An Energy-efficient Storage Strategy for Cloud Datacenters based on Variable *K*-Coverage of a Hypergraph

Ting Yang, Haibo Pen, Wei Li, *Senior Member, IEEE,* Dong Yuan, *Member, IEEE*, Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Distributed storage systems, e.g. Hadoop Distributed File System (HDFS), have been widely used in datacenters for handling large amounts of data due to their excellent performance in terms of fault tolerance, reliability and scalability. However, these storage systems usually adopt the same replication and storage strategy to guarantee data availability, i.e. creating the same number of replicas for all data sets and randomly storing them across data nodes. Such strategies do not fully consider the difference requirements of data availability on different data sets. More servers than necessary should thus be used to store replicas of rarely-used data, which will lead to increased energy consumption. To address this issue, we propose an energy-efficient storage strategy for cloud datacenters based on a novel hypergraph coverage model. According to users' requirements of data availability in different applications, our proposed algorithm can selectively determine the corresponding minimum hyperedge coverage, which represents the minimum set of data nodes required in the datacenter. Hence, some other data nodes can be turned off for the purpose of energy saving. We have also implemented our proposed algorithm as a dynamic runtime strategy in a HDFS based prototype datacenter for performance evaluation. Experimental results show that the variable hypergraph coverage based strategy can not only reduce energy consumption, but can also improve the network performance in the datacenter.

Index Terms—Energy saving, cloud datacenters, data availability, hypergraph, minimum *K*-coverage

1 Introduction

TODAY, we are facing the challenges of storing and processing the ever-increasing amount of data produced by various applications – social network data sharing, worldwide collaborative scientific projects etc. Cloud Computing is deemed to be a powerful computing paradigm to meet this challenge as it reduces the need for and costs of maintaining expensive computing hardware, dedicated space, and software [1], [2], [3]. Datacenters are the key infrastructure for Cloud Computing to provide the resources needed for data processing and storage at all scales. Along with the increasing requirements from end-users and their willingness to use cloud-based services, current applications have become data intensive, which raises more critical demands for the storage capacity of datacenters. An increasing number of datacenters have been built around the world to store the massive amount of data, which not only consumes a huge amount of energy in order to maintain their capabilities, but also becomes a major source of pollution (carbon emissions) [4], [5], [6]. To develop energy-efficient storage strategies for datacenters has now become a key research issue to further reduce the energy consumption of datacenters, so that the goal of green computing can eventually be achieved.

A distributed storage system [7], [8] is one of the key elements of Cloud datacenters. Improving the performance of data storage at the system level can greatly advance the computation efficiency of Cloud Computing, as well as the energy efficiency of datacenters. Currently, data replication technologies are widely used in distributed storage systems to enable better data reliability [9], [10]. For example, the Hadoop Distributed File System (HDFS) generates three replicas by default for every data set and randomly stores them across all available data nodes, which are hosted on the physical machines. With this strategy, if any kind of failure happens to a file, the data availability of this file can be guaranteed since its replicas can still be retrieved from other data nodes. However, this straightforward approach encounters several issues. Firstly, the data replication strategy increases the datacenter's need for storage capacity since the data replicas require more data nodes in order to store the data at all times. With more data nodes in working mode, there will inevitably be more energy consumption for the datacenter. Secondly, the random replication strategy does not systematically consider the location of a data replica, which could cause a noticeable negative impact on the computation performance as a data replica could be used in the middle of the computation. Last but not least, the accessing rate for each dataset is not taken into consideration, and the same number of data replicas is simply used for all datasets. However, using the same number of data replicas for all data goes against real world scenarios. According to a statistical study conducted at a Google

- *Ting Yang and Haibo Pen are with the Key Laboratory of Smart Grid of Ministry of Education, School of Electrical and Information Engineering, Tianjin University, Tianjin, China. E-mail: yangting@tju.edu.cn*
- *Wei Li and Albert Y. Zomaya are with School of Information Technology, The University of Sydney, Australia, E-mail: liwei@it.usyd.edu.au, albert.zomaya@sydney.edu.au*
- *Dong Yuan is with School of Electrical and Information Engineering, The University of Sydney, Australia, E-mail: dong.yuan@sydney.edu.au*

datacenter [11], the real data accessing rate follows the Pareto principle, where only a small part of the stored data is frequently used.

To address the aforementioned issues, we propose an energy-efficient improved HDFS storage strategy based on a variable *K*-coverage of hypergraph model. In our proposal, we develop an algorithm by utilizing the rationale of $\widetilde{K}$ - transverse from hypergraph theory, so that we can calculate the minimum coverage of data nodes according to the actual needs of each data block. More precisely, the variable *K*-coverage represents the optimal set of data nodes that needs to be run to satisfy the replicas availability requirements of each dataset. By doing this, the data nodes not in the coverage can be put into power saving mode for a period of time. Further, minimizing the number of running data nodes can also improve the overall network performance of the datacenter.

The major contributions of this paper can be summarized as follows.

- We develop a hypergraph-based storage model for Cloud datacenters, which can precisely represent the many-to-many relationship among files, data blocks, data racks, and data nodes.
- To improve the energy efficiency of HDFS-based systems, our approach can dynamically adjust the number of active replicas for each data block according to application requirements of data availability. Besides, our approach does not change the original rule of producing data replicas in HDFS, so it provides better backward compatibility compared to existing systems.
- From our developed hypergraph model, we propose a novel $\widetilde{K}$ - transverse of the hypergraph algorithm to calculate the minimum set of data nodes required to guarantee the data availability requirement. By running the required data nodes, we can not only save energy for the datacenter, but also maintain full functionality.
- We also propose a practical runtime strategy in conjunction with our proposed variable *K*-replica coverage algorithm. The conjunction strategy can dynamically adapt the incoming requests for data storage in datacenters. By doing this, our approach can achieve load balance among data nodes.

The remainder of the paper is organized as follows: Section 2 briefly reviews related work in the field. In Section 3, we introduce our developed hypergraph-based storage model to precisely represent the relationship of files, data blocks, data nodes and racks followed by the problem description. In Section 4, we propose our approach based on the $\widetilde{K}$ - transverse from hypergraph theory. The runtime duty cycle strategy is proposed in Section 5 to achieve load balance among data nodes. Section 6 presents the experimental results of our algorithm. We conclude the paper in Section 7.

2 Related Work

A distributed storage system is considered to be the key component for data storage in cloud datacenters [2] [3], HDFS has been widely used as the representative example. However, due to the ever-growing amount of data, the issues of using HDFS have been gradually revealed, e.g. high energy consumption and low resource utilization on processing big data. Several works have focused on addressing energy saving issues in datacenters. Afianian et al. [12] proposed a dynamic data management system for Hadoop, called Apstore. By storing data in different layers of the system according to their popularity, Apstore can improve the I/O throughput and reduce storage costs for Hadoop. Armbrust et al. [3] proposed a method whereby the idle periods of a data node can be identified during the execution of a task in Hadoop, which can be used to help design energy saving strategies. Yun et al. [13] proposed a memory based distributed key-value storage system called CloudM, which provides functions like read, write, backup and recovery for big data applications. In CloudM, all data are loaded to the memory in order to improve the overall system performance, and a hash algorithm is adopted to achieve the load balance and extensibility of the system, while data backup and fast recovery strategies are used to guarantee system reliability.

Apart from the work mentioned above, the work on energy-efficient storage for datacenters can be classified into two groups: one is to design new storage strategies to achieve better energy efficiency, and the other is to design mechanisms to save energy without changing the original storage strategy.

