystem.

The remainder of the paper is organised as follows: Section 2 presents the related work with intuitive reasoning of our proposal. Section 3 provides a brief account of the map Reduce programming model and its Hadoop implementation. Section 4 presents the principles involved in designing the proposed three phase dynamic data placement strategy. It presents the algorithm with examples for easy illustration. Section 5 demonstrates the simulation results and evaluation. Finally, Section 7 summarises the conclusions of the research.

2 Related work

Hadoop assumes that the computing capacity of each node attached to a virtualised data centre is identical, i.e., each node is assigned the same amount of work. However in real-world scenarios, clusters often work in heterogeneous environments where different nodes attached to cloud has different computing and data storage capacities. If such heterogeneous clouds use the Hadoop's default data placement (HDDP) strategy and distributes the workload evenly across each node then the overall completion time and hence performance of Hadoop falls. Since nodes with higher computation capacities complete their assigned tasks sooner than their low-end counterparts, hence more incomplete jobs will be assigned by the master node to idle, high-end node. But such nodes might not have the data necessary to complete the jobs, leading to runtime data movement among clusters. We aim to minimise such data movements. Zaharia et al. (2008) proposed a heterogeneity-aware scheme for improved performance of Hadoop system. Also there has been attempts to schedule jobs to virtualised clusters considering node heterogeneity by assigning jobs to nodes in proportion to their capacities, computing ration being the metric for deciding the fraction to be assigned to every node (Xie et al., 2010; Lee et al., 2014). Very recently, there has been attempt of data placement and task scheduling in MapReduce environments considering job deadlines, data locality and adaptive heartbeats (Wang and Li, 2016).

Another modification to improve map reduce performance was introduced in MapReduce online (Condie et al., 2010), which allowed map reduce to start executing jobs before the data has been totally materialised, thus proposing time-multiplexed execution mapper and reducer executions. For instance, it allows the reducers to start before all the mappers have completed and are available for

use. P-aware (Zhou et al., 2016) presented a novel resource pooling strategy for aggregating data centre resources that applied a multi-dimensional bin packing approach for achieving lower energy consumption and reduced resource contention (Zhou et al., 2016). In Patil and Chaudhary (2013), presented a rack aware scheduling in High Performance Computing data centres that employed intelligent switching of idle rack nodes for improved power consumption.

Chang et al. (2011) introduced an approximation algorithm that organises all map reduce jobs such that overall job completion time is minimised. Reddy et al. (Reddy et al., 2017) introduced a coordinated minimisation approach that organises all map reduce jobs such that over price and cost is minimised in user and provider point of view respectively. Certain scientific workflows had been very successfully deployed in virtualised environments. However, for such systems data movements at runtime have posed as very challenging overhead. Though traditional data management systems had their own data placement strategies but the objective had never been on reducing data movements. For instance, Kepler (Baru et al., 1998) uses the storage resource broker (SRB) system, an actor oriented actor-oriented data model in grid environment for large datasets. Gridbus (Buyya and Venugopal, 2004; Srikumar et al., 2004) uses a service-broker system where whole data is considered equally vital for applications. However these data management systems disregard the dependencies among datasets either at build time or at run time and also they did not aimed to reduce the data movements. However, locality property among datasets in terms of their usage by applications has been a well-known fact and has been termed as dependency (Yuan et al., 2010), interest locality (Yuan et al., 2010) and so forth.

Hsu et al. (2015) presented a method to improve MapReduce execution in heterogeneous environments using dynamic partitioning of data before the map phase and by using virtual machine mapping in the reduce phase in order to maximise resource utilisation. In Hsu et al. (2015), De Souza and De Araújo (2017) introduced an estimation model called HCEm model, which is designed to estimate the size of a cluster running Hadoop, in a given timeframe on cloud environments. The HCEm consists of a light optimisation layer for MapReduce jobs and a model to estimate the size of a Hadoop cluster. A comparative study of HCEm using similar applications and workloads in two production Hadoop clusters, the Amazon elastic MapReduce and a private cloud of a large financial company. Hagos et al. (2016) presented a study report on broadly useful open technologies and methodologies for applying an OpenFlow-enabled SDN to scalable cloud-based services and a variety of diverse applications. As a novelty, author(s) introduced new research challenges in the design and implementation of advanced techniques for bringing an SDN-enabled components and big data applications into a cloud environment in a dynamic setting. Some of these challenges become pressing concerns to

cloud providers when managing virtual networks and data centres, while others complicate the development and deployment of cloud-hosted applications from the perspective of developers and end users.

Recently Xiong et al. (2015) presented a wholesome data placement strategy in Hadoop considering node heterogeneity, data hotness-based replication policy and a snakelike data placement (SLDP) by integrating the former two policies. SLDP was tested on real data intensive applications and had an energy efficient extension. However, there are major differences in between SLDP and our proposed EDPS. EDPS does not consider power consumption, but can handle evolving data sets, that are very common for many real time applications.

Reddy, et al. (2015) also proposes a data placement strategy for reducing runtime data movements by using a dependency-based strategy that is developed by observing the data usage pattern of map reduce applications. However, such an attempt is static in the sense that they are not very suitable for use in systems that has an evolving data. Many contemporary applications spit out huge volume of data in a real time fashion. The proposed entropy-based data placement strategy (EDPS) is dynamic in the sense that it can work on such continuously arriving and evolving datasets. The essence of the entropy-based scheme lies in computing expected entropy, a measure for dissimilarity of MapReduce jobs and their data usage patterns in terms of data blocks stored in HDFS and finally placing new data among clusters such that entropy is reduced. Additionally, the DPPACS framework in Reddy et al. (2015) did not consider heterogeneity of Hadoop clusters, whereas the present work addresses this issue and is thus dynamic in that sense also. Entropy-based clustering approach has been employed successfully to other engineering fields such as archaeology, knowledge discovery among others in Xiong (2015), bioinformatics (Barbará et al., 2002), etc. The proposed EDPS works by checking changes in expected entropy of clusters once newly generated datasets are placed in each cluster and at the end the cluster with the least entropy is selected and the dataset is placed therein.

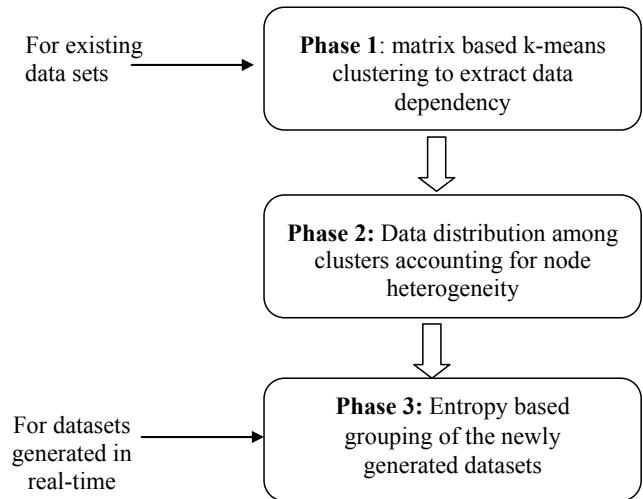
3 A brief overview of the MapReduce paradigm and Hadoop

Efficient data placement in virtualised environments has been a fledgling area of research within cloud computing and big data research circles. Among the varied data management paradigms in recent past, the Google FILE SYSTEM (Lee et al., 2014) and Hadoop (Ghemawat et al., 2004) have gained attention, the latter gaining the status of the de facto standard for data-intensive analysis and processing. Google's MapReduce framework (Dean and Ghemawat, 2008). The fundamental motivation is to provide the users with a simple and easy to use interface for working with big data. The following subsections provide a brief overview of the MapReduce framework and the Hadoop environment.

3.1 The MapReduce programming model

MapReduce, proposed by Google in 2004, is a programming model used in clusters with numerous nodes to manage large amount on data in parallel. In the MapReduce programming model, an application is termed as job which are assigned by a master and is divided into two parts the 'map task' which runs the map function and the 'reduce task' which runs the reduce function. The map function processes the input data and generates intermediate $< \text{key}, \text{value} >$ pairs. Based on these key-value pairs, the reducer functions merges, sorts, combine and returns the final result. Figure 1 depicts a schematic representation of how the map reduce model functions.

Figure 1 Three phase of dynamic data placement framework



The MapReduce framework basically employs the idea of divide and conquers technique. As it distributes a large amount of data to many nodes to perform parallel processing of data, it improves performance by reducing execution time of tasks. The most appealing aspect of this MapReduce framework lies in its simplicity, as it hides the complex details of parallel computing from the programmer and the programmer just needs to use available APIs. In past years, the Map Reduce model has been employed in various data intensive web applications from Google, Facebook (Borthakur et al., 2011), Yahoo!, even on graphics processing unit applications (GPU) such as in Hadoop (2016).

3.2 Hadoop and HDFS

Hadoop is the utmost popular open-source implementation of the MapReduce model. The Hadoop framework consists of two major components, namely the MapReduce language and the HDFS.

Hadoop's runtime environment along with HDFS coordinates all low-level issues from users providing a user friendly interface that is reliable by dint of its configurable replication option. It consists of a master node and a slave node. A master node controls a group of slave nodes where map and reduce tasks are run in parallel. The master node

assigns tasks to the slave node as and when required, provided the slave nodes have empty task slots. Whenever the HDFS receives any input, it splits the input file into evenly spaced fragments and these are distributed among a group of slaves for further map reduce processing. Unlike other traditional distributed file systems, HDFS aims to provide high throughput and is highly reliable because each file fragment stored on one node is replicated and copies of it are stored on other nodes for fault tolerance purposes. By default HDFS follows a three way replication system that ensures that a file residing in HDFS is always intact at three different node allocations, although the degree of replication is configurable.

