Cluster Comput (2015) 18:1465–1480
DOI 10.1007/s10586-015-0495-z

Optimizing data placement in heterogeneous Hadoop clusters

Runqun Xiong[1] · **Junzhou Luo**[1] · **Fang Dong**[1]

Received: 13 March 2015 / Revised: 17 September 2015 / Accepted: 19 September 2015 / Published online: 1 October 2015

Abstract Data placement decision of Hadoop distributed file system (HDFS) is very important for the data locality which is a primary criterion for task scheduling of MapReduce model and eventually affects the application performance. The existing HDFS's rack-aware data placement strategy and replication scheme are work well with MapReduce framework in homogeneous Hadoop clusters, but in practice, such data placement policy can noticeably reduce MapReduce performance and may cause increasingly energy dissipation in heterogeneous environments. Besides that, HDFS employs an inflexible replica factor acquiescently for each data block, which will give rise to unnecessary waste of storage space when there is a lot of inactive data in Hadoop system. In this paper, we propose a novel data placement strategy (SLDP) for heterogeneous Hadoop clusters. SLDP adopts a heterogeneity aware algorithm to divide various nodes into several virtual storage tiers (VSTs) firstly, and then places data blocks across nodes in each VST circuitously according to the hotness of data. Furthermore, SLDP uses a hotness proportional replication to save disk space and also has an effective power control function. Experimental results on two real data-intensive applications show that SLDP is energy-efficient, space-saving and able to improve MapReduce performance in a heterogeneous Hadoop cluster significantly.

✉ Runqun Xiong
rxiong@seu.edu.cn

Junzhou Luo
jluo@seu.edu.cn

Fang Dong
fdong@seu.edu.cn

1 School of Computer Science and Engineering, Southeast University, Nanjing 211189, People's Republic of China

Keywords Hadoop cluster · HDFS · Data placement · Heterogeneous · Replica

1 Introduction

With the recent emergence of cloud computing [1] based services on the Internet (such as e-commerce websites, online video and social networks), enormous volumes of data or called "big data" [2] is being generated by these services around us at all times. As a potential gold mine for high economic and social value, big data is creating a culture in which business and IT leaders must join forces to realize value from all kinds of data. But escalating demand for insights requires not only analytic capabilities and skills but also optimal processing power, which means that to extract meaningful value from big data, it needs a fundamentally new approach to architecture, tools and practices.

The MapReduce [3] model popularized by Google is very attractive for ad-hoc parallel processing of arbitrary big data sets as mentioned above. MapReduce breaks a computation into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. Its popular open-source implementation, Hadoop [4], was developed primarily by Yahoo!, where it runs jobs that produce hundreds of terabytes of data on a large amount of cores. The Hadoop runtime system provides a distributed file system (HDFS) [5] and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm. Apart from web data-intensive applications, scientific data-intensive applications (e.g., seismic simulation and natural language processing) take considerable benefits from the Hadoop system [6].

The MapReduce model was conceived with the principle that "moving computation is much cheaper than moving data" [7]. Data locality determined by the data placement strategy of HDFS is a primary criterion for MapReduce jobs scheduling on Hadoop clusters, and eventually affects the applications performance. To the best of our knowledge, however, three main problems are observed in current data placement strategy of HDFS:

Firstly, and most importantly, the existing data placement strategy of HDFS [8] may noticeably reduce MapReduce performance in heterogeneous environments. Essentially, Hadoop system is designed to run on a set of homogeneous nodes. Such a homogeneous configuration of Hadoop allows the MapReduce framework to effectively schedule computing tasks on an array of nodes where data blocks are residing, leading to a high aggregate bandwidth across the entire Hadoop cluster. But, in practice, the homogeneity assumptions do not always hold [9]. On the one hand, Hadoop clusters scale computation power, storage capacity and I/O bandwidth by simply adding commodity servers, and consist of hundreds or thousands of machines. Therefore, machine failures are common. It needs frequent equipment renewal, and there are multiple generations of hardware in a cluster. On the other hand, with virtualization techniques being widely used in cloud computing data centers [10], massive virtual machines with variations in performance are deployed to process large amounts of data-intensive applications. In such a heterogeneous Hadoop cluster, a high performance node is capable of processing local data faster than a low performance node. After the fast node finished processing data residing in its local disk, the node has to handle unprocessed data from a remote slow node. The overhead of transferring unprocessed data from slow nodes to fast peers is high if the amount of moved data is huge. That means Hadoop's homogeneity assumptions will lead to incorrect and excessive speculative executions, and degrade MapReduce performance in heterogeneous environments.