In the group of designing new storage strategies, Cheng et al. [14] proposed an elastic data replication strategy call ERMS, which contains a transaction engine to differentiate data popularity by a temperature model. By dynamically changing the temperature of data according to real-time visiting requests, the system can promptly remove the "cold" data in order to achieve energy saving for the datacenter. Abad et al. [15] proposed a self-adaptive replication algorithm, DARE, which optimizes the number of data replicas and solves the data locality problem based on the existing remote data access system. Maheshwari et al. [16] designed a power controller and a data block migration strategy. In this work, the controller is used to monitor the status of clusters; if the current workload of the system is lower than a pre-defined threshold, some data nodes will be turned off for saving energy and the data will be migrated to other nodes. Liao et al. [17] defined the "cold-zone" and "hot-zone" in the datacenter according to the visiting pattern of data nodes. By migrating data to the hot-zone, the idle data nodes in the cold-zone can hibernate in order to save energy.

In a datacenter, energy saving on the data nodes is different from the computation nodes. For computation nodes, the virtual machines running on them can be integrated and migrated with small cost so as to improve the CPU utilization as well as energy efficiency [18], [19]. However, for data nodes, to move data around will consume large computation, I/O, and bandwidth resources. To make matters worse, the potential data collision in the data transmission among data nodes could lead to bad system performance, even permanent data loss. Hence, a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

data migration based approach is not the best way to achieve energy saving in datacenters. In light of this, research on energy saving without changing the storage strategy has appeared in recent years.

Based on analysis of real systems, Harnik et al. [20] found a big difference in data access rate between daytime and nighttime with a strong pattern of periodicity. Based on this finding, they introduced an auxiliary node to find the coverage of data nodes for all data items and data nodes outside the covering set that can be shut down to save energy. Kim et al. [21] proposed a system architecture whereby every data block will have at least one replica in the covering set in order to guarantee the data availability requirement. This means that the data node outside the covering set can be shut down for energy saving. The authors further proposed the algorithms to calculate the 1-replica covering set and *K*-replicas covering set for the datacenter [22]. Yazd et al. [23] proposed a data block mirroring approach for data placement and storage. In this approach, they first calculate the covering set, and then use the same placement of data blocks in the covering set to store the data replicas in the data nodes outside the covering set. Whenever the data nodes in the covering set cannot fulfill the data availability requirement of a task, it can directly start up the mirroring data nodes with the required replicas and does not need to start up all data nodes and search for the data.

As mentioned, most research on datacenter energy saving without changing storage strategy is based on a graph complete coverage model [22] [24]. Existing algorithms for the replication strategy used by HDFS did not consider the difference of usage frequency of data files. Therefore, a fixed number of replicas (normally three by default) is used in the storage strategy and 1-replica coverage of data nodes is calculated for the energy saving strategy. In these approaches, the data node consisting of frequently used data files will have high pressure on network and I/O performance during simultaneous reading and storing of data. On the contrary, more redundant replicas in the covering set will raise the required storage, and cause energy resources to be wasted dramatically.

In order to cope with the issues discussed above, we propose a variable *K*-replicas coverage strategy for HDFS datacenters to achieve better energy saving. With the help of a hypergraph-based model of datacenters, we precisely represent the many-to-many relationship among files, data blocks, data nodes and racks. To dynamically determine the specific number of replicas needed for different data blocks, we propose a $\widetilde{K}$ -transverse based algorithm to calculate the variable minimum $\widetilde{K}$ -coverage hyperedge set with the consideration of data availability requirements. By running the minimum number of data nodes, we can achieve significant energy saving for the datacenter. Furthermore, our proposed strategy can also improve the overall network performance of the datacenter.

3 The Hypergraph Models and Problem Formulation

3.1 Hypergraph Model of HDFS infrastructure

A typical HDFS datacenter is composed of multiple racks, and each rack contains a number of physical machines serving as storage servers, also known as DataNodes. The racks are connected by wired switches, which realize data transmissions among DataNodes, as shown in Fig. 1.

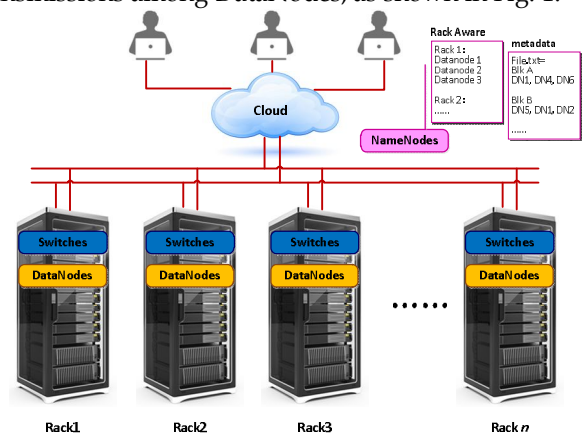

Fig. 1. Typical datacenter architecture and HDFS storage

Files stored in the HDFS-based datacenter are split into a number of data blocks, and the default size of each data block is 64M. In order to improve the reliability of data blocks, HDFS creates multiple replicas for each data block in the datacenter. The replicas and the original data blocks will be stored in DataNodes according to the rack-aware storage strategy [8], which works as follows.

In the rack-aware storage strategy, the first replica of the data block is randomly stored in one of the available DataNodes. The second replica is stored in a DataNode that is located in a rack that does not contain the first replica. Then the third replica will be stored in the same rack that is storing the second replica but in a different data node. If the generated number of replicas of a given data block is greater than three, the remaining blocks will be randomly stored in any DataNodes except the DataNodes that store the first, second and third replicas [8].

As introduced above, the data blocks of a given file, the replicas of the data blocks and DataNodes implement many-to-many associations. In other words, the replicas of a given file can be stored in different DataNodes and each DataNode can store multiple replicas of different files. The traditional graph theory can accurately represent the binary associations by using an adjacency matrix. The adjacency matrix has one row and one column for each vertex. If the vertex x_i is adjacent to x_j, then $(i.j)-$ entry in the matrix is 1, otherwise it is 0. However, this characteristic no longer holds for the many-to-many associations. To address this issue, we employ the hypergraph theory and propose a hypergraph model to better represent the storage architecture of HDFS-based datacenters.

Definition 1. *Let* $X=\{x_1, x_2, \cdots, x_n\}$ *be a finite set and* $E=\{E_1, E_2, \cdots E_m\}$ *be a set of non-empty subsets of X, If* $E_i \neq \Phi$ (i=1, 2, 3, …, m) *and* $\bigcup_{i=1}^{m} E = X$, *the binary association*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

$H=(X,E)$ is defined as a hypergraph, where the elements x_i in X are called the vertex of the hypergraph, and E_i is called the hyperedge [25].

By representing the storage architecture of HDFS-based datacenters as a hypergraph model, DataNodes are mapped to the vertex set *X* of the hypergraph *H*, and each type of data block and its replicas are mapped to a specific hyperedge E_i. By doing this, we can accurately represent the many-to-many associations between data blocks and DataNodes:

(1) Data block *b* and its replicas are stored in different DataNodes, which is represented as $E_b \supset \{x_i, x_j, ..., x_k\}$. The rank of the hyperedge E_b is defined as the number of vertices contained, denoted by $r(E_b)$. In other words, it indicates the number of DataNodes used to store the data block *b* and its replicas.

(2) The number of data blocks stored in a DataNode x_i can be represented by the number of hyperedges connected to that vertex in the hypergraph. This is called degree of the vertex x_i and denoted by $d_H(x_i)$. It can be further described as below.

Definition 2. *(Hypergraph Incidence Matrix). We use the incidence matrix $A(a_{ij})$ to represent the hypergraph H (X, E), where the jth column in the matrix A represents the hyperedge j in H (j=1, 2, …, m), and the ith row in the matrix A represents the vertex i in H (i=1, 2, …, n). If $x_i \in E_j$, then a_{ij}=1, otherwise, a_{ij}=0.*

Therefore, the node degree of DataNode x_i is $d_H(x_i) = \sum_{j=1}^{m} a_{ij}$ according to Definition 2.