3.3 File system related issues

Now days the general approach to deal with data is in terms of Giga bytes and Petabytes and handling such large data is general file system is bit difficult and required more human effort. Regarding the size and amount of data can handle in existing file system in the form of parallel, is termed as parallel file system or general parallel file system (GPFS). GPFS is a high-performance clustered file system developed by IBM. It can be deployed in shared-disk or shared-nothing distributed parallel modes. Even in case of GPFS the size and number of disk structures are limited (He et al., 2008; Schmuck and Haskin, 2002; IBM, 2000; http://www.ibm.com/servers/eserver/pseries/news/pressreleases/2000/jun/asci_white.html). The major issues related parallel file systems are data striping, allocation, pre-fetch and write-behind. To support efficient file name lookup in very large directories (large directory support, <http://www.llnl.gov/asci/platforms/white/home/>) and extensible hashing (VERITAS Software Corporation, 2000) to organise directory entries within a directory (Fagin et al., 1979). In a large file system it is not feasible to run a file system check (fsck) to verify/restore file system consistency each time the file system is mounted or every time that one of the nodes in a cluster goes down (logging and recovery). In addition to these issues security issues are more important along with issues like file systems maintain many data structures, all data structures are cached for better performance, several file system operations update multiple data structures and dealing with mechanical latencies (like scheduling algorithms, layout, caches, RAID)

4 Dynamic data placement strategy

This section presents the proposed three phases dynamic data placement strategies to improve the efficiency of map reduce operations for big datasets. Figure 2 summarises the proposed threefold dynamic data placement strategy. The following subsections detail the working principle of each of the three phases.

4.1 Data grouping phase

In virtualised environments data movement from one data centre to another is expensive from the perspective of task scheduling (Ghemawat et al., 2003). Hence our basic strategy in this phase is to place data in virtualised environments in such a way that the computation and data collocation is maximised. First of all, dependencies among blocks in datasets are created based on data block requirements by map tasks. This is done by employing a BEA (Schmuck et al., 1999).

We denote the set containing all the data sets as D and the set containing all the tasks as T . For every $D_j \in D$ we have three attributes $\langle T_i, S_i, L_i \rangle$, where $T_j \in T$, S_i denotes the size of the dataset and L_i denotes the location of the data set. Furthermore, data dependency between two data sets d_i and d_j is denoted by d_{ij} and this dependency value depends on the number of tasks common between d_i and d_j , as shown in equation (1).

$$d_{i,j} = \text{count}(T_i \cap T_j) \quad (1)$$

Therefore we can say that the more number of task that datasets share in common DM_{ij} the higher dependency they have. Now, for the existing data we create a $n \times n$ dependency matrix (DM) where the value of DM_{ij} is drawn from (2).

$$DM_{i,j} = d_{i,j} = \text{count}(T_i \cap T_j) \quad (2)$$

Next, BEA is employed to convert the DM into a cluster matrix (CM) as per (3). BEA was proposed in 1972 and since then has been widely been used for vertical partition of large tables in distributed database systems (Hagos, 2016). In CM the items with similar values are grouped together.

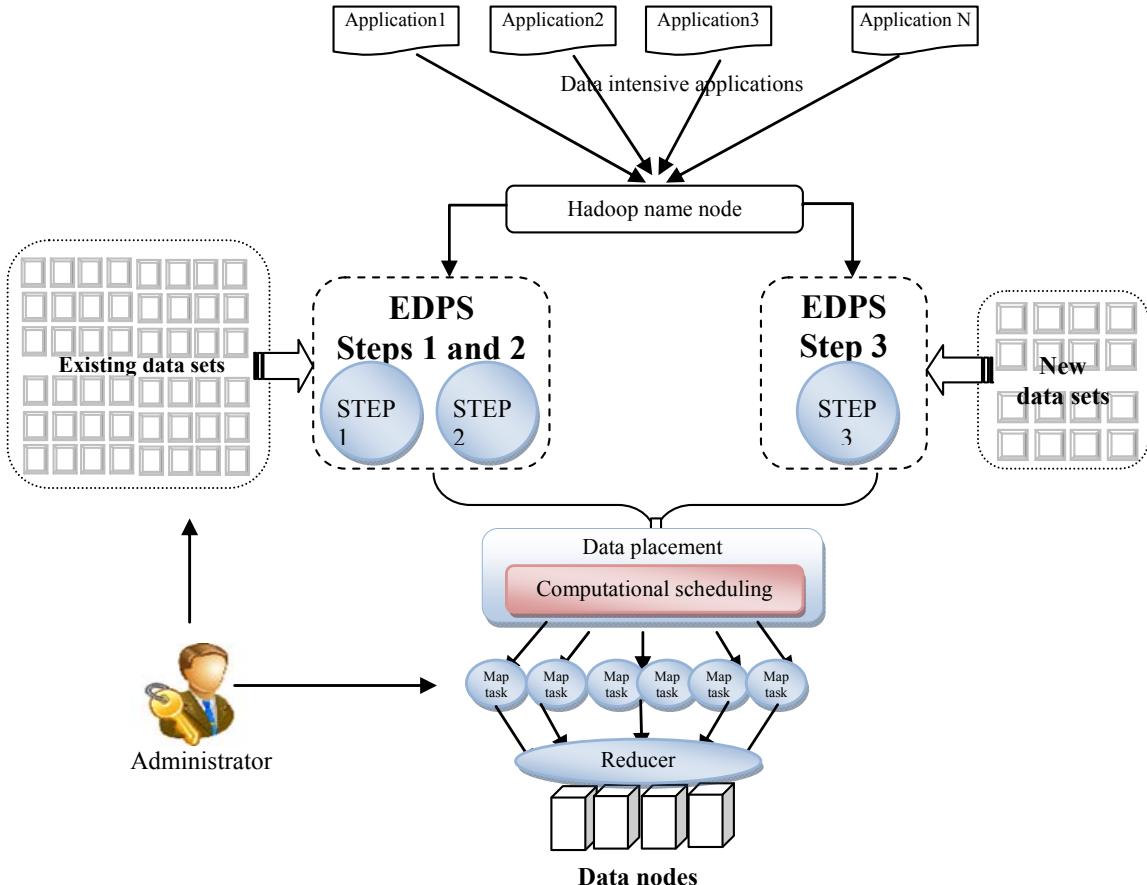
$$CM_{i,j} = BEA(DM_{i,j}) \quad (3)$$

After BEA transformation all the datasets are clustered into respective data groups denoted by DG_i as per their dependencies, irrespective of their initial location. The data grouping phase algorithm presented in this section is summarised in Algorithm 1.

Algorithm 1 Data grouping phase algorithm

<i>Input</i>	$D: \text{set of data sets } \{d_1, d_2, d_3, \dots, d_n\}$
<i>DC</i>	$\text{set of data centres } dc_1, dc_2, dc_3, \dots, dc_n$
<i>Output</i>	$'K' \text{ is the set of data groups } DG_1, DG_2, DG_3, \dots, DG_n$

- 1 Generate the list of all tasks that uses $d_1, d_2, d_3, \dots, d_n$ individually
 - 2 Calculate dependency $d_{i,j} = \text{count}(T_i \cap T_j)$
 - 3 Construct $DM = d_{i,j} = \text{count}(T_i \cap T_j)$
 - 4 $CM = BEA(DM);$
 - 5 return K set of DG_n
 - 6 End.
-

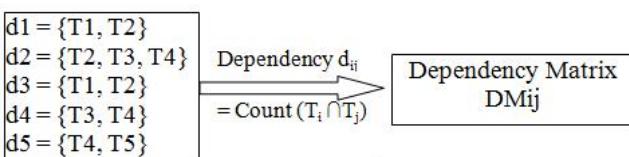
Figure 2 Overview of Hadoop MapReduce using EDPs (see online version for colours)

4.1.1 Illustrative example

For instance, let us consider the following case where there are five data blocks d_1, d_2, d_3, d_4, d_5 and five tasks that use these five data blocks. Task T_1 uses data blocks d_1 and d_3 , T_2 uses data blocks d_1, d_2 and d_3 , T_3 uses data blocks d_2 and d_4 , T_4 uses data blocks d_2, d_4 and d_5 and T_5 uses data block d_5 . Table 1 depicts the same. The data dependency can be extracted from Table 1. We can see that both tasks T_1 and T_2 use the same data blocks d_1 and tasks T_2, T_3 and T_4 use the same data block d_2 . Such dependencies can be tabulated as shown in Figure 3 and thus DM can be formed.

Table 1 Tasks with their required data blocks

$T_1 = \{d_1, d_3\}$
$T_2 = \{d_1, d_2, d_3\}$
$T_3 = \{d_2, d_4\}$
$T_4 = \{d_2, d_4, d_5\}$
$T_5 = \{d_5\}$

Figure 3 Data dependency matrix**Figure 4** Conversion data dependency matrix

$\begin{array}{cccccc} & d_1 & d_2 & d_3 & d_4 & d_5 \\ d_1 & 2 & 1 & 2 & 0 & 0 \\ d_2 & 1 & 3 & 1 & 2 & 1 \\ d_3 & 2 & 1 & 2 & 0 & 0 \\ d_4 & 0 & 2 & 0 & 2 & 1 \\ d_5 & 0 & 1 & 0 & 1 & 2 \end{array}$	BEA	$\begin{array}{cccccc} & d_1 & d_2 & d_3 & d_4 & d_5 \\ d_1 & 2 & 2 & 1 & 0 & 0 \\ d_2 & 2 & 2 & 1 & 0 & 0 \\ d_3 & 1 & 1 & 3 & 2 & 1 \\ d_4 & 0 & 0 & 2 & 2 & 1 \\ d_5 & 0 & 0 & 1 & 1 & 2 \end{array}$
DM = Dependency Matrix	CM = Cluster Matrix	

Figure 4 depicts an example of conversion from DM to CM. DM presents the number of tasks that access data set d_i and d_j . For example, the dependency element at (1, 1) of the DM matrix is 2, which indicates that data set d_1 is accessed by two tasks (T_1 and T_2). Similarly other entries of dependency values can be understood.