Secondly, according to the current data placement strategy of HDFS, a Hadoop cluster has to keep all nodes active to ensure data availability, and even if significant periods of inactivity are observed during data processing, the need for data availability prohibits the shutting down of idle nodes. Such data placement strategy may negatively impact energy efficiency for lack of energy-proportional ability [11].

In addition, each data block is replicated to ensure data availability in case of connectivity failure to nodes or even complete racks, and Hadoop implements a 3-way rack-aware block replication policy for the files that are stored on HDFS. For the default replica factor of three, HDFS's replica placement strategy is to put one replica of the block on one node in the local rack, another on a different node in the same rack, and the third on a node in some other rack. Such replication policy will result in an unnecessary waste of storage space when there is a lot of inactive data.

In this paper, we address the above challenges by proposing a novel data placement strategy for HDFS in heterogeneous Hadoop clusters to improve MapReduce performance, reduce power consumption and enhance the efficiency of storage space.

Our Contributions The main contributions of this paper are summarized as follows:

- We propose a heterogeneity aware algorithm (Haag) for nodes in Hadoop clusters, which can divide various nodes into several virtual storage tiers (VSTs) according to the performance properties of each node.
- We present a hotness-proportional replication policy (HP) to determine the replica factor of each data block based on its hotness (i.e., the hotness of a data block is higher its replica factor will be larger).
- We design a snakelike data placement strategy (SLDP) by integrating Haag and HP to distribute data blocks across DataNodes in each VST circuitously like a slowly-swimming snake shape, which can place hot data blocks into the high performance VSTs and those cold data into the low performance VSTs.
- We implement a power control scheme to endow with energy-proportional capacity for Hadoop clusters based on the data blocks distribution results from SLDP, to reduce the power consumption without affecting MapReduce performance.
- We realize our work on Apache Hadoop-2.3.0 release and evaluate it with two real data-intensive applications. Experimental results show that SLDP is energy-efficient, space-saving and able to improve MapReduce performance in heterogeneous Hadoop clusters.

The remainder of the paper is organized as follows. Section 2 introduces the data placement and replication management in HDFS and the need for power control in Hadoop. Next, we describe the detailed design and implementation of a Hadoop system architecture that using SLDP and its relevant algorithms in Sect. 3. Section 4 evaluates our work with two real data-intensive applications from three perspectives. Section 5 reviews the related work and Sect. 6 concludes the paper with future research directions.

2 Background and motivation

2.1 Data placement and replication policy in HDFS

HDFS stores file system meta-data and application data separately. As in other distributed file systems, like PVFS [12], Lustre [13], and GFS [14], HDFS stores meta-data on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols. Unlike Lustre and PVFS, the DataNodes in HDFS do not rely on data protection mechanisms such as RAID to make the data durable. Instead, like GFS, the file content is replicated on multiple DataNodes for reliability. Each file stored on HDFS is split into smaller chunks of even-sized called data blocks and these blocks are distributed equally across the cluster. Each block is replicated to ensure data availability in case of connectivity failure to nodes or even complete racks. And Hadoop implements a rack-aware data block replication policy for the data that is stored on HDFS. For the default replica factor of three, HDFS's replica placement policy is to place the first replica on the node where the writer is located. The rest are placed on random nodes with restrictions that no more than one replica is placed at any one node and no more than two replicas are placed in the same rack, if possible.

Unlike conventional file systems, HDFS provides an API that exposes the locations of file blocks. This allows applications like the MapReduce framework to schedule a task to where the data are located and thus improve the read performance. It also allows an application to set the replica factor of a file severally. By default, a file's replica factor is three in HDFS. For critical files or files which are accessed very frequently, having a higher replica factor improves tolerance against faults and increases read bandwidth.