3.2 A variable $\widetilde{\kappa}$-covering model for HDFS

In the above sub-section, we proposed a general hypergraph model for HDFS storage. However, the usage frequency and the importance of data blocks are different in HDFS. In order to further include these practical factors in our model, we develop a variable $\widetilde{\kappa}$-covering model to better describe the actual datacenter requirements, where $\widetilde{\kappa}$ represents the particular replica requirement for each data block. In other words, for a given data block, to solve its $\widetilde{\kappa}$-covering problem is to search a minimal covering set of DataNodes, which covers variable replicas of all types of data blocks. In this covering set, the number of replicas of each data block is a different $\widetilde{\kappa}$ according to the data availability requirements.

If $k_1=k_2=\ldots=k_m=k$, it is a general *k*-covering problem, and if *k*=1, it is known as 1-covering problem, which is the special case of the *k*-covering problem. Several previous works, e.g. [22] and [26], create active zones to store only one replica for each data block, which is equivalent to the 1-covering problem. Unlike these solutions, we defined the variable $\widetilde{\kappa}$-covering problem in our model, since $\widetilde{\kappa}$ could be different according to data availability requirements. In addition, $\widetilde{\kappa}$-covering problem is more general than the *k*-covering problem so that it is more complex and difficult than the traditional *k*-covering problem when *k* is a fixed value for all data blocks.

Definition 3. *Let Γ be a $\widetilde{\kappa}$-covering set of hypergraph H. If $\forall\, x \in \Gamma$ makes $\left\|(\Gamma - x) \cap E_j\right\| \geq k_j$ no longer valid, Γ is a feasible solution of the minimal $\widetilde{\kappa}$-covering sets.*

If we find a minimal $\widetilde{\kappa}$-covering set of hypergraph *H*, we can determine a minimal set of DataNodes that satisfies the data blocks' available $\widetilde{\kappa}$ replicas requirements. The more DataNodes belong to the complementary set $\overline{\Gamma}$, i.e. out of the $\widetilde{\kappa}$-covering set, the more DataNodes can be powered off, thereby saving energy for the datacenter.

Hence, the objective function of HDFS variable $\widetilde{\kappa}$-covering problem is:

$$\begin{aligned} &Min\{\sum_{i=1}^{n} \Re_i\} \\ &\text{s.t.} \sum_{i=1}^{n} \left(a_{ij} \cdot \Re_i\right) \geq k_j \quad j = \{1, 2, ..., m\} \end{aligned} \tag{1}$$

where $\Re_i$=0 or 1, *i*={1, 2, 3, …, *n*}, denotes the status of the *i*th DataNode. If $\Re_i$=0, it indicates that the *i*th DataNode is powered off, otherwise, $\Re_i$=1 indicates that the *i*th DataNode is running.

4 Minimal Set Of HDFS Variable $\widetilde{\kappa}$-Covering Algorithm

Definition 4. *Let H=(X, E), if there exists a vertex set $\Gamma \subset X$, and it satisfies: $\left\|\Gamma \cap E_j\right\| \geq k_j$, $\widetilde{\kappa} = \{k_1, k_2, ..., k_j, ..., k_m\}$, we call Γ a $\widetilde{\kappa}$-transverse of H.*

Definition 4 shows that the variable $\widetilde{\kappa}$-covering set is the $\widetilde{\kappa}$-transverse of the hypergraph *H*, and searching one minimal variable $\widetilde{\kappa}$ covering set of HDFS is equivalent to solving the corresponding minimal $\widetilde{\kappa}$-transverse of hypergraph. It can be solved by the method of implicit enumeration. We describe the pseudo code of the algorithm as below:

Algorithm 1. Minimal $\widetilde{\kappa}$-transverse Calculation

Step 1. Initialization, let variable ω=1;

Step 2. Judge whether $\omega \leqslant 2^n$-1, if the inequality holds then do the next steps, else go to Step 5;

Step 3. Do binary transform of ω to $(\omega)_2$, where every bit of $\Re_i$ represents the state of the corresponding DataNode, including power on and power off.

Step 4. Calculate and judge $\sum_{i=1}^{n} (a_{ij} \cdot \Re_i) \geq k_j$, if the inequality satisfies all of *j*={1, 2, …, *m*} constraints then do the next step; else, execute *single bit shift operation* and back to Step2.

Step 5. Output $(\omega)_2$ and $\{\Re_i\}_{i=1}^{n}$。

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

Theorem 1. *The result of $\widetilde{\kappa}$ -transverse solved by the implicit enumeration is the minimum.*

Proof. From Step 4 of Algorithm 1, the "*single bit shift operation*" is employed in the search of implicit enumeration. The search order is from the smallest value of $\sum_{i=1}^{n}\Re_i$ to the larger ones. In other words, if a feasible solution is found with the minimum value of $\sum_{i=1}^{n}\Re_i$ that can satisfy all constraints, the search process will be terminated and the remaining feasible solutions with larger $\sum_{i=1}^{n}\Re_i$ value will be ignored. Therefore, the result of $\widetilde{\kappa}$ -transverse solved by the implicit enumeration is the minimum one. Fig 2 presents a demonstrated example of the process of *single bit shift operation*, which n=4.

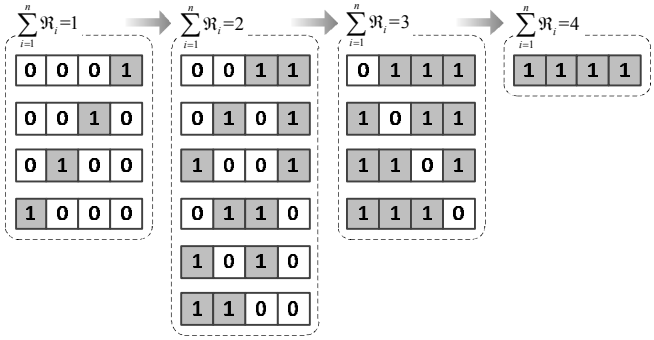

Fig. 2. The demonstrated example of the single bit shift operation

The time complexity analysis: Algorithm 1 searches the solution of the smallest $\sum_{i=1}^{n}\Re_i$, where the search space of Algorithm 1 is 2^n and the time complexity of the search is O(2^n). To satisfy all constraints, each solution needs to execute one time *sum operation*, i.e. $\sum_{i=1}^{n}(a_{ij}\cdot\Re_i)$ and m time *comparison operation*, i.e. $\sum_{i=1}^{n}(a_{ij}\cdot\Re_i)\geq k_j$, j={1, 2, …, m}. The total time complexity of algorithm 1 is O($mn2^n$). It shows that the implicit enumeration has high computational complexity and low computational efficiency. In this regard, we have improved the efficiency of the algorithm.

Theorem 2. *The sum of the node degree in $\widetilde{\kappa}$ -transverse must satisfy:*

$$\sum_{i=1}^{n} d_H(x_i)\big|_{\Re_i=1} \geq \sum_{j=1}^{m} k_j \qquad (2)$$

Proof. Reduction to absurdity: $\sum_{i=1}^{n} d_H(x_i)\big|_{\Re_i=1}$ is the total data blocks that were stored on all DataNode servers belonging to $\widetilde{\kappa}$ -*transverse*, i.e. DataNodes in working state. If condition (2) is violated, that is the total number of data blocks stored on the $\widetilde{\kappa}$ -*transverse* set is smaller than the lowest number of required replicas, it must violate the availability requirements. Therefore, Theorem 2 is proven.

We improve the solving process of implicit enumeration based on Theorem 2, which promotes computational efficiency.

Algorithm 2. Improved Implicit Enumeration

Step 1. Sort the set of nodes X of hypergraph H according to the node degree $d_H(x_i)$ from the largest one to the smallest one, then let the sorted node set be X^S;

Step 2. Start from the first element x_1^S and calculate $\sum_{i=1}^{r} d_H(x_i^S)$, until it satisfies Equation (2), record the value r;

Step 3. Run the $\widetilde{\kappa}$ -*transverse* algorithm from r.