Table 2 Evaluating computation ratios for nodes in heterogeneous cluster

Task type	Node	Total task completion time		Computing ratio
		App1	App2	
App1	n1	10		3
	n2	15		2
	n3	30		1
App2	n1	4		2
	n2	12		1.5
	n3	16		0.5

As discussed earlier, the CM groups are highly dependent datasets in a particular group. The CM is formed from the DM with the help of BEA algorithm. The detail of the algorithm is presented in Algorithm 1. Figure 4 depicts an example of the extraction of data CM from the data DM. Once the CM is formed and clustered data groups (DG_1 , DG_2 , DG_3) are identified in the CM, then these data groups are distributed among different nodes in virtualised clusters considering the heterogeneity of nodes.

4.2 Data group placement considering the heterogeneity of nodes in virtualised environments

As discussed earlier, in a heterogeneous virtualised environment, the computing capacity of the attached nodes varies. Therefore the overhead of moving unprocessed data from slower nodes to faster nodes becomes a critical overhead and affects Hadoop's performance (Yuan et al., 2010). Therefore in this phase we place the data groups that are extracted in the previous phase and distribute them among different cluster nodes taking their inherent heterogeneity into account. Section 4.2.1 elaborates how computing ratios are calculated and Section 4.2.2 describes the approach to redistribute the data sets among virtualised cluster nodes according to their computing ratios.

4.2.1 Evaluating computing ratios

The computing ratio of each node attached to a heterogeneous cluster is based on the average execution time of a specific task in that node. Computation ratio is used to determine the data blocks allocation ratio in nodes when the data is written into the HDFS and is used to determine whether the data blocks must be reallocated when the job is executed. The computation ratio table records the types of jobs and the computing capacity ratio of each node. The computing capacity of each node is based on the average execution time of a single task in that node. The *namenode* (master in Hadoop's runtime environment) computes the computing ratio of each node according to the task execution time returned by the *heartbeat* message of each data node.

Let us assume that we have three heterogeneous nodes n1, n2 and n3 among which node n1 is the fastest then B and C being the slowest one. Let us also assume that node n1 takes 10 seconds, node n2 takes 20 seconds and node n3 takes 30 seconds to complete the same task. Now, based on the least common multiple principle, the computing ratio for node n1 is 3, 2 for node n2 and 1 for node n3 as shown in Table 1, for two applications. We use two applications and refer to them as app1 and app 2.

4.2.2 Placement of data groups as per the computing ratios

Whenever a job is assigned to the HDFS, the *namenode* first checks whether this job has been executed earlier or not. If so, then the newly written data would be allocated to each

node in accordance with their computing ratio by the *namenode*. If the *namenode* does not have the record for that task then the data blocks will either be equally distributed among all the nodes or some decided policy and a new record for this type of job is appended in the *namenode*.

From phase 1 of our data placement strategy we obtained the three data groups namely $\{d_1, d_3\}$, $\{d_2, d_4\}$, $\{d_5\}$. App1 uses these five data sets over three heterogeneous nodes whose computation ratios is shown in Table 3. Similarly, we get the computing ratio for app2.

Algorithm 2 Data placement algorithm

<i>Input</i>	<i>Jtype: Job Type, Datasets, DSize: Dataset Size, DC: Set of data centers $dc_1, dc_2, \dots dc_n$,</i>
<i>Output</i>	<i>DG: Set of data groups $DG_1, DG_2, \dots DG_n$, CRT: ComputationRatioTable, CR ComputationRatio, Data_Groups allocated as per node computing capacity</i>
1	<i>Set Jtype \leftarrow JobType;</i>
2	<i>Set DSize \leftarrow Data set size;</i>
3	<i>Set BSize \leftarrow Data block size;</i>
4	<i>Calculate TotalBlocks $\leftarrow [DSizeBSize]$;</i>
5	<i>Set i $\leftarrow 1$; 6. Set total CRatio $\leftarrow 0$;</i>
6	<i>for each record R_i in ComputationRatioTable do</i>
7	<i> if $R_i.Jtype = New.Jtype$ then</i>
8	<i> for each node n_j in ComputationRatioTable do</i>
9	<i> TotalCRatio = TotalCRatio + $n_j.CR$</i>
10	<i> Endfor;</i>
11	<i> for each node n_j in ComputationRatioTable do</i>
12	<i> BCapacity = TotalBlocks * $n_j.CR$</i>
	<i> / $\sum CR$ of all nodes;</i>
13	<i> Place the data group DG_k in node n_j as per node Block capacity</i>
14	<i> endfor</i>
15	<i>else</i>
16	<i> Set all node CRatio $\leftarrow 1$;</i>
17	<i> BCapacity = TotalBlocks * $n_j.CR$</i>
	<i> / $\sum CR$ of all nodes;</i>
18	<i> Place the data group DG_k in node n_j as per node block capacity;</i>
19	<i>Run all applications;</i>
20	<i>for each node n_j in ComputationRatioTable do</i>
21	<i> Calculate new computation ratio NCRatio of node n_j;</i>
22	<i> Update NCRatio in ComputationRatioTable</i>
	<i> $n_j.CR$ = NCRatio;</i>
23	<i>Endfor</i>
24	<i>for each node n_j in ComputationRatioTable do</i>
25	<i> BCapacity = TotalBlocks * $n_j.CR$</i>
	<i> / $\sum CR$ of all nodes;</i>

```

26      Place the data group  $DG_k$  in node  $n_j$  as per node
27      block capacity;
28  endfor
29  endif
30 End

```

In the proposed Algorithm 2, we have introduced a variable block capacity which helps us to decide the number data sets that can be placed in that node at runtime to achieve maximum efficacy considering its heterogeneity. The value of this variable is calculated by (4).

$$\begin{aligned}
 & \text{Block capacity} \\
 & = \text{Total DataBlocks} \\
 & * \left(\frac{\text{Computing ratio of node}}{\left(\sum \text{Computing ratio of all nodes} \right)} \right)
 \end{aligned} \tag{4}$$

Table 3 depicts the block capacity of each node calculated using (4) while considering the computing capacity of those nodes.

Table 3 Node block capacity

Job type	Node	Calculation	Block capacity
Bowtie indexing	N1	$5 * (3 / 5)$	3
	N2	$5 * (2 / 5)$	2
	N3	$5 * (1 / 5)$	1

Since we have two data blocks each in DG_1 and DG_2 and the block capacity of node n_1 and n_2 are 3 and 2 respectively as shown in Table 3, therefore DG_1 and DG_2 can be placed in node n_1 and n_2 respectively. And since DG_3 has only one data block, therefore it can be placed in node n_3 . Algorithm 2 summarises the steps followed in phase 2.

4.3 Entropy-based placement of newly generated datasets

In third phase of the proposed data placement strategy we use the concept of entropy among data sets. Entropy is the measure of information stored or uncertainty in any random variable or groups (Lee et al., 2014). Mathematically, if X is any random variable, $S(X)$ is the set of values that can take and $p(x)$ is the probability function of X , then entropy can be defined as shown in (5).

$$E(X) = - \sum_{x \in S(X)} p(x) \log(p(x)) \tag{5}$$

It has to be understood that the entropy is computed is not entropy in Shannon issue. To calculate the entropy of different data groups we have used the concept of weighted probability where the value of $p(x)$ for each data group is calculated as per equation (6).

$$p(x) = (\text{Co-occurring datasets} / \text{Total datasets}) * RF \tag{6}$$

In equation (6) we have introduced a new variable as relatedness factor (RF) which gives the quantitative value of the number of times two datasets occur together as per equation (7).

$$\begin{aligned}
 RF &= \frac{\text{Count(Dependency)in a group}}{\text{Max}(\text{Count(Dependency)})} \\
 &\quad / \text{Max}(\text{Count(Dependency)})
 \end{aligned} \tag{7}$$

where count (dependency) is the value of the number of times the new data set has been used along with the data sets, in the considered data groups and Max (Count (Dependency)) is the maximum value of count (dependency) among all the datasets considered for placement of the new dataset.

This value of P_w is placed in equation (5) to find the entropy of each data groups. Therefore in this phase whenever we have a new dataset, we first evaluate the relationships of the new data sets with other data groups already present in the Hadoop cluster using the information stored in the *namenode*. From phase 2 of the proposed algorithm we had data groups: $\{D_1, D_3\}$, $\{D_2, D_4\}$, D_5 . Now let us consider that a new data D_6 is added to HDFS. We study the tasks which use D_6 recorded in the *namenode* and try to establish its dependency with other data groups. Let us assume that tasks t_1 , t_2 and t_3 use dataset D_6 .

$$\begin{aligned}
 <t_1> &= \{d_1, d_2, d_4, d_6\} & <t_2> &= \{d_1, d_2, d_5, d_6\} \\
 <t_3> &= \{d_1, d_2, d_4, d_6\}
 \end{aligned}$$

Now, in DG_1 we have data blocks d_1 and d_3 and from above we observe that d_6 occurs thrice with d_1 but it has no occurrence with data block d_3 . Therefore for DG_1 and dataset d_6 we have only one co-occurring data block, i.e., d_1 and the value of count (dependency) for DG_1 and d_6 is 3, as d_1 and d_6 has been used three times among the set of tasks considered. Similarly, for DG_2 having d_2 and d_4 and d_6 has been used together with both d_2 and d_4 , therefore co-occurring datasets in this case is 2 and count(dependency) is 5 as d_6 has occurred thrice with d_2 and twice with d_4 , amounting to a total count of 5. Similarly DG_3 has only one data block d_5 and d_6 and d_5 have been used only once. Therefore co-occurring data blocks and count (dependency) both are 1 in this case.