In a word, the localities of blocks and their replicas are critical to HDFS reliability and the MapReduce performance. Optimizing data placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tunings and experiences.

2.2 Need for a power controller in Hadoop

With the increase in the sheer volume of big data that needs to be stored and processed, data center footprint is becoming extremely large-scale. Data centers are known to be expensive to operate and they consume huge amounts of electric power. And over the lifetime of IT equipment, the operating energy cost is comparable to the initial equipment acquisition cost and constitutes a significant part of the total cost of ownership (TCO) of a data center [15]. Hence, energy conservation of the extremely large-scale, commodity server farms, such as Google cluster, has become a priority. Meanwhile, Google's study on server utilization and energy consumption reports that the energy efficiency peaks at full utilization and significantly drops as the utilization level decreases [16]. And the power consumption at zero utilization is still considerably high (around 50 %), as illustrated in Fig. 1. Apparently, even an idle server consumes about half its maximum power. It's also observed that the energy efficiency of these servers lies in the range of 20–60 % when operating under 20–50 % utilization.

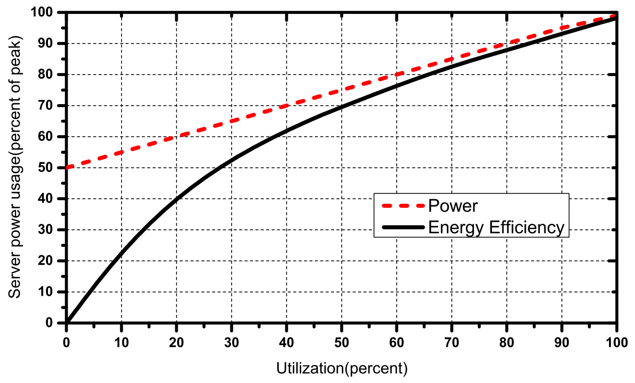

Fig. 1 Server power usage and energy efficiency at varying utilization levels

Hadoop as a Google's in house MapReduce implementation also has similar problem, and it may consume large quantities of power even while being low-utilization status. In light of our observation, this is mainly due to the current data placement strategy of HDFS which will make all DataNodes being active all the time to ensure data availability, i.e., the Hadoop cluster is short of the energy-proportional ability [11,17]. This indicates us that dynamically reconfiguring the number of active DataNodes according to the current applications workload is a good way to improve the energy efficiency of a heterogeneous Hadoop cluster.

2.3 Motivation

The existing data placement strategy and replication scheme of HDFS are working well with MapReduce framework in a homogeneous Hadoop cluster, but in reality, such homogeneity assumption can noticeably reduce MapReduce performance and may cause increasingly energy dissipation in heterogeneous environments. Besides that, HDFS employs an inflexible replica factor acquiescently for each data block, which will give rise to unnecessary waste of storage space when there is a lot of inactive data in Hadoop system.

All of the above analytical results motivate us to develop a new data placement strategy, which is energy-efficient, space-saving and able to improve MapReduce performance in heterogeneous Hadoop clusters.

3 System model and implementation

In this section, we discuss the detailed design of Hadoop system model that using SLDP and its relevant algorithms.

3.1 Heterogeneous Hadoop system model

Figure 2 illustrates the overall architecture of a heterogeneous Hadoop cluster system for which our proposed algorithms are