The time complexity of Algorithm 2: The first step in Algorithm 2 is the *sorting operation* for the set of nodes X in hypergraph H, the time complexity using Merge-sort algorithm is O($n\log_2 n$). In the second step, a subset with $\sum_{i=1}^{n}\Re_i < r$ is eliminated from the solution space. Moreover, benefiting from the sort operation, the method of bisection can be used in the search process, then the time complexity is O($\log_2(2^n-2^r)$). For each solution, one time *sum operation* and m times comparison operation are also executed to examine the constraints. The total time complexity is O($mn\log_2(2^n-2^r)$), so that the time complexity of the improved Algorithm 2 is no more than O(mn^2).

The minimal $\widetilde{\kappa}$ -transverse that we have found is the minimal $\widetilde{\kappa}$ -covering set, which indicates the DataNodes needed to be running to satisfy the availability requirement of data blocks.

If there is more than one solution of the minimal $\widetilde{\kappa}$ -transverse with the same number of $\sum_{i=1}^{n}\Re_i$, we propose two strategies to further optimize the performance of HDFS:

(1) In the case of two or more $\widetilde{\kappa}$ -transverse sets which have the same $\sum_{i=1}^{n}\Re_i$, we calculate the sum of the node degrees $\sum_{i=1}^{n} d_H(x_i)\big|_{\Re_i=1}$, and then choose the minimum one to be the configuration scheme. In other words, starting with the same number of DataNode servers, the total amount of stored data blocks is minimized, which can provide more storage space for the newly arrival data, and also benefits for I/O operations in HDFS.

(2) If $\sum_{i=1}^{n}\Re_i$ and $\sum_{i=1}^{n} d_H(x_i)\big|_{\Re_i=1}$ are the same, we choose the configuration scheme that contains the least Racks occupied by the running data nodes. This can fur-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

ther reduce energy consumption by shutting off more idle Racks and their associated cooling equipment.

5 A Dynamic Runtime Energy Saving Strategy

Due to the dynamic nature of the Cloud, we need a runtime strategy to extend the hypergraph-based energy saving algorithm, so that it can adapt to the datacenters' dynamic storage processing. This is because, in real practice, it is infeasible to keep the running or sleeping data nodes unchanged at all times. First, data may arrive at any time and require processing. If we always store the newly arrived data in some settled data nodes, they will gradually become overloaded. Second, if the applications' requirements for data availability change, some of the replicas in the sleeping DataNodes will be needed accordingly. Too frequently wake up the sleeping DataNodes consumes extra energy, and is harmful for the device health. Hence, we need to switch the DataNodes from running to sleeping (and vice versa) periodically or when the users' requirements have changed. To solve this problem, we propose the following runtime strategy to control the working status of the DataNodes:

Initialize: Set the rotation cycle counter g for Datacenter, and set the index λ for DataNodes.

(1) At the beginning of a cycle g, let λ=1 be the DataNodes that have been identified in the variable $\widetilde{\kappa}$-transverse Γ_{cyc}^{g}, let λ=0 be the complementary transverse;

(2) In the cycle g, the newly arrived data is stored in the Γ_{cyc}^{g} according to the rack-aware storage strategy, then the sleeping DataNodes in the $\overline{\Gamma_{cyc}^{g}}$ do not need to be turned on at the same time;

(3) At the end of the cycle g, we firstly run the variable $\widetilde{\kappa}$-transverse algorithm for the data newly arrived in this cycle, then we can get a variable $\widetilde{\kappa}$-transverse $\Gamma_{new\text{-}data}^{g+1}$ for these data.

(4) After that, count the number of available replicas of each data block in the variable $\widetilde{\kappa}$-transverse $\Gamma_{new\text{-}data}^{g+1}$ set, and calculate $\tilde{\kappa}_{old\text{-}data}^{g+1}$.

$$\tilde{\kappa}_{old\text{-}data}^{g+1}=\tilde{\kappa}_{old\text{-}data}^{g}-\tilde{\kappa}(\Gamma_{new\text{-}data}^{g+1})$$

That is the number of old data replicas reduces from $\tilde{\kappa}_{old\text{-}data}^{g}$ to $\tilde{\kappa}_{old\text{-}data}^{g+1}$, because a part of the replica had been selected in the $\Gamma_{new\text{-}data}^{g+1}$.

(5) Calculate the variable $\tilde{\kappa}_{old\text{-}data}^{g+1}$-transverse in the range X- $\Gamma_{new\text{-}data}^{g+1}$. Select the optimal transverse from the $\tilde{\kappa}_{old\text{-}data}^{g+1}$-transverse set $\{\Gamma_{old\text{-}data}^{g+1}\}_u$ with the optimal condition of $\min \sum_{i\in\Gamma_{old\text{-}data}^{g+1}} \lambda_i$. It can give more sleeping DataNodes in the cycle g the chance to store the new data arriving in the period g+1 cycle, which is beneficial for storage load balancing of the datacenter. The final active set of data nodes in the next cycle g+1 is:

$$\Gamma_{cyc}^{g+1}=(\Gamma_{old\text{-}data}^{g+1})\cap(\Gamma_{new\text{-}data}^{g+1}).$$

With the above process, the newly arrived data and its replicas can also be optimally stored with the same rule of $\widetilde{\kappa}$-covering strategy. Minimal $\widetilde{\kappa}$-transverse reach the minimal number of active DataNodes to consume minimal energy. The rotation process provides enough opportunities to each data node to store newly arrived data and balance the storage load during the whole datacenter running period.

The elaborate strategy also brings another advantage, i.e. improving the DataNodes' I/O throughput and datacenter network performance. Minimal active DataNodes mean the required data are located in fewer places, so that the number of data retrieving operations among DataNodes will be reduced, which accelerates the data read/storage process. The routing in datacenter networks becomes easier and it generates fewer overhead messages, both of which are beneficial to improve the transmission performance directly.

6 Experiment and Performance Evaluation

This section presents the details of the conducted experiments to evaluate the performance of our proposed algorithm.

6.1 Prototype Implementation and Evaluation Strategy

We have implemented a prototype system of our proposed strategy in the local experimental datacenter, which is composed of six Racks with nine physical servers serving as DataNodes (from now on, we will call them DataNode servers) in each Rack. The specification of the DataNode servers is listed in Table 1. The datacenter network is implemented on the Fat-tree topology to realize the communication among servers. The Hadoop distributed file system is installed in the DataNode servers to store the data sets of different applications. The algorithm in the strategy is implemented in the name node where the information of all applications' data sets is saved. Given the requirement of replicas numbers for each application, our algorithm will calculate the minimum set of DataNodes (i.e., the $\widetilde{\kappa}$-transverse) based on where the required replicas are stored. Then the unused DataNodes will be turned off to save energy. We also implement the dynamic mechanism to run the strategy periodically at runtime in order to cope with the requirement change and maintain load balance. In order to implement the proposed strategy in large datacenters, the component for managing application requirements need to be developed and deployed in different name nodes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

Table 1
CONFIGURATION OF THE DATANODE SERVER

Parameters	Values
CPU	Intel 2.7GHz/2 cores
OS	Ubuntu12.04
Hard Disk	SATA3.0 (6 Gbps)
Java Version	1.6 for Linux
Hadoop Version	1.1.2
Idle Power Consumption	168.2W
Max Power Consumption	344.8W

In the experiments, we will evaluate the performance our proposed algorithm from four different aspects, namely, data availability, load balance, energy consumption and network performance of the datacenter. To this end, we deploy three typical applications in our local datacenter and conduct experiments with the prototype system to test the data availability and load balance and measure the energy consumption of the datacenter. Due to the limited scale of our local datacenter, the network performance for using our strategy in large datacenters cannot be well evaluated. Hence we conduct simulations in the NS-2 platform to evaluate the performance of data reading/transmission process of our proposed strategy, which has been widely used for large datacenter network performance evaluation [33]. Please note that all data points are averaged over 300 simulation rounds.