Also, from the above calculations the value of Max (Count (Dependency)) is 5 as the value of count (dependency) is maximum between DG_2 and d_6 . Algorithm 3 summarises the process of entropy-based data placement. As soon as a new dataset arrives or is available (newly generated), it has to be stored within the HDDPs spanning across multiple data centres. For all data groups in such new datasets, a value of entropy is obtained for vis-a-vis existing data groups and since lower entropy value ensures greater correlation with existing datasets, hence these new datasets are placed accordingly. Steps 6 to 7 of Algorithm 3 are obtained using equations (6) and (7) and Step 10 implicates data placement. To implement these attributes in Hadoop, we introduced a new parameter namely *dependency_partition_id* and *edps_partition_id* within Hadoop's *HDFS-site.xml* configuration file.

Algorithm 3 Entropy-based DataPlacement algorithm

<i>Input</i>	<i>NDS: New Datasets, DC: Set of data centers $dc_1, dc_2, \dots dc_n$, DG: Set of data groups $DG_1, DG_2, \dots DG_n$</i>
<i>Output</i>	<i>New data sets placed with highly related data group</i>
1	<i>Extract information of stack using the NDS from the Name node;</i>
2	<i>Extract task details where NDG is being used along DS in our DGS</i>
3	<i>Calculate count(dependency) = number of times the new data set has been used along with the data sets in the considered data 'g'</i>
4	<i>Find Max(Count(Dependency));</i>
5	$RF = \text{Count(Dependency)} / \text{Max(Count(Dependency))}$
6	<i>Calculate $P_w = \text{co-occurring datasets} / \text{Total data sets} * RF$ for a specific DG;</i>
7	<i>Calculate entropy, $H(DG) = -P_w * \log(P_w)$;</i>
8	<i>Repeat step 2-step 7 for all DG;</i>
9	<i>Find $\text{Min}(H)$;</i>
10	<i>Place NDS in the DG having $H = \text{Min}(H)$;</i>
11	<i>End</i>

4.3.1 Illustrative example for enumerating entropy of new datasets

- P_w between DG_1 and d_6 can be calculated as follows

Total data sets = 2, i.e.,

d_1, d_3 co-occurring datasets = 1, only d_1

Count(dependency in group) = 3

Max(count (dependency in group)) = 5 (in group b)

Therefore using equation (6), i.e.,

$$\begin{aligned} P_w &= (\text{Co-occurring datasets} / \text{Total data sets}) * RF \\ &= (1/2) * (3/5) = 0.3 \end{aligned}$$

Now, using equation (5) entropy,

$$\begin{aligned} H(DG+) &= -P_w * \log(P_w) \\ &= -0.3 * \log(0.3) \\ &= 0.519 \end{aligned}$$

- Between DG_2 and d_6

Total data sets = 2, i.e.,

d_2, d_4 co-occurring datasets = 2, d_2, d_4

Count(dependency in group) = 5

Max(count (dependency in group)) = 5 (in group b)

Therefore using equation (6), i.e.,

$$\begin{aligned} P_w &= (\text{Co-occurring datasets} / \text{Total data sets}) * RF \\ &= (2/2) * (5/5) = 1 \end{aligned}$$

Now, using equation (5) entropy,

$$\begin{aligned} H(DG_1) &= -P_w * \log(P_w) \\ &= -1 * \log(1) \\ &= 0 \end{aligned}$$

- Between DG_3 and d_6

Total data sets = 1, i.e., d_5

Co-occurring data sets = 1, only d_5

Count(dependency in group) = 3

Max(count (dependency in group)) = 5 (in group b)

Therefore using equation (6), i.e.,

$$\begin{aligned} P_w &= (\text{Co-occurring datasets} / \text{Total data sets}) * RF \\ &= (1/1) * (1/5) = 0.2 \end{aligned}$$

Now, using equation (5) entropy,

$$\begin{aligned} H(DG_1) &= -P_w * \log(P_w) \\ &= -0.2 * \log(0.2) \\ &= 0.464 \end{aligned}$$

These calculations and results are summarised in Table 4.

As, discussed earlier that the value of entropy of any system defines the randomness and un-relatedness among any system. Therefore a system with the least value of entropy is considered to be the most stable system. Therefore from the above calculations it was found that the value of entropy between DG_1, DG_2, DG_3 and d_6 is 0.519, 0 and 0.464 respectively. As we can see that the entropy between DG_1 and DG_3 along with d_6 is nearly equal to 0.5 which means that some order of disorder will be present in DG_1 and DG_2 when d_6 is placed in any one of them but entropy in DG_2 and d_6 is the minimum therefore we can place d_6 in DG_2 along with d_2 and d_4 . Therefore the overall entropy of DG_3 having d_2, d_4, d_6 is 0 means that no new disorder or randomness is introduced herewith and that all the data sets are highly dependent on each other. Such placement ensure that of MapReduce jobs with new dataset are scheduled, no run time overhead will be incurred for transferring data nodes.

Table 4 Entropy calculation for data groups

Data group	Total data sets	Co-occurring data sets	Count (dependency in group)	Max (count dependency)	Pw	Entropy
DG1	2	1	3	5	0.3	0.519
DG2	2	2	5	5	1	0
DG3	1	1	1	5	0.2	0.464

Table 5 Hardware Configuration of the cluster employed for experiment

Hardware models details	CPU	RAM	Internal hard disk	Network connection	Node details
HPC Rack 1: Dell Dual Intel Power Edge R610 Xeon quad core E5620	2.93 GHz processor	12 GB DDR2	3 x 300 GB SAS HDD/RAID5	RPS/dual NIC	Master: 4 GB DDR Slave1: 1 GB DDR Slave2: 1 GB DDR Slave3: 3 GB DDR Slave4: 3 GB DDR
HPC Rack 2: Dell Dual Intel Power Edge R410 Xeon quad core E5620	2.4 GHz processor	12 GB DDR2	3 x 300 GB SAS HDD/RAID5	RPS/dual NIC	Slave5: 1 GB Slave6: 2 GB Slave7: 4 GB Slave8: 5 GB
Slave1 :1GB DDR HPC Rack 3: Dell Dual Intel Power Edge R410 Xeon quad core E5620	2.4 GHz processor	12 GB DDR2	3 x 300 GB SAS HDD/RAID5	RPS/dual NIC	Slave9: 2 GB Slave10: 2 GB Slave11: 4 GB Slave12: 4 GB
HPC Rack 4: Dell Dual Intel Power Edge R410 Xeon quad core E5620	2.4 GHz processor	12 GB DDR2	3 x 300 GB SAS HDD/RAID5	RPS/dual NIC	Slave13: 2 GB Slave14: 4 GB Slave15: 4 GB Slave16: 2 GB

Notes: Hadoop details: Hadoop 2.0; Operating system: CentOS 5.0.

5 Experimental results

This section presents the experimental environment and the experimental results for the proposed algorithm.

5.1 Experiment setup

To test the efficacy of the proposed EDPS, we have employed a testbed. The test-bed had four racks, each having 16 nodes with Hadoop 2.0.0 installed in them. Among these 16 nodes, we configured one *NameNode* and *Job Tracker*, the remaining 15 nodes being configured as *DataNode* and *TaskTrackers*. In order to capture heterogeneity, we have purposely configured the nodes with varying capacities, primarily in terms of their memory capabilities, as can be observed from the node details column of Table 5. In a virtualised environment, it is also possible to configure computational capability of nodes. Detailed node configurations employed for the experimenting have been summarised in Table 5.

5.2 Chosen map reduce applications

We used two data intensive applications for experimentation purposes. The first one is a Bowtie indexing (Shannon, 1948; LAngmead et al., 2009; <http://bowtie-bio.sourceforge.net/index.shtml>; <http://enterix.cbcn.umd.edu/>) application that indexes chromosomes of different species using 40 GB of obtained genome data available at the UCSC genome bioinformatics (<http://michael.dipperstein.com/bwt/>; <http://genome.ucsc.edu/>) database; and another map-reduce application pertaining to weather forecasting and analysis of temperature data from the weather dataset obtained from National Climatic Data Centre (NCDC) (Unipro UGENE, 2016; <http://www.ncdc.noaa.gov/>). For checking the efficacy of EDPs in terms of placement efficacy of new datasets, these data were however not uploaded in one go; rather they incrementally added as detailed in the next section.

5.3 Experiments conducted

In order to assess EDPS' performance, this paper conducts three experiments, details of which have been summarised in this subsection. Since the goal of this paper is to study the efficacy of EDPS for possible reduction of runtime through intelligent data placement, the first experiment studies data distribution patterns among clusters, once for HDDP and then for EDPS. Thus in this experiment, we place data once using HDDP and thereafter using EDPS strategy and observe the node-wise data distribution and then rack-wise data distribution patterns. The results are presented in the following subsection. The goal of this experiment is to test EDPS' capability to place related data based on the clustering algorithm depicted in Algorithm 2. The second experiment tests the relative efficacy of EDPS with respect to HDDP in terms of block movements. To test this, we find out the number of data blocks moved at runtime, once using HDDP and thereafter using EDPS. The third experiment was conducted to observe the efficacy of EDPS to place data for newly arrived/generated datasets. To do so, the datasets were incrementally added at different iterations. The first iteration uses only 50% of the datasets and thereafter every subsequent iteration employing an additional 10% data and checking the placement efficacy. For instance, for the NCDC dataset, weather data of the first half of decades were used for placement and then in subsequent iterations, we used data for the next decade to be appended for placement. We also test the overhead of the proposed EDPS in time taken towards decision making where to place the new data. The results of these experiments have been presented in the subsequent subsection.