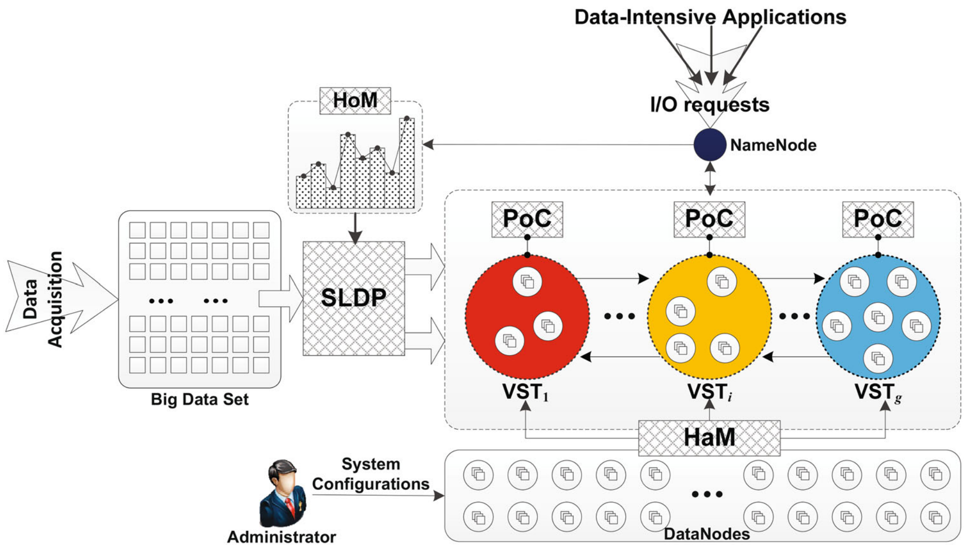

Fig. 2 Overview of a Hadoop system using SLDP

adopted. In this system, in order to improve the MapReduce performance and reduce the waste of power and storage-space consumptions, the following four main components are projected:

Component 1 (Heterogeneity-aware Module, HaM). As a heterogeneity perceptron for multifarious DataNodes in the cluster, this component uses a heterogeneity-aware algorithm (named Haag, the details can be seen in Sect. 3.2) to divide these DataNodes into several VSTs according to their performance properties. And the nodes with similar performance configuration will be categorized into a same VST.

Component 2 (Hotness Monitor, HoM). HDFS is able to log all file system access requests. The logging is implemented using `log4j` and once enabled, logs every HDFS event in the NameNode's log. This module monitors the HDFS meta-data checkpoint and logs for analysis, and then figures out the real-time hotness of all data blocks, based on that, it will use hotness-proportional replication policy (HP) to determine the replica factor of each data file.

Component 3 (Snakelike Data Placement Module, SLDP). By integrating the information collected from HoM and HaM, this module can spread data blocks across DataNodes in each VST circuitously like a slowly swimming snake, and place hot data into the high performance VSTs and cold data into the low performance VSTs, such that, most of the MapReduce tasks will be processed on the faster DataNodes and rest of them can be done on the slower DataNodes.

Component 4 (Power Controller, PoC). According to the scheme of data blocks placement determined by SLDP, this module can scale the number of active DataNodes in each VST dynamically, and manage the energy consumption of the entire Hadoop cluster by using an intelligent power control strategy (PCS). It is a channel between the cluster's electric meter and HDFS.

The four main components mentioned above cooperate with each other to improve MapReduce performance, reduce system power consumption and enhance the efficiency of storage space in a heterogeneous Hadoop cluster. The detailed design of primary algorithms for them are described in the following subsections.

3.2 Haag: heterogeneity-aware algorithm

The heterogeneity of DataNodes in Hadoop clusters is becoming a more and more sensitive factor to MapReduce framework performance. Xie et al. [18] developed a data placement mechanism in HDFS that distributed and stored a large data set across multiple heterogeneous nodes in accordance to the computing capacity of each node. However, the I/O performance parameter of a DataNode is another key factor that should not be neglected especially for most of data-intensive applications. Consequently, as the first release of the year 2014, Hadoop-2.3.0 brings a significant enhancement to HDFS, that is, support for heterogeneous storage hierarchy in HDFS (`HDFS-2832`) [19]. Unfortunately, the differences between storage types are described qualitatively by this prototype without a quantitative calculation.

In this subsection, we attempt to introduce a heterogeneity aware algorithm (Haag) to distinguish the comprehensive performance of all DataNodes by a rational mathematic method. The core idea of Haag is, based on fuzzy classifier method [20], taking into account the multiple performance parameters (such as computing capacity, memory size and IOPS, etc.) of the DataNodes, and then quantificationally dividing them into several VSTs. Thereby, Haag can allow

HDFS to make better decisions about the distribution of data blocks with input from applications to an appropriate VST.