6.2 Data Availability Evaluation

Data availability is the prime aspect for datacenter storage. In this subsection we conduct a numerical experiment for our proposed energy-efficient variable $\widetilde{K}$ -covering algorithm from the perspective of data availability. The evaluation is focused on three different levels, data blocks, files and applications, in which we compare our algorithm with the original rack-aware storage strategy in HDFS, as well as the storage strategy based on the fixed covering set optimization [17][22].

Since our proposed strategy is intended to be applied in cloud datacenters where various applications are deployed, we selected three typical applications in our experiment to emulate the environment of a real world cloud datacenter. The selected three applications have different features: 1) MapReduce application with Wordcount program tasks, which are computation intensive, 2) general-purpose cloud storage application with file retrieve and update tasks, which are I/O intensive, and 3) online video application for public users with video streaming tasks, which are bandwidth intensive. The corresponding data sets for these applications are generated as follows:

Appl_1: We generate 32 data sets for the MapReduce application. Each data set is as {Total Size=0.064G~0.64G, Num of files=1};

Appl_2: We generate 16 data sets for the cloud storage application. Each data set is as {Total Size=1G, Num of files=2};

Appl_3: We generate 4 data sets for the video streaming application. Each data set is as {Total Size=4G, Num of files=4}.

Hence, the total size of the generated data is around 48G. Furthermore, the data is stored according to the rack-aware storage strategy, i.e. splitting the files into a series of 64M data blocks and storing three replicas of each data block on different DataNode servers. Therefore, we have 640 different data blocks with 1960 replicas (in total, 2560) randomly stored in 54 DataNode servers.

We set the availability requirements of data sets according to the applications' access requirements, which are as follows: the data blocks of 32 *Appl_1* are required to keep three replicas at the same time, including the original data block; the data blocks of 16 *Appl_2* are required to keep two replicas at the same time; and the data blocks of four *Appl_3* are required to keep one replica at the same time.

We conducted experiments to compare our variable $\widetilde{K}$ -covering set algorithm with two other representative strategies, i.e. un-optimized strategy and fixed covering set strategy [17] [22]. Figs. 3, 4 and 5 depict the relationship between the total number of power-off DataNode servers and the available percentage at three different levels, data blocks, files and applications respectively. With the increment of the number of power-off DataNodes, the available percentages at all three levels are decreased accordingly. Compared to the available percentage of data blocks, the available percentages of files and applications executable rate drop much more sharply. Specifically, for the un-optimized strategy, DataNode severs are randomly turned off. In this case, the availability requirements of data blocks are easily violated, and it also disrupts the data integrity of files and the whole application. For the fixed covering set strategy, we set the fixed covering rate as two for all kinds of data blocks to meet the availability requirements. In this case, only two data nodes can be safely turned off without violating the data availability requirements. In contrast, by using our proposed variable $\widetilde{K}$ -covering set algorithm, we can safely turn off up to 10 DataNode servers, while all types of data blocks, files, and applications can still meet the availability requirements.

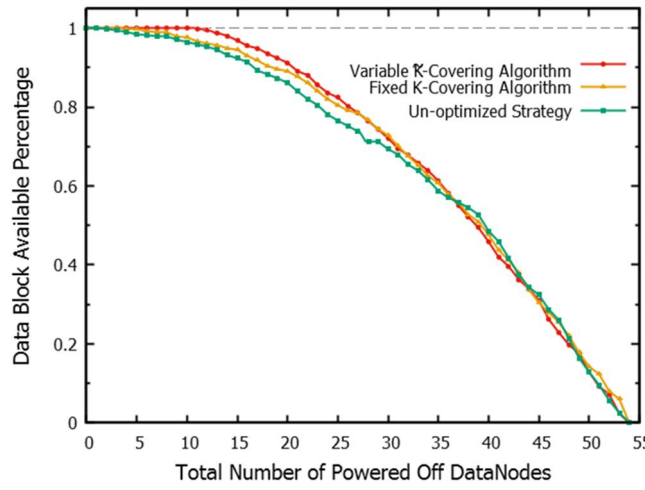

Fig.3. Data block available percentage with shutting off different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

number of DataNodes

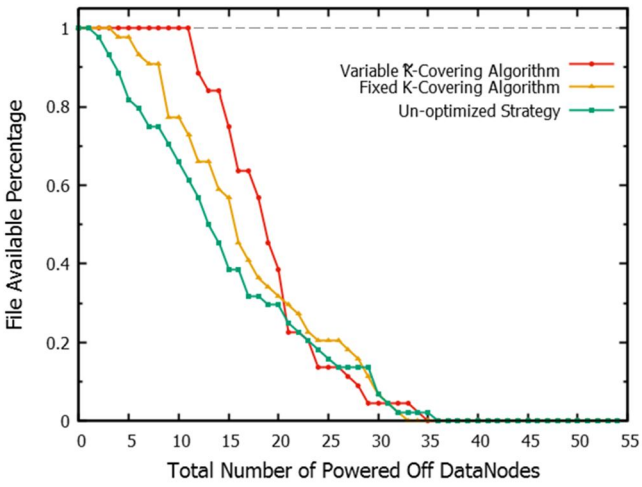

Fig.4. File available percentage with shutting off different number of DataNodes

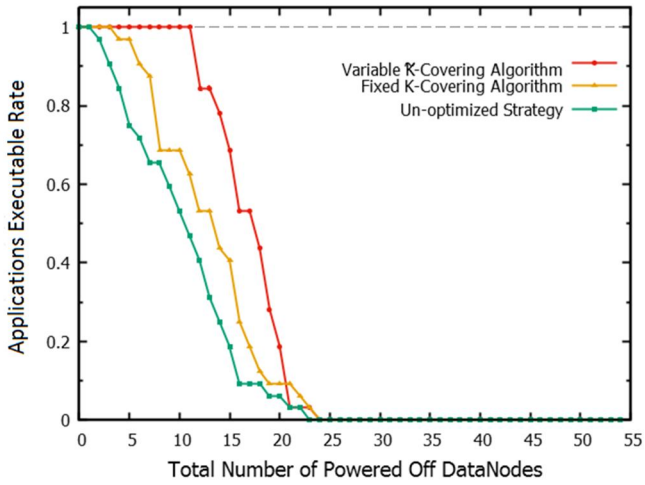

Fig.5. Applications executable rate with shutting off different number of DataNodes

6.3 Storage Load Balancing Experiments

In this sub-section, we studied the performance of our proposed algorithm from the perspective of load balance and compare it with our selected benchmark. In our experiments, the initial HDFS stored 640 different data blocks, where the replica numbers and availability requirements of data blocks are set to be the same as in Section 6.2. In order to evaluate the performance of our strategy under the different data arrival rates, the numbers of newly arriving data blocks in each cycle are set as 32, 64, 128, and 256 respectively. For all the generated data blocks, they only need to have one replica by application requirements to ensure data availability.

The experiment results showed, by using the dynamic rotation strategy described in Section 5, each DataNode can be turned on at least once within every four consecutive cycles. In contrast, without the dynamic rotation strategy, the powered-off DataNodes cannot be turned on again. As long as the new data successively arrive, the running DataNode servers became heavily overloaded. We use the standard deviation δ of total data blocks stored in each DataNode to measure the storage load imbalance. The results of our proposed algorithm and the selected benchmark under different data arrival rates are shown in Fig. 6. When the data arrival rate is low, i.e. 32 data blocks for each cycle, $\Delta\delta=\delta_{un}-\delta_{lb}=1.4165$ in the fifth round, where δ is the standard deviation of the number of data blocks in each DataNode server, δ_{un} is the un-optimized case and δ_{lb} is the load balancing case with the dynamic rotation strategy. When the data arrival rate is high, i.e. 256 new data blocks in each cycle, $\Delta\delta=13.2087$ in the fifth round.