Table 6 Incremental uploaded genome block in MB size

Species	Stage 1	Stage 2	Stage 3	Stage 4
Human	1,266.804	844.536	1,266.804	1,266.804
Dog	1,266.804	1,689.072	1,689.072	1,689.072
Dolphin	422.268	422.268	844.536	844.536
Elephant	1,266.804	1,689.072	1,689.072	1,689.072
Cow	1,689.072	1,689.072	1,689.072	1,689.072
Atlantic cod	422.268	422.268	422.268	0
Gorilla	1,689.072	1,689.072	1,689.072	1,689.072
panda	1,266.804	1,689.072	1,689.072	1,689.072

5.4 Performance analysis

Since HDDP does not consider heterogeneity among nodes and places data blocks equally and randomly among cluster

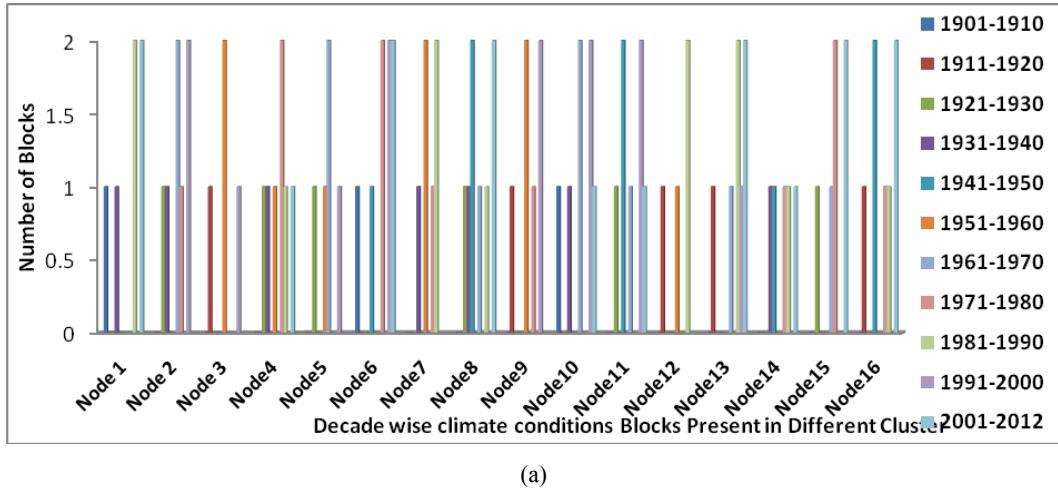
nodes, unlike the proposed EDPS, the number of nodes in which data is distributed is more for HDDP for both the two datasets used. Figures 5(a) and 7(a) depict the node-wise data distribution pattern for the NCDC dataset and the Genome dataset respectively. Figures 5(a) and 7(a) depict the rack-wise data distribution pattern for the NCDC dataset and the Genome dataset respectively.

Figures 6(a) and 8(a) depict the node-wise data distribution pattern for the NCDC dataset and the Genome dataset respectively. Figures 6(b) and 7(b) depict the rack-wise data distribution pattern for the NCDC dataset and the Genome dataset respectively. The dispersedness of the data placement is obvious from these figures when HDDP is employed and this is in stark contrast to the case when EDPS is employed which shows visibly lower dispersedness. It is obvious that such skewed distribution can be a major overhead since they might possibly lead to runtime data movements.

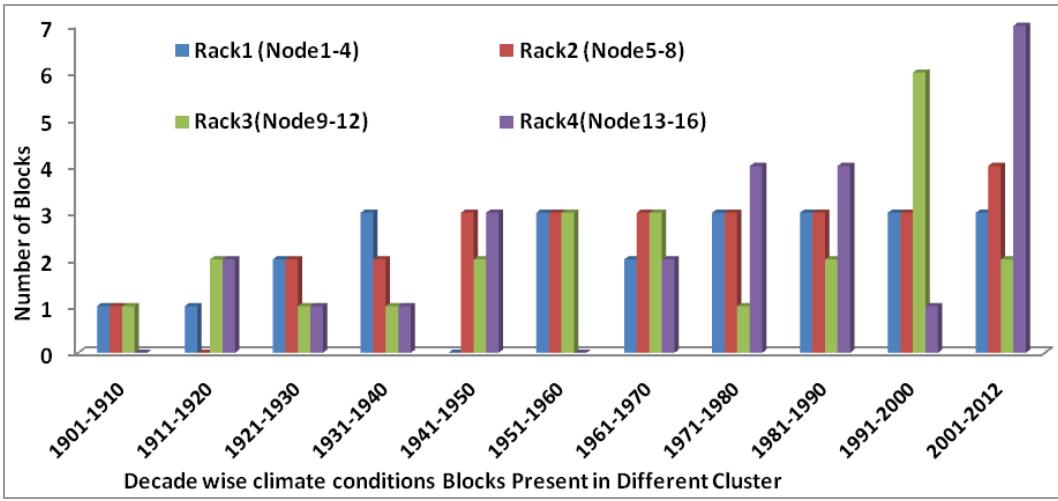
Figures 9(a) and 9(b) presents a comparison the number of block movements involved at runtime while using HDDP and EDPS, the former figure for NCDC data and the latter for the Genome dataset respectively. It has been seen that EDPS can reduce runtime block movements by 30–40% for the datasets used.

Figures 10(a) and 10(b) the relative execution time for the two maps reduces applications have been depicted for the NCDC and the genome dataset respectively. It can be observed that EDPS' reorganised data placements can reduce overall execution time around 30%–35% on an average and up to more than 40% for certain cases. This is a comprehensive improvement over HDDP.

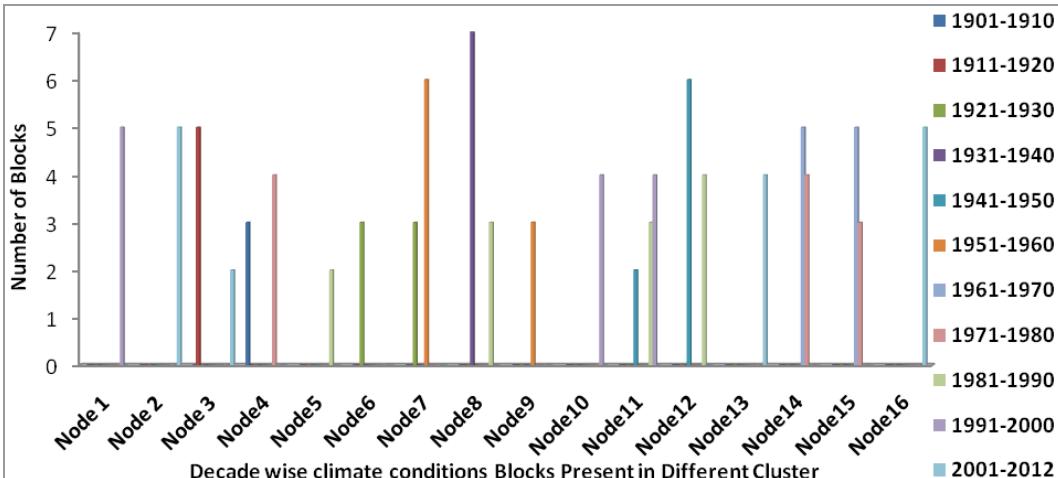
Finally, we capture the efficacy of EDPS in placing data blocks for newly arrived/generated datasets. The benefits if EDPS over HDDP in terms of reduction in data block movements and execution times have been demonstrated. Thus we test the overhead for running EDPS over HDDP. It can be observed from Figures 11(a) and 11(b) that the overhead is within a few milliseconds for dealing with datasets as large as those employed herewith. By incrementally appending new datasets to the Map Reduce applications, EDPS shows almost the same amount of performance benefits in comparison to HDDP. This is because, in HDDP data placement is done randomly without accounting for their interdependencies. Thus, we compare this feature of the EDPS with our previous work DPPACS framework (Reddy et al. 2015), since DPPACS was not designed to handle evolving datasets. Thus in Figures 12(a) and (b) comparative efficacy of HDDP, DPPACS and EDPS has been presented for the standard datasets. EDPS shows dexterity in placing evolving datasets also, amounting close to a 16%–20% improvement on an average.

Figure 5 Hadoop default distribution: NCDC data (see online version for colours)

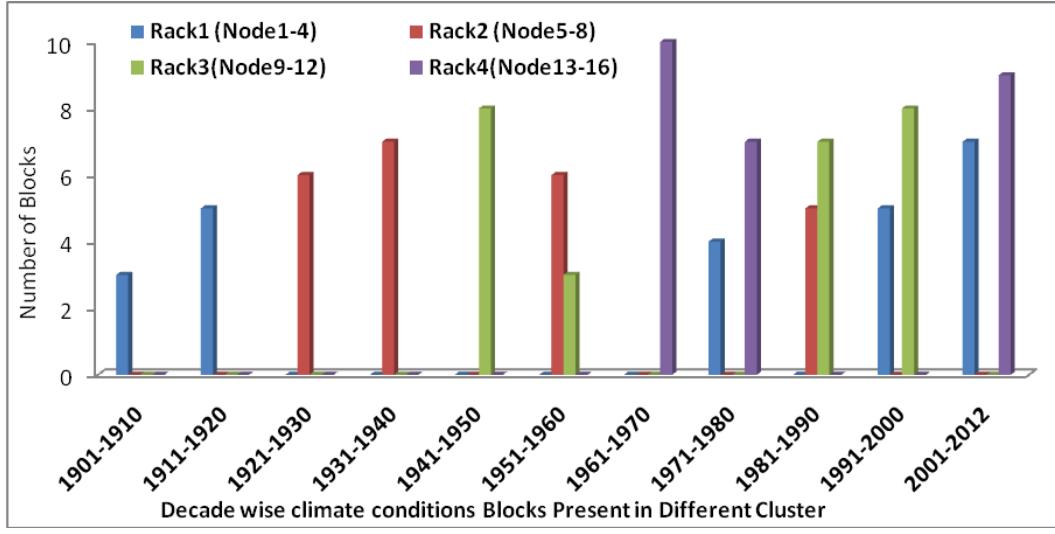
(a)