Without loss of generality, we denote the DataNodes set as $DN = \{dn_1, dn_2, \cdots, dn_n\}$, and the performance indicators set of each DataNode is $Y = \{y_1, y_2, \cdots, y_m\}$. The j-th indicator value of DataNode dn_i is denoted by $x_{ij} = dn_i \otimes y_j$, and we can get a relation matrix about the n DataNodes' performance, that is, $X = DN \otimes Y = (x_{ij})_{n\times m}$. Based on such relation matrix X, the Haag can deal with it to distinguish the comprehensive performance of all DataNodes. The specific steps are depicted as follows:

Step 1. *Get the standardization of all DataNodes' performance indicators*. From the viewpoint of mathematics, an exact classification is determined by a generalized relation of equivalence, and the fuzzy classification system is identified with a fuzzy equivalent relation matrix. In order to construct a fuzzy relation matrix, we need to pre-process all quantitative values of DataNodes' performance indicators, and normalize them into [0,1] interval. Specifically, we first calculate the average value and standard deviation of each performance indicator for all DataNodes by the following formulas (1) and (2) respectively.

$$\overline{x_j} = \frac{1}{n}\sum_{i=1}^{n} x_{ij}, \ (x_{ij} \in X) \tag{1}$$

$$S_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x_j})^2} \tag{2}$$

And then, we introduce Eq. (3) to standardize the value of raw data x_{ij}, and we have:

$$x^*_{ij} = \frac{x_{ij} - \overline{x_j}}{S_j} \tag{3}$$

After that, based on the extreme values, we use Eq. (4) to normalize x^*_{ij} into [0,1] interval:

$$x_{ij} = \frac{x^*_{ij} - x^*_{jmin}}{x^*_{jmax} - x^*_{jmin}} \tag{4}$$

where, $x^*_{j\,\min} = \min\{x^*_{ij} | i \in [1, n]\}$, and $x^*_{j\,\max} = \max\{x^*_{ij} | i \in [1, n]\}$. Obviously, there is $x_{ij} = 0$ when $x^*_{ij} = x^*_{j\,\min}$, and $x_{ij} = 1$ when $x^*_{ij} = x^*_{j\,\max}$.

Step 2. *Get the calibration of fuzzy similar matrix using Cosine method*. We introduce λ-similarity factor r_{ij} to construct the fuzzy similar matrix R by using Cosine method, and we can get:

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{pmatrix} \tag{5}$$

where $r_{ij} = \frac{\sum_{k=1}^{m} x_{ik} \cdot x_{jk}}{\sqrt{\left(\sum_{k=1}^{m}(x_{ik})^2\right)\cdot\left(\sum_{k=1}^{m}(x_{jk})^2\right)}}$ is a variable that can indicates the level of similarity between DataNode dn_i and dn_j. If r_{ij} is closer to 1, it means the performance difference between DataNode dn_i and dn_j is smaller.

Step 3. *Get the transformation of R by using Transitive Closure method*. Here gives two important primary definitions firstly.

Definition 1 (**Transitive Closure**). Let R be a fuzzy relation matrix, the lowest transitive fuzzy matrix $t(R)$ that contains R is called the transitive closure of R if:

1. $t(R) \otimes t(R) \subseteq t(R)$;
2. $t(R) \supseteq R$;
3. If $S \supseteq R$ and $S^2 \subseteq S$, then there is $S \supseteq t(R)$.

Definition 2 (**Fuzzy Equivalence Relation**). Let $R = (r_{ij})_{n\times n}$ be a fuzzy equivalence relation matrix, if it satisfies the following axioms:

1. *Reflexivity*, i.e., $r_{ii} = 1$ $(i = 1, 2, \cdots, n)$;
2. *Symmetry*, that is, $r_{ij} = r_{ji}$ $(i, j = 1, 2, \cdots, n)$;
3. *Transitivity*, i.e., $R \otimes R \subseteq R$.

In general, the calibration of fuzzy similar matrix R obtained from **Step 2** by using Cosine method is not a fuzzy equivalence relation matrix for lack of transitivity. Fortunately, [21] has proved that given a fuzzy similar matrix R on finite universe, there exists a transitive closure of R, that is $t(R) = R^k$, which is a fuzzy equivalence relation matrix, where $k \leqslant n$.