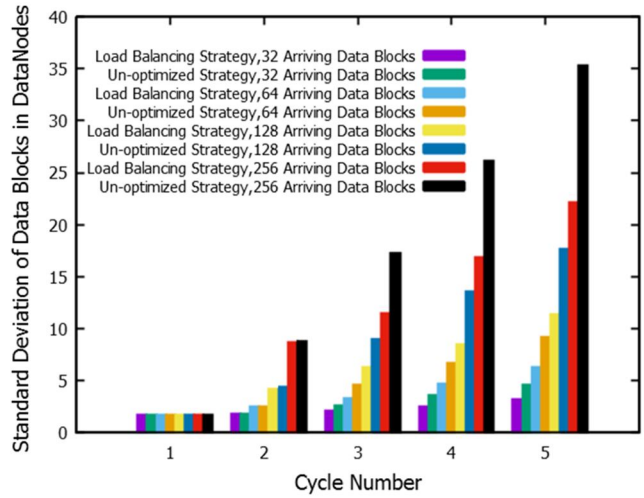

Fig.6. The comparison of DataNodes storage imbalance

6.4 Energy Consumption Analysis for HDFS

From the experiments presented in Sections 6.2 and 6.3, we observed that more DataNodes could be powered off without violating the data availability requirements by using the variable $\widetilde{\kappa}$-covering algorithm. In this section, we ran the well-known Hadoop test program – Wordcount – to measure the performance of our proposed algorithm and the selected benchmark from the perspective of energy saving. We choose the MapReduce application for this experiment, because it is both computation and data intensive, which will heavily use the CPU, network and storage of the datacenter.

We ran nine groups of WordCount programs in total at different scales, where the number of search data blocks is increased from four to 1024. In the experiments we still set different availability requirements for different data blocks, where the ratios of having one replica, two replicas and three replicas are set as 4:3:1. Therefore, the variable $\widetilde{\kappa}$-covering algorithm used this ratio to configure the DataNodes, compared to the selected benchmark of the fixed k-covering set algorithm, where k is set to three to satisfy the maximum number of replicas – three replicas. Fig. 7 shows how the number of powered-off DataNodes and the average CPU executing time are changed accordingly to execute the Wordcount programs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

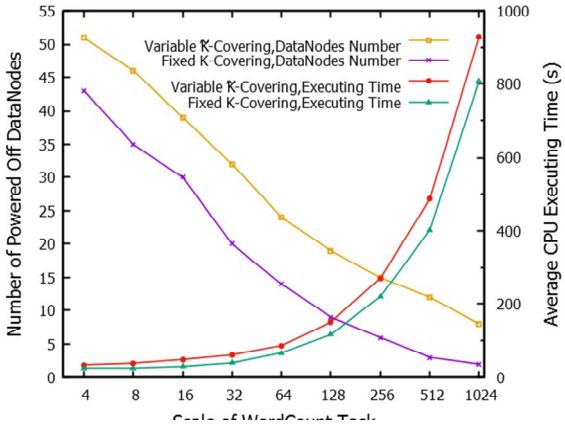

Fig.7. The number of powered-off DataNodes and the average execution time with different scale Wordcount tasks

As indicated in Fig. 7, the number of DataNodes that can be powered off by using the proposed variable $\widetilde{K}$-covering algorithm is larger than the fixed 3-covering set algorithm at all times, no matter in the case of applications searching fewer data blocks or searching more data blocks, and the superiority is stable. This is because the variable $\widetilde{K}$- transverse can accurately represent each data block's availability requirements and redundant DataNodes can be powered off.

Fig. 7 also shows the average CPU time for executing the *Map* programs. As we can see from the figure, the average CPU time by using the variable $\widetilde{K}$-covering set algorithm becomes slightly longer than the fixed 3-covering set algorithm after the scale of the WordCount program reaches 64. This is caused by the WordCount programs being assigned to fewer servers for their processing since more DataNodes are powered off by using the variable $\widetilde{K}$-covering set algorithm. However, with the number of programs increasing, the computation will be shared by more data nodes, and then the gap of CPU time will gradually decrease.

Power and energy consumption: As we know, the power consumption of servers is composed of ground power consumption and dynamic power consumption. The ground power consumption is the power consumed by the servers in the idle state, denoted as P_{idle}, and the dynamic power consumption is closely related to the usage of CPUs, fans, etc. The total energy consumption of a server can be represented by the following nonlinear model [27]:

$$P_{sever} = P_{idle} + (P_{max} - P_{idle}) \times (2u - u^r) \quad (3)$$

where P_{sever} denotes the total power consumption of a server, P_{max} denotes the maximum power consumption of a server operating at its full capacity, u denotes the CPU utilization, r denotes a calibration coefficient which can minimize the variance between theoretical and actual power consumptions. The value of r is different for different servers, which can be obtained through experiments. By obtaining the data from thousands of servers, Equation (3) can accurately fit the actual servers' power consumption with only 1%~5% error [17]. In addition, extensive research [28]-[31] has shown that ground power consumption of servers accounts for 50%~70% of the maximum power consumption.

In our experiments, we denoted the energy consumption of a server as Eg_i, and the energy consumption of the whole datacenter as Eg_{total}. They can be computed by the following equations respectively:

$$Eg_i = \int_{T_i^{star}}^{T_i^{end}} \Re_i(t) \cdot P_i dt \quad (4)$$

$$Eg_{total} = \sum_{i \in active\ servers} \left(\int_{T_i^{star}}^{T_i^{end}} \Re_i(t) \cdot P_i dt \right) \quad (5)$$

where P_i denotes the total power consumption of the ith server, $\Re_i(t)$ denotes the status of server i, $\Re_i(t)$ =0 indicates that the ith DataNode at time t is powered off, otherwise, $\Re_i(t)$ =1 indicates that the ith DataNode at time t is running. Please note that in our experiments, we only considered the energy consumption from servers. In a real-world datacenter, the energy consumption of corresponding network devices and cooling devices will also be reduced as the number of active servers decreases. As a result, the overall energy consumption of the whole datacenter can be further reduced than the result we are shown in the experiments.

By simply observing Equation (5), we noticed that two key variables can affect the total energy consumption of the datacenter, one is t_i and the other is the number of running servers. Although the variable $\widetilde{K}$-covering algorithm increases the execution time on each active DataNode and results in more energy consumption on them, the extra energy consumption is still not comparable to the energy consumption of turning on more DataNodes. Thus, the total energy consumption of the datacenter can be reduced, which well matches the result shown in Fig 8.

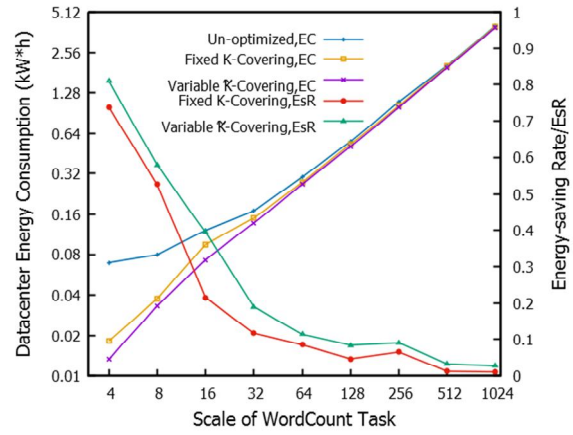

Fig.8. Energy-saving analysis in executing WordCount programs in different data scales

The experimental results and the numerical analysis of the energy consumption of executing different types of WordCount programs are shown in Fig 8. It demonstrated that the energy consumption with the variable $\widetilde{K}$-covering algorithm is always the minimum in all nine types of WordCount program. For the execution of four data block WordCount programs, our proposed algo-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

rithm's energy-saving rate is 7.08% more than the fixed 3-covering set algorithm. In the case of executing WordCount programs on a large scale, 1024 data blocks, the energy-saving rate is 2.76% for our variable $\tilde{K}$-covering algorithm, which is still 1.65% more than the fixed 3-covering algorithm.