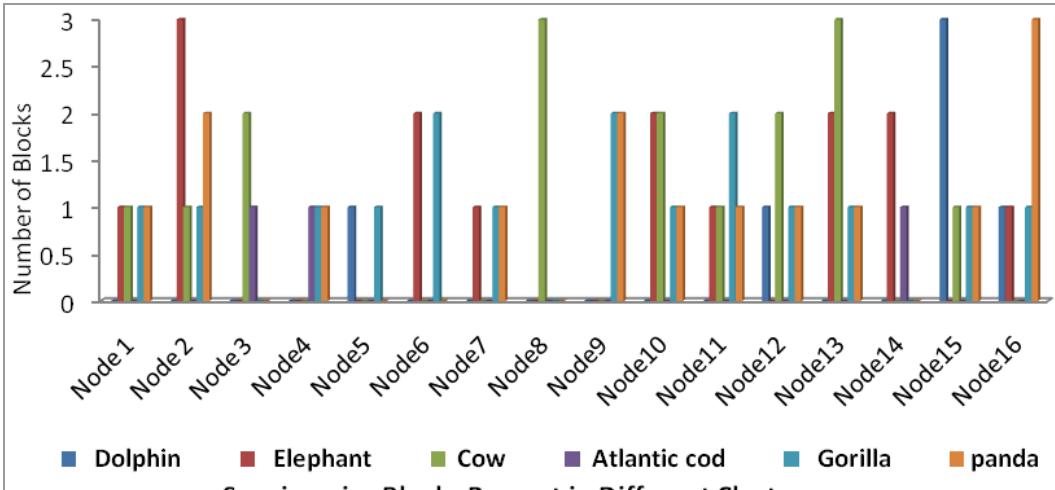
(b)

Figure 6 EDPS distribution: NCDC data (see online version for colours)

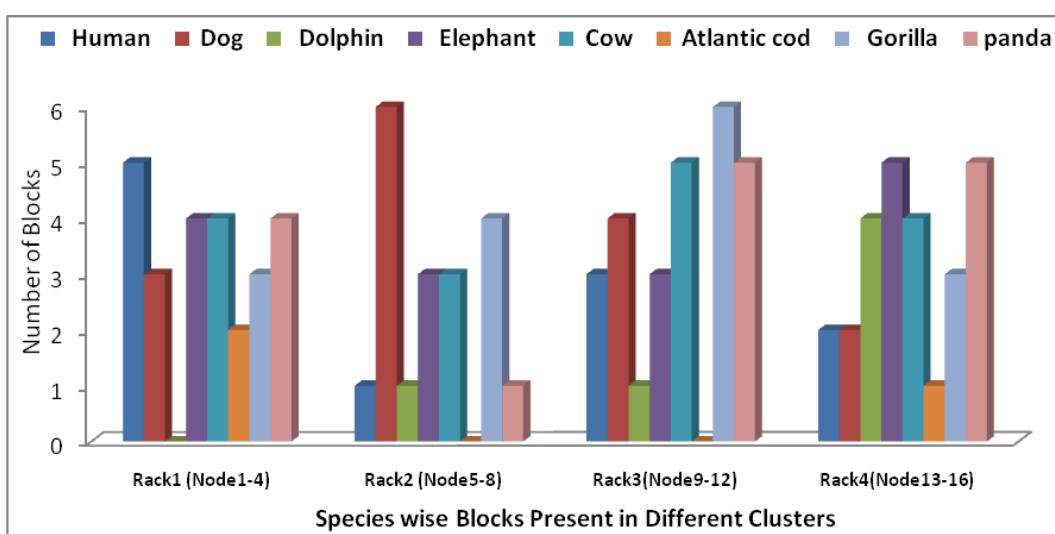
(a)

Figure 6 EDPS distribution: NCDC data (continued) (see online version for colours)

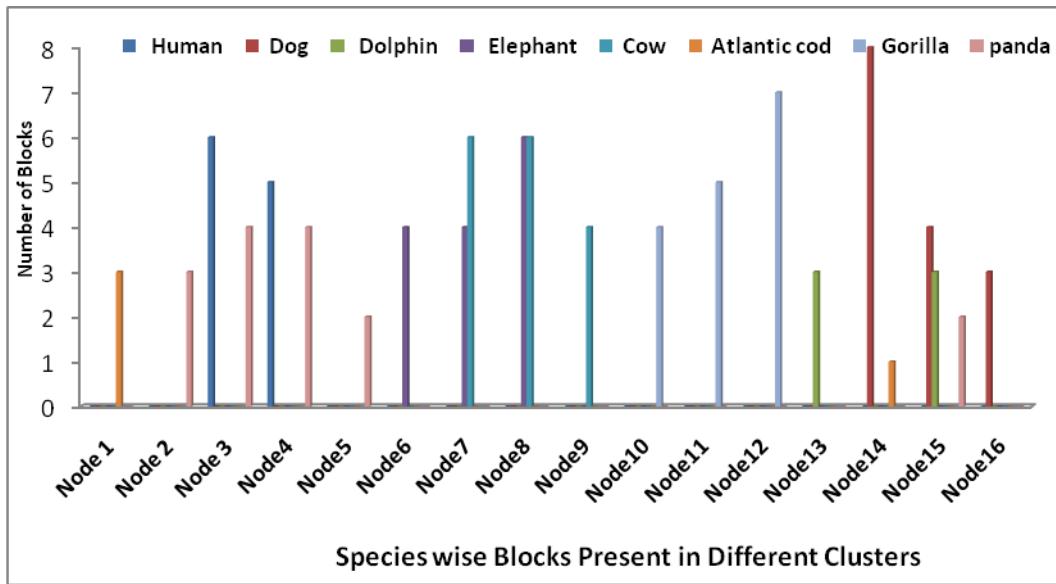
(b)

Figure 7 Hadoop default distribution: genome data (see online version for colours)

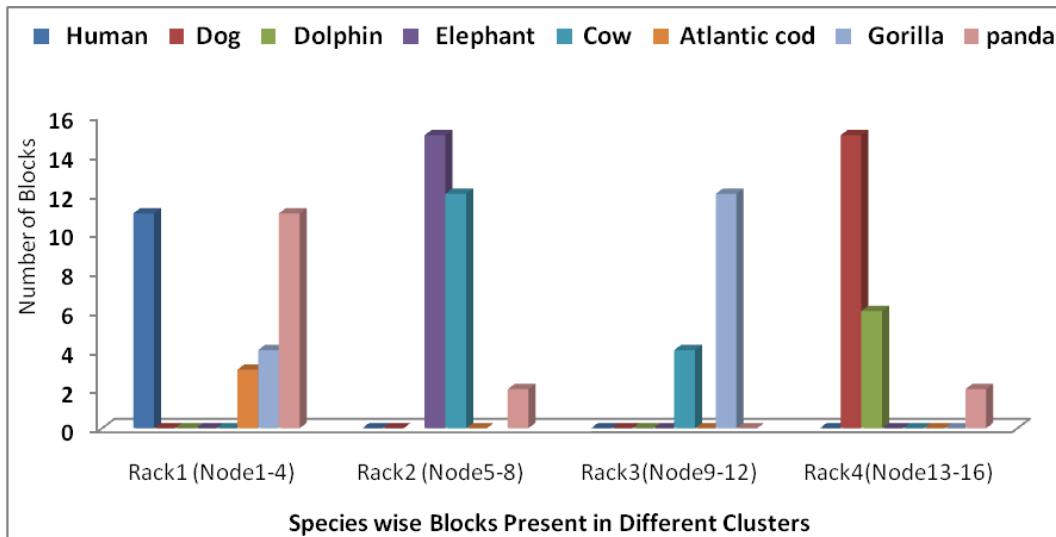
(a)



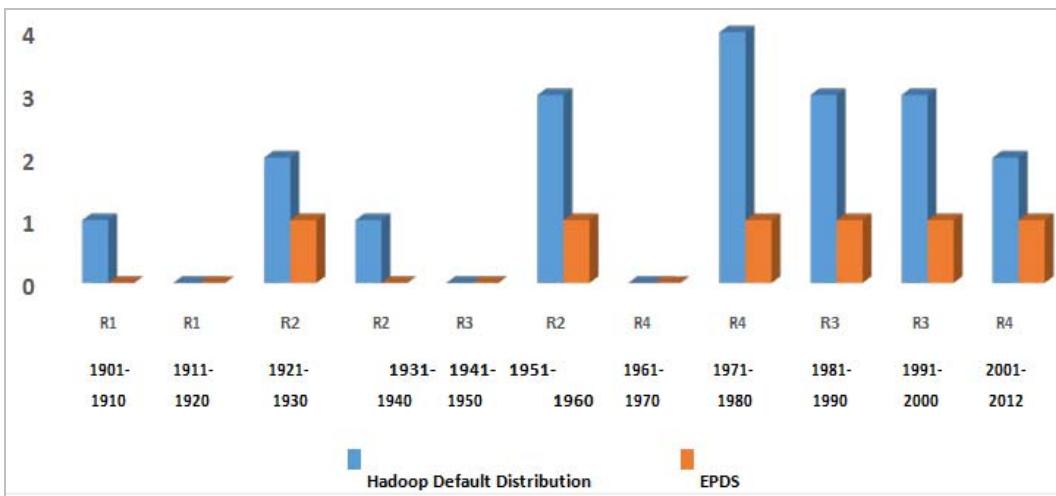
(b)