As a consequence, we use a matrix exponential function $f(R) = R^2$ to figure out the transitive closure of R after finite calculations, that is, $R \mapsto R^2 \mapsto R^4 \mapsto \cdots \mapsto R^{2k}$, then we have $R^k = (R^k)^2$, and $t(R) = R^k$ is the fuzzy equivalence relation matrix that we want.

Step 4. *Analyse the classification of all DataNodes*. Once the fuzzy equivalence relation matrix R^k is acquired from **Step 3**, we can use an intercept factor λ to reconstruct a new classification matrix R^k_λ. Concretely, by traversing all elements of the matrix R^k, we can set a element of R^k_λ to be 1 if the corresponding element is greater than or equal to λ, otherwise assign it to be 0. And, finally, we obtain the reconstruct matrix R^k_λ.

Step 5. *Get the virtual storage tiers*. We divide all DataNodes that have the same row value of R^k_λ into a same VST region, such that we can distinguish the performance of all DataNodes clearly.

With the above five steps, we can divide various Hadoop DataNodes into several VSTs according to the performance properties of each DataNode. And then, the hot data will be spread into high performance VSTs, and vice versa.

3.3 HP: hotness-proportional replication policy

A study of Yahoo!'s Hadoop clusters illustrated in [22] shows significant variation in the access patterns of the data files stored in the cluster. We could find existence of a significant amount of data in the system which either wasn't accessed at all or rarely accessed after some amount of elapsed time. As shown in [22], the `FileLifeSpanCFR` (the file lifespan between the file creation and first file read access) of 90.26 % of data is less than 2 days. Thus, data is accessed soon after its creation. 89.61 % of data has a `FileLifeSpanCLR` (the file lifespan between the file creation and the last file read access) of less than 10 days. This indicates that majority of the data is hot for less than 10 days after its creation in the system. 40 % of the data in the cluster has a `FileLifeSpanLRD` (the file lifespan between last file read access and file deletion) of higher than 20 days. This prompts that 40 % of the data lies untouched in a dormant state in the cluster for more than 20 days. In such a situation, for the default fixed replica factor of three, HDFS's replication policy will result in an unnecessary waste of storage space when there is a lot of cold data in Hadoop system.

But fortunately, this study indicates that there are tremendous opportunities to distinguish the data files into different replication classes in a Hadoop cluster. Thus, we propose a hotness-proportional replication policy (HP) for HDFS to improve the efficiency of storage space. More specifically, HP will determine the replica factor of each data block according to its hotness (which reflects the popularity of data, and is usually calculated by the access frequency of the data in a specific time slot. If the hotness of a data block is high, we can say it is a hot or active block, else if the hotness is low, this data block is cold or inactive), that means if the hotness of a data block is higher its replica factor will be larger. Therefore, how to determine the hotness of data is a critical process for the realization of HP policy.

In a practical application, the data hotness is not only determined by the access frequency, but also closely interrelated to the time series of access requests. Taking the physics analysis of AMS-02 experiment [23,24] as an example, scientists around the world from AMS group will frequently access scientific data named *root files* that reconstructed on our cluster with specific raw data collected from International Space Station. We randomly selected three different root files and counted the number of access times per hour of them in a same time slot T_t (one week), and then we obtained the results as shown in Fig. 3. As demonstrated in this figure, although the total number of times that each root file was accessed are almost identical to each other, the root file of Scenario 1 was frequently accessed earlier in the week, and the root file