6.5 Network Performance Evaluation for the HDFS Datacenter

Datacenter network performance is one of critical factors that directly affect data reading and storing speeds. As analyzed in Section 5, the variable $\tilde{K}$-covering algorithm can optimize allocation for data storage, which can not only reduce the energy consumption of a datacenter, but also reduce transmission source/destination and path in datacenter networks, because there are fewer DataNodes intensively supplying data. Therefore, the algorithm can effectively improve datacenter network performance. To validate this advantage, the following experiments are carried out.

We employ WordCount in 512 data blocks as the calculation example and run the MapReduce program. First, the *Map* program of each data block is run at its local DataNode server, and then the results of *Map* programs are shuffled to other servers through the data center network to run the *Reduce* programs and generate the final WordCount results. Table 2 shows the specification of the datacenter network used in our experiments.

Table 2
CONFIGURATION OF DATACENTER NETWORK

Parameter	Value
Data Center Network Structure	Fat-Tree with 6 ports Switch
Buffer size per output port	500 packets
Traffic generator	CBR
Link speed	Access Level: 200Mbps Aggreg and Core: 1000Mbps
One Map Program Result	324.57KB
Total Map Program Results	270.049MB
CBR ratio	2Mbps
Packet size	1000 Bytes

The average end-to-end delay of data packets is an important QoS factor to measure network performance. Fig. 9 shows the maximum and minimum transmission delay with different storage strategies. For the data nodes configured by the variable $\tilde{K}$-covering algorithm, the maximum delay is 0.06128s. Compared with the fixed 3-covering algorithm's maximum delay 0.06152, the proposed algorithm reduced 0.39% of the end-to-end delay. Meanwhile, the minimum delay of variable $\tilde{K}$-covering algorithm is 0.06024s, which is 0.29% less than the fixed 3-covering algorithm's 0.060416s. This is because our algorithm can power off more DataNode servers than the fixed 3-covering algorithm for running *Map* programs. This is equivalent to reducing many simultaneous data sources for transmitting *Map* results, and the number of switches and communication links used for service communication is also reduced. Therefore, the "incast" congestion, one of the typical problems in datacenter networks with TCP transmission [32], can be effectively avoided.

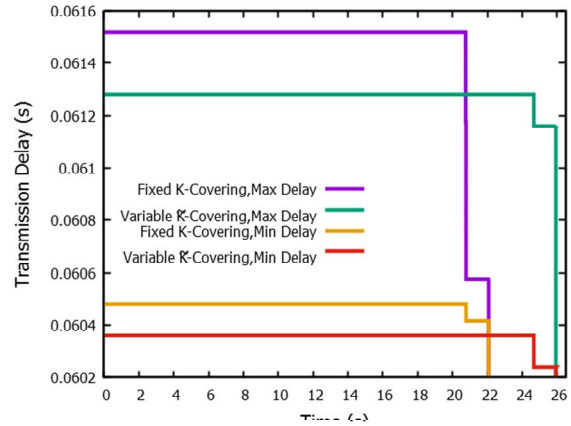

Fig.9. The comparison of average end-to-end transmission delay of data packets.

Fig. 10 shows the statistical analysis of the bandwidth utilization over 300 experiments. The average bandwidth utilization of the Reducer port is 83156.61*kbps* for using the variable $\tilde{K}$-covering algorithm. For the fixed 3-covering algorithm, its bandwidth utilization is 100703.56*kbps*, which is 17.42% higher than our proposed algorithm. Although both algorithms did not exceed the upper limit of total bandwidth 200Mbps, we should be aware that the experiment is run under the light-load case. When the number of hosted applications increases, the bandwidth utilization will increase rapidly at the Reducer port by using fixed 3-covering algorithm. Therefore, it could cause network congestion if the data transmission exceeds the upper limit of bandwidth. We also ran a heavy-load experiment, and analyzed the packet loss rate to evaluate network congestion next.

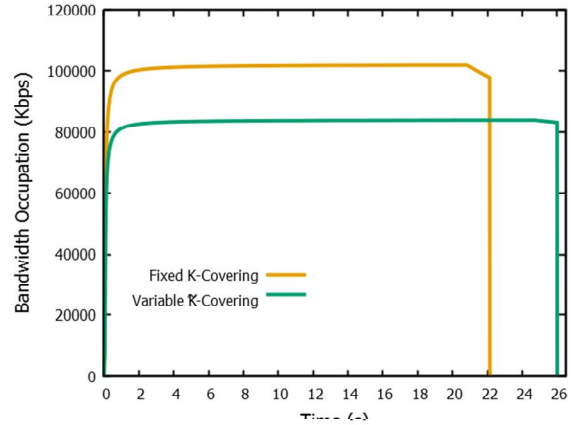

Fig.10. The comparison of bandwidth occupancy at Reducer port

We also evaluated the network performance by counting the packet loss rate in a heavy-load scenario created by increasing the transmission load of the network. We ran three independent groups of MapReduce programs at the same time, where the parameters of programs and network remain the same as in Section 6.4. For each program, calculating covering set and transmitting intermediate data are independent from one another. The experiment's results show that, in this heavy-load scenario, the datacenter network using the fixed 3-covering algorithm

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2723004, IEEE Transactions on Parallel and Distributed Systems

has the fault of dropping packets, and the packet loss rate is about 1.9%. But this problem does not happen in the variable $\widetilde{\kappa}$-covering algorithm, where the packet loss rate is 0. The experiment's results are shown in Table 3.

Table 3
PACKET LOSS RATE IN HEAVY-LOAD SCENARIO

	3-covering algorithm			$\widetilde{\kappa}$ -covering algorithm		
	Job_1	Job_2	Job_3	Job_1	Job_2	Job_3
Send packets	270049	270049	270048	270049	270048	270049
Lost packets	5158	5160	5116	0	0	0
Loss rate %	1.91%	1.91%	1.89%	0	0	0

All the experiments conducted in this sub-section for evaluating three QoS parameters, namely average end-to-end delay, bandwidth utilization and packet loss rate, represent a consistent result that the variable $\widetilde{\kappa}$-covering algorithm can effectively improve datacenter network performance.

6.6 Discussion

The experiments conducted in this work were run on a local datacenter. However, to deploy the proposed strategy into large datacenters, we need to consider certain implementation issues, e.g., scalability and overhead. In terms of scalability, as our strategy needs to be implemented on NameNode, its efficiency is dependent on the volume of data stored in the DataNodes of a given NameNode. Theoretically, a Hadoop cluster could have thousands of DataNodes under one NameNode, but in practice large cloud datacenters normally have small scale DataNode clusters to avoid the performance bottleneck of the NameNode. Our strategy works independently on different NameNodes, it is thus scalable and can be implemented in large datacenters. The main overhead of our strategy is running the hypergraph-based algorithm. Since this can be performed offline, the overhead of running the algorithm is negligible comparing to the workloads of applications deployed in the datacenter.

7 CONCLUSION AND FUTURE WORK

In this work, we investigated energy saving methods for datacenters, and presented a novel variable $\widetilde{\kappa}$ covering algorithm, which can flexibly adapt to various availability requirements from different types of data. It extends the existing fixed number coverage algorithm and optimizes the storage in HDFS. Specifically, we 1) proposed a hypergraph-based storage model that precisely depicts the many-to-many relationship of files and data blocks located in DataNodes and Racks, 2) formulated the variable $\widetilde{\kappa}$ -covering problem to model the variable data replicas required in HDFS storage, and 3) developed the novel $\widetilde{\kappa}$ -transverse of hypergraph algorithm to solve it. We also extended the algorithm to a dynamic runtime energy saving strategy, which achieves load balancing of DataNodes in the whole datacenter running period.