Figure 8 EDPS distribution: genome data (see online version for colours)

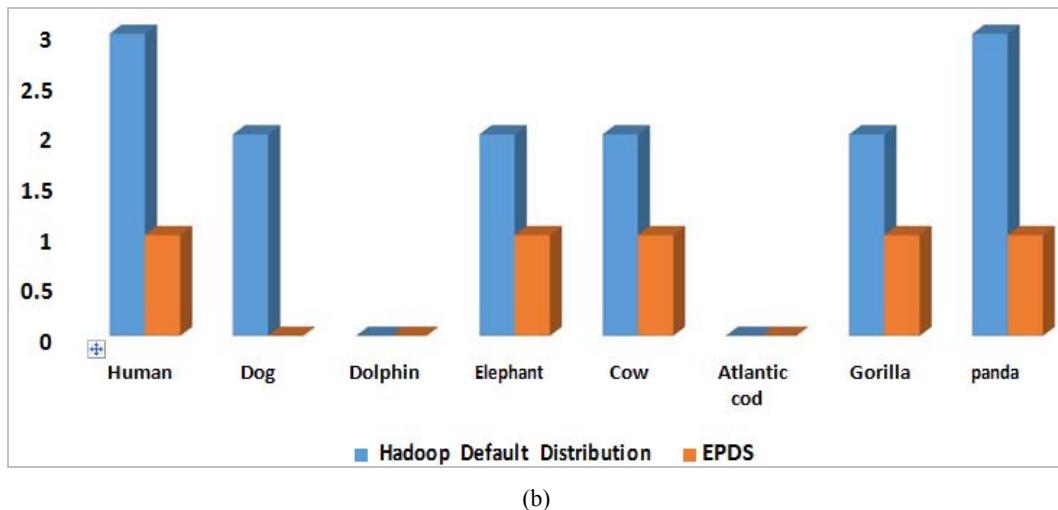
(a)



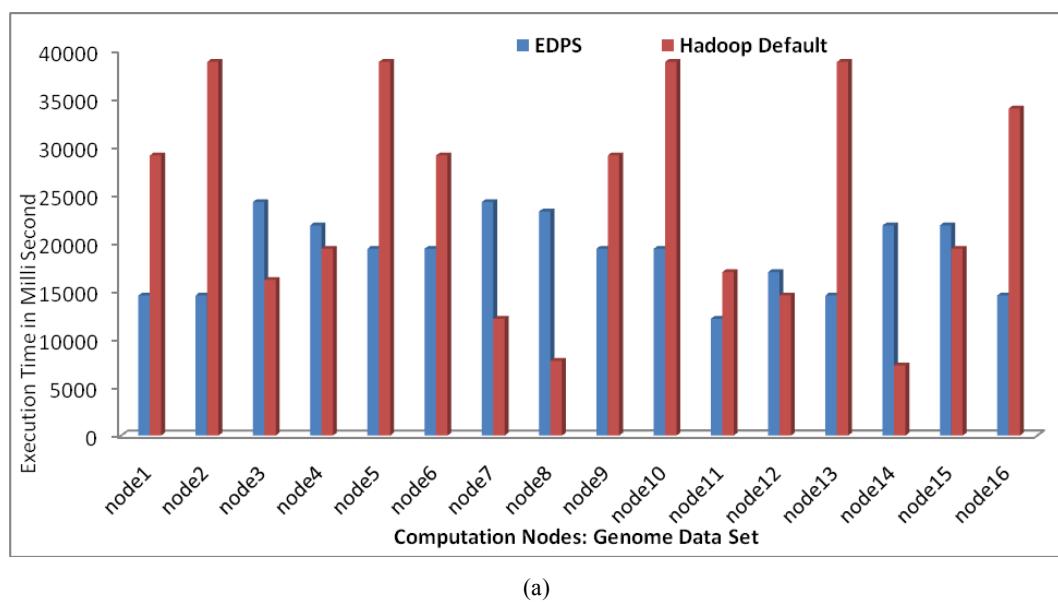
(b)

Figure 9 Number of block movement, (a) NCDC data (b) genome data (see online version for colours)

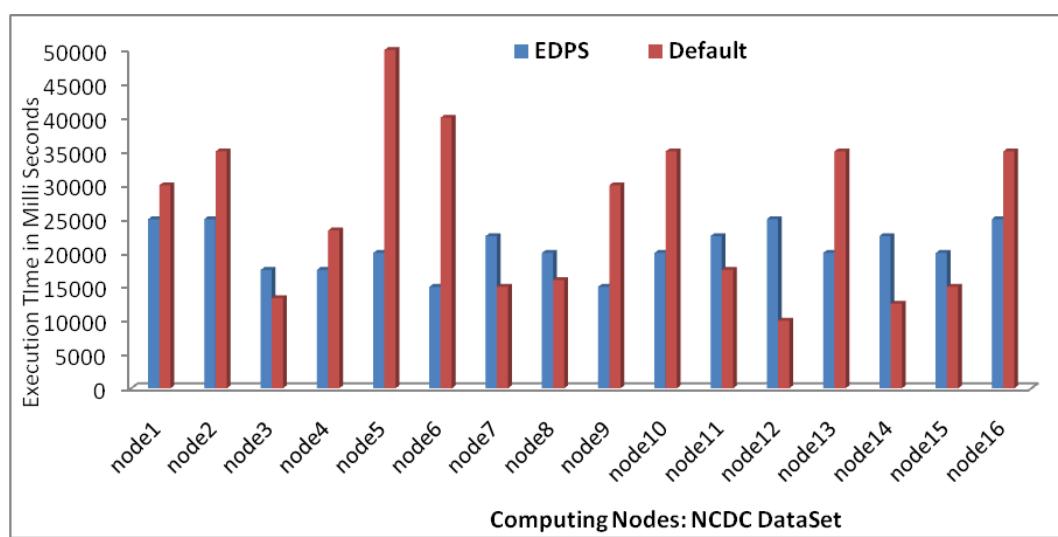
(a)

Figure 9 Number of block movement, (a) NCDC data (b) genome data (continued) (see online version for colours)

(b)

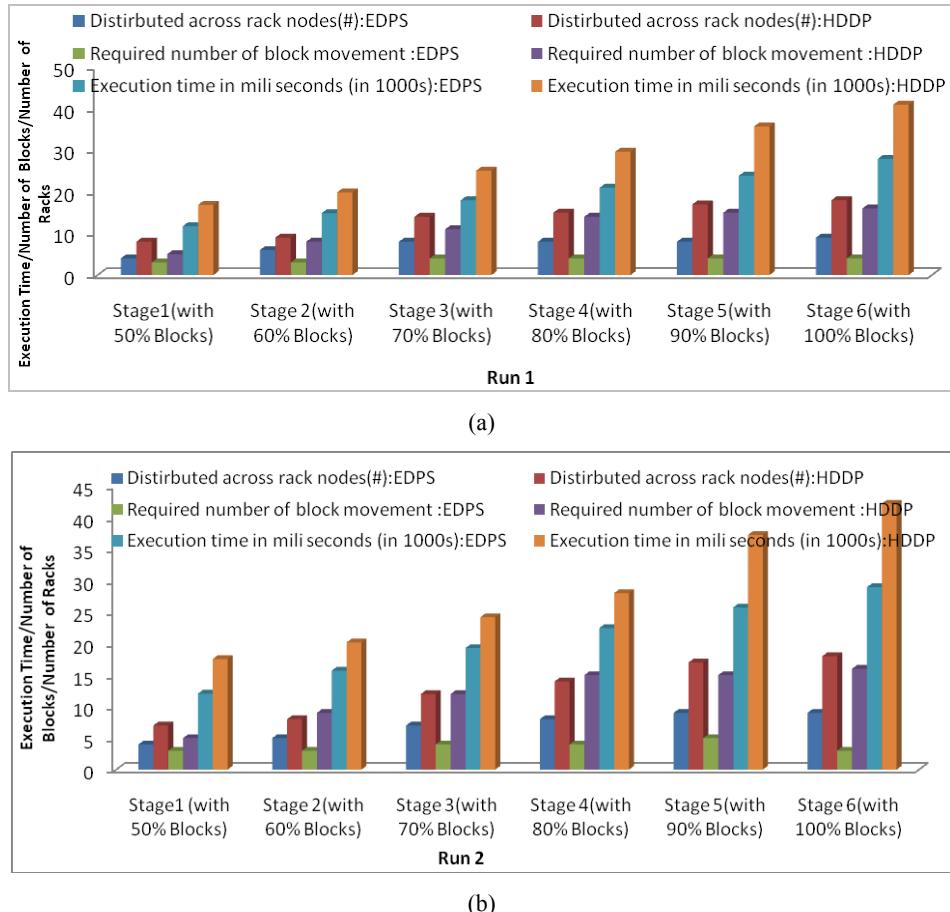
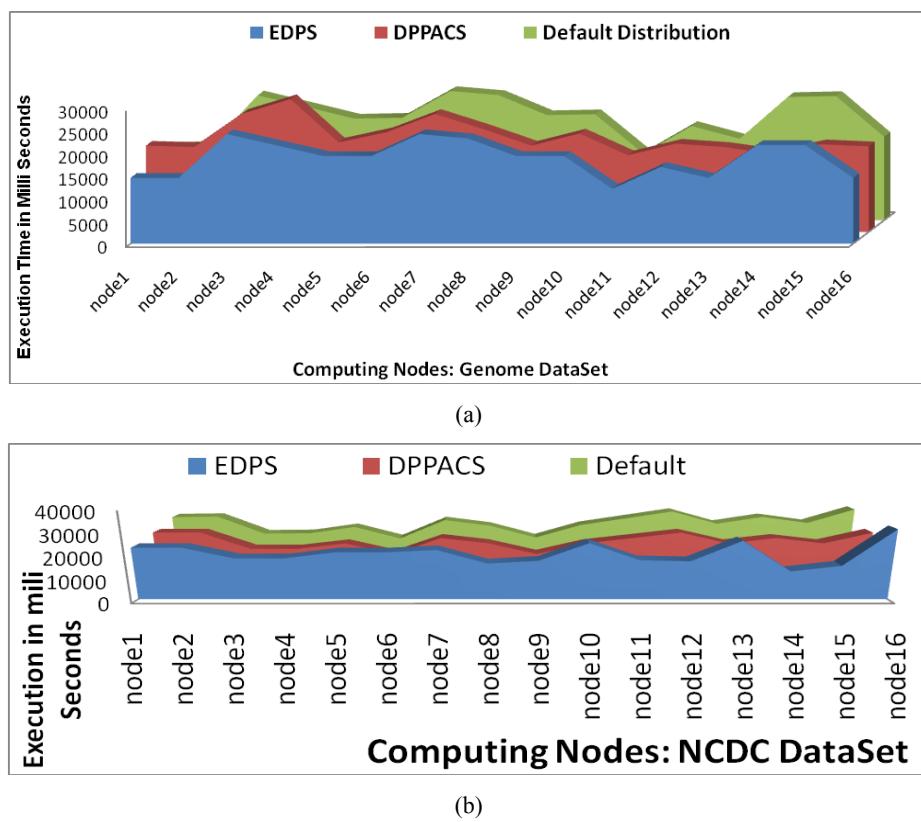
Figure 10 Execution time, (a) NCDC data (b) genome data (see online version for colours)