Extensive experimental results showed that 1) the proposed algorithm can turn off more DataNodes than the existing fixed number coverage algorithm so that more energy can be saved without violating the data availability requirements (Section 6.2); 2) with the continuous arrival of data, load balance can be maintained among the DataNodes (Section 6.3); 3) the energy conservation using our proposed strategy is measured in our local datacenter (Section 6.4); and 4) the strategy can also improve the datacenter's network performance (Section 6.5).

In the future, based on the prototype that we have built in our local datacenter, we will extend our approach to a real-life system, e.g. a large-scale datacenter and fine tune its performance.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China No.61571324, the Natural Science Foundation of Tianjin No.16JCZDJC30900, the National high technology research and development program (863 Program) No.2015AA050202, and the National program of international S&T cooperation No.2013DFA11040. Professor Zomaya's work is supported by an Australian Research Council Discovery Grant (DP 130104591).

REFERENCES

[1] Subashini S and Kavitha V, "A survey on security issues in service delivery models of cloud computing," *J. Network and Computer Applications,* vol 34, no.1 ,pp. 1-11, Jan. 2011.

[2] Prasad A S and Rao S, "A Mechanism Design Approach to Resource Procurement in Cloud Computing," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 17-30, Jan. 2014.

[3] Armbrust M, Fox A, Griffith R, Griffith R, Joseph A.D., Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, and Zaharia M, "Above the clouds: A Berkeley view of cloud computing," *Eecs Department University of California Berkeley,* vol. 53, no. 4, pp. 50-58, Feb. 2009.

[4] Deng W, Liu F, Jin H, Li B, Li D, "Harnessing renewable energy in cloud datacenters: Opportunities and challenges," *IEEE Network,* vol. 28, no. 1, pp. 48-55, Jan. 2014.

[5] Wired Real Estate Group, "Data Center Power Innovation to Reduce TCO," [Online]. Available: http://wiredre.com/data-center-power-innovation-to-reduce-tco/. [Accessed 20 August 2015].

[6] Bashroush R, Woods E and Noureddine A, "Data Center Energy Demand: What Got Us Here Won't Get Us There," *IEEE Software,* vol. 33, no. 2, pp. 18-21, Mar. 2016.

[7] Borthakur D, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website,* vol. 11, no. 11, pp. 1-10, Jan. 2007.

[8] S. Ghemawat, H. Gobioff, and S. Leung Garcia H, "The Google file system", *Proc. ACM Symp. Oper. Syst. Principles,* pp. 29-43, Oct. 2003.

[9] Li W, Yang Y and Yuan D, "A Novel Cost-Effective Dynamic Data Replication Strategy for Reliability in Cloud Data Centres," *Proc. Dependable, Autonomic and Secure Computing (DASC),* pp. 496-502, Dec. 2011.

[10] Li W, Yang Y and Yuan D, "Ensuring Cloud data reliability with minimum replication by proactive replica checking," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1494-1506, May. 2016.

[11] Di, Sheng, Daishi Kondo, and Franck Cappello. "Characterizing cloud applications on a Google data center" In 42nd International Conference on Parallel Processing (ICPP), 2013, pp. 468-473. IEEE, 2013.

[12] Afianian A, Nobakht S S and Ghaznavi-Ghoushchi M B, "Energy-Efficient Secure Distributed Storage in Mobile Cloud Computing[C], *Proc. Iranian Conference on Electrical Engineering (ICEE),* pp. 740-745,

May.2015.

[13] Yun D, Lee J, Yi Y., "Research in green network for future internet", *J. KIISE,* vol. 28, no. 1, pp. 41-51, 2010.

[14] Cheng Z, Luan Z, Meng Y, Xu Y and Qian D, "Erms: an elastic replication management system for hdfs," *Proc. IEEE Cluster Computing Workshops (CLUSTER WORKSHOPS)*, pp. 32-40, Sept. 2012.

[15] Abad C L, Lu Y and Campbell R H, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling" *Proc. IEEE Cluster Computing (CLUSTER),* pp. 159 -168, 2011.

[16] Maheshwari N, Nanduri R and Varma V, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 119-127, Jan. 2012.

[17] Liao B, Yu J, Zhang T, Guo B, Sun H and Ying C, "Energy-efficient algorithms for distributed storage system based on block storage structure reconfiguration," *J. Network and Computer Applications,* vol. 48, no. 1, pp. 71-86, Oct. 2014.

[18] Van H N, Tran F D and Menaud J M, "Performance and Power Management for Cloud Infrastructures" *Proc. IEEE Cloud Computing (CLOUD),* pp. 329-336, 2010.

[19] Taheri, J., Zomaya, A.Y., Kassler, A., "vmBBThrPred: A Black-Box Throughput Predictor for Virtual Machines in Cloud Environments," *European Conference on Service-Oriented and Cloud Computing. Springer International Publishing*, 2016: 18-33.

[20] Harnik D, Naor D and Segall I, "Low power mode in cloud storage systems" *Proc. IEEE Int. Parallel Distrib. Process. Symp.,* pp. 1-8, 2009.

[21] Kim J, Chou J and Rotem D, *Energy Proportionality and Performance in Data Parallel Computing Clusters*. Scientific and Statistical Database Management. Springer Berlin Heidelberg, pp. 414-431, 2011.

[22] Kim J, Chou J and Rotem D, "iPACS: Power-aware covering sets for energy proportionality and performance in data parallel computing clusters," *J. Parallel Distributed Computing,* vol. 74, no. 1, pp. 1762-1774, 2014.

[23] Yazd S A, Venkatesan S and Mittal N, "Boosting energy efficiency with mirrored data block replication policy and energy scheduler," *Acm Sigops Operating Systems Review,* vol. 47, no. 2, pp. 33-40, 2013.

[24] Leverich J, Kozyrakis C. "On the energy (in) efficiency of hadoop clusters," *ACM SIGOPS Operating Systems Review*, Vol.44, No.1, pp. 61-65, 2010.

[25] Dan A S and Djeraba C, *Graphs and Hypergraphs*. North-Holland, pp. 1307-1315, 1973.

[26] Leverich J and Kozyrakis C, "On the energy (in) efficiency of Hadoop clusters," *Acm Sigops Operating Systems Review,* vol. 44, no. 1, pp. 61-65, 2010.

[27] Fan X, Weber W D and Barroso L A, *Power provisioning for a warehouse-sized computer*. ACM SIGARCH Computer Architecture News. ACM, pp. 13-23, 2007.

[28] Raghavendra R, Ranganathan P, Talwar V, Wang Z and Zhu X, "No power struggles: coordinated multi-level power management for the data center," *Acm Sigops Operating Systems Review,* vol. 36, no. 1, pp. 48-59, 2008.

[29] Yang T, Lee Y, Zomaya A, "Collective Energy-Efficiency Approach to Data Center Networks Planning," *IEEE Trans. Cloud Computing*, DOI 10.1109/TCC.2015.2511732.

[30] Verma A, Ahuja P and Neogi A, *pMapper: power and migration cost aware application placement in virtualized systems*. Middleware 2008. Springer Berlin Heidelberg, pp. 243-264, 2008.

[31] Gandhi A, Harchol-Balter M, Das R, et al. "Optimal power allocation in server farms," *ACM SIGMETRICS Performance Evaluation Review*. ACM, vol. 37, no. 1, pp. 157-168, 2009.

[32] Stephens B, Cox A L, Singla A, et al. "Practical DCB for improved data center networks" *Proc. IEEE INFOCOM*, pp.1824-1832, 2014.

[33] Benson T, Anand A, Akella A, et al. "Understanding data center traffic characteristics" *ACM SIGCOMM Computer Communication Review*. ACM, vol. 40, no. 1, pp. 92-99, 2010.