(a)



Computing Nodes: NCDC DataSet

(b)

Figure 11 Phase wise execution performance, (a) Run1 (b) Run2 (see online version for colours)**Figure 12** Execution performances of three approaches with (a) genome data and (b) NCDC data (see online version for colours)

6 Conclusions

This paper proposes an entropy-based data placement strategy that can dynamically place data among nodes in virtualised environments. The dynamic nature is both in terms of handling heterogeneity of nodes in virtualised clusters as well as for handling new datasets. Experiments carried on a pseudo virtualised cluster have been used to test efficacy of the proposed EDPS for two real life data intensive applications. It has been demonstrated that EDPS shows better performance in terms of compact data placement among nodes, reduces runtime data movements consistently in excess of 30% and consistently reduces application runtime in excess of 40% while incurring very little overhead for making such placement decisions. The uniqueness of the proposed EDPS is its capability handle evolving datasets by means of a novel entropy expectation calculation that it does before placing data among virtualised clusters.

Acknowledgements

This work has been facilitated by and carried out in parts at the High Performance Computing (HiPC) Lab and the Data Sciences Lab, Department of Computer Science and Engineering, National Institute of Science and Technology, Berhampur. This work was partially funded by the fellowship NSRIF-15-1011 awarded by the National Institute of Science and Technology, Berhampur, India.

References

- Amazon (2016) *Amazon Elastic MapReduce* [online] <http://aws.amazon.com/elasticmapreduce/> (accessed 10 January 2016).
- Apache Software Foundation (2016) *Hadoop* [online] <http://hadoop.apache.org> (accessed 10 January 2016).
- Barbará, D., Li, Y. and Couto, J. (2002) ‘COOLCAT: an entropy-based algorithm for categorical clustering’, *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, ACM.
- Baru, C. et al. (1998) ‘The SDSC storage resource broker’, *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research*, IBM Press.
- Borthakur, D. et al. (2011) ‘Apache Hadoop goes real-time at Facebook’, *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ACM.
- Buyya, R. and Venugopal, S. (2004) ‘The gridbus toolkit for service oriented grid and utility computing: an overview and status report’, *1st IEEE International Workshop on Grid Economics and Business Models, GECON 2004*, IEEE.
- Chang, H. et al. (2011) ‘Scheduling in mapreduce-like systems for fast completion time’, *INFOCOM, 2011 Proceedings IEEE*, IEEE.
- Condie, T. et al. (2010) ‘MapReduce online’, *NSDI*, Vol. 10, No. 4, p.20.
- De Souza, J.B.B. and De Araújo, AP.F. (2017) ‘HCEm model and a comparative workload analysis of Hadoop cluster’, *International Journal of Big Data Intelligence*, Vol. 4, No. 1, pp.47–60.
- Dean, J. and Ghemawat, S. (2008) ‘MapReduce: simplified data processing on large clusters’, *Communications of the ACM*, Vol. 51, No. 1, pp.107–113.
- Fagin, R., Nievergelt, J., Pippenger, N. and Strong, H.R. (1979) ‘Extendible hashing – a fast access method for dynamic files’, *ACM Transactions on Database Systems*, New York, NY, Vol. 4, No. 3, pp.315–344.
- Ghemawat, S., Gobioff, H. and Leung, S.T. (2003) ‘The Google file system’, *ACM SIGOPS Operating Systems Review*, Vol. 37, No. 5, ACM.
- Hadoop (2016) [online] <http://hadoop.org/> (accessed 15 January 2016).
- Hagos, D.H. (2016) ‘Software-defined networking for scalable cloud-based services to improve system performance of hadoop-based big data applications’, *International Journal of Grid and High Performance Computing (IJGHPC)*, Vol. 8, No. 2, pp.1–22.
- He, B. et al. (2008) ‘Mars: a MapReduce framework on graphics processors’, *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ACM.
- Hsu, C-H., Slagter, K.D. and Chung, Y-C. (2015) ‘Locality and loading aware virtual machine mapping techniques for optimizing communications in mapreduce applications’, *Future Generation Computer Systems*, Vol. 53, pp.43–54.
- IBM (2000) *IBM Builds world's Fastest Supercomputer to Simulate Nuclear Testing for US Energy Department*, IBM Press Release, Poughkeepsie, NY, 29 June.
- Langmead, B., Trapnell, C., Pop, M. and Salzberg, S.L. (2009) ‘Ultrafast and memory-efficient alignment of short DNA sequences to the human genome’, *Genome Biol.*, Vol. 10, p.R25.
- Lee, C-W. et al. (2014) ‘A dynamic data placement strategy for Hadoop in heterogeneous environments’, *Big Data Research*, Vol. 1, pp.14–22.
- Patil, V.A. and Chaudhary, V. (2013) ‘Rack aware scheduling in HPC data centers: an energy conservation strategy’, *Cluster Computing*, Vol. 16, No. 3, pp.559–573.
- Reddy, K., Kumar, H. and Roy, D.S. (2015) ‘DPPACS: a novel data partitioning and placement aware computation scheduling scheme for data-intensive cloud applications’, *The Computer Journal*, bxv062.
- Reddy, K.H.K., Mudali, G. and Roy, D.S. (2017) ‘A novel coordinated resource provisioning approach for cooperative cloud market’, *Journal of Cloud Computing*, Vol. 6, No. 1, p.8.
- Schmuck, F.B. and Haskin, R.L. (2002) ‘GPFS: a shared-disk file system for large computing clusters’, *Proceedings of the Conference on File and Storage Technologies*, pp.231–244, 28–30 January 2002.
- Schmuck, F.B., Wyllie, J.C. and Engelsiepen, T.E. (1999) *Parallel File System and Method with Extensible Hashing*, US Patent No.05893086.
- Shannon., C.E. (1948) ‘A mathematical theory of communication’, *Bell System Technical Journal*, Vol. 5, No. 1, pp.379–423.
- Srikumar, V., Buyya, R. and Winton, L. (2004) ‘A grid service broker for scheduling distributed data-oriented applications on global grids’, *Proceedings of the 2nd workshop on Middleware for Grid Computing*, ACM.
- Unipro UGENE (2016) [online] <http://ugene.unipro.ru> (accessed 15 January 2016).

- VERITAS Software Corporation (2000) *VERITAS SANPoint Direct File Access*, Whitepaper, VERITAS Software Corporation, Corporate Headquarters, 1600 Plymouth Street, Mountain View, CA 94043, August.
- Wang, J. and Li, X. (2016) 'Task scheduling for MapReduce in heterogeneous networks', *Cluster Computing*, Vol. 19, No. 1, pp.197–210
- Wang, J., Shang, P. and Yin, J. (2014) 'DRAW: a new data-grouping-aware data placement scheme for data intensive applications with interest locality', *Cloud Computing for Data-Intensive Applications*, pp.149–174, Springer New York.
- Xie, J. et al. (2010) 'Improving mapreduce performance through data placement in heterogeneous hadoop clusters', *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, IEEE.
- Xiong, R., Luo, J. and Dong, F. (2015) 'Optimizing data placement in heterogeneous Hadoop clusters', *Cluster Computing*, Vol. 18, No. 4, pp.1465–1480.
- Yuan, D., Yang, Y., Liu, X. and Chen, J. (2010) 'A data placement strategy in scientific cloud work-flows', *Future Gener. Comput. Syst.*, Vol. 26, No. 8, pp.1200–1214.
- Zaharia, M. et al. (2008) 'Improving MapReduce performance in heterogeneous environments', *OSDI*, Vol. 8, No. 4.
- Zhou, H. et al. (2016) 'P-aware: a proportional multi-resource scheduling strategy in cloud data center', *Cluster Computing*, Vol. 19, No. 3, pp.1089–1103.

Websites

- <http://enterix.cbcn.umd.edu/enteric/enteric-eco.html> (accessed 15 January 2016).
- <http://genome.ucsc.edu/> (accessed 15 January 2016).
- <http://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/integrated-surface-database-isd> (accessed 15 January 2016).
- <ftp://ftp3.ncdc.noaa.gov/pub/data/noaa/> (accessed 15 January 2016).