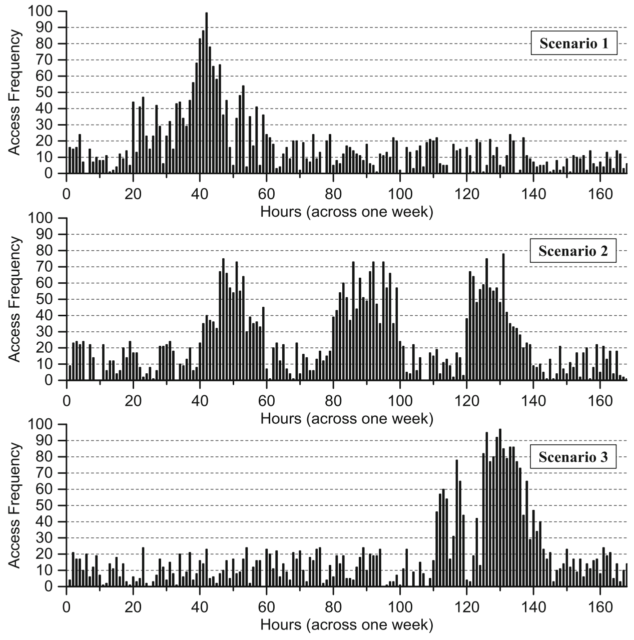

Fig. 3 Different scenarios about data access frequency in a same time slot T (1 week)

of Scenario 2 was accessed in three main waves throughout the whole week, meanwhile, the root file of Scenario 3 was accessed mostly at the end of the week. From the viewpoint of the principle of program locality, there is a higher possibility for the most recently used files will be accessed again in a near future. Thus, intuitively, the data hotness of Scenario 3 is higher than the others, and data file in Scenario 1 is the coldest one.

Based on this, we had analysed a large amounts of history monitoring data collected from HoM module as illustrated in Fig. 2, and then put forward the following experiential formulas (6) and (7) to calculate the hotness of each data blocks. The NameNode maintains a time-stamp queue and stores the corresponding hotness for every data block into a dedicated table. Once a new time-stamp T_{t+1} is added into the queue of block b_i, the NameNode will update its hotness according to the Eq. (7).

$$h_0(b_i) = 0 \tag{6}$$

$$h_{t+1}(b_i) = h_t(b_i) \times e^{-\lambda(T_{t+1}-T_t)} + k \times R_t \tag{7}$$

where, $h_{t+1}(b_i)$ denotes the updated hotness of data block b_i when the $(t+1)^{th}$ access request for this block is arriving. R_t is the number of access requests in T_t time slot. $k \in (0, 1)$ is a constant coefficient, and the positive number λ is the hotness extraction coefficient.

Formula (7) can synthetically take into account the effects of time interval $\triangle T = T_{t+1} - T_t$, the number of access requests R_t and the pre-update hotness $h_t(b_i)$ for the value of upcoming hotness, that means, it can reflect the long term trends in data files' access patterns.

Then, the HoM module will refresh and record the replica factor for all data blocks stored in HDFS by using the following Eq. (8) periodically (about once a week in our system, and can be dynamically adjusted by administrator according to need of the practical application).

$$rf(b_i) = h_t\left(b_i|\overset{\wedge}{\theta}_{mle}\right) \tag{8}$$

where $rf(b_i)$ denotes the replica factor of data block b_i, $h_t(b_i)$ is the hotness of it that can be queried from NameNode, and

$$\overset{\wedge}{\theta}_{mle} = \arg\max_{\theta\in\Theta} \sum_{i=1}^{m} \ln h_t(b_i|\theta)$$

is the maximum likelihood estimator (*mle*) of the vector of parameters $\theta \in \Theta$ observed from HDFS' logs by using the Maximum Likelihood Estimation [25].

3.4 SLDP: snakelike data placement strategy

From the above two subsections, we could acquire the information of VSTs (that is, for every DataNode, which VST should it belong to), and the hotness, replica factor of each data block also. Based on that, we design a novel SLDP for HDFS, which can spread the hot data into the high-performance VSTs and the cold data into the low-performance VSTs. Specific procedures of SLDP are described as follows:

Step 1. SLDP divides all the DataNodes into g VSTs by using Haag algorithm, that is, we can get the virtual storage tiers set as VST. The number of DataNodes classified in $\text{VST}_k \in VST$ is $T(k)$, where the index $k \in [1, g]$ is a natural number (line 1 in Algorithm 1).

Step 1.1. For each $\text{VST}_k \in VST$, SLDP sorts all the $T(k)$ DataNodes in each VST_k according to the IOPS performance parameter of them (line 3 in Algorithm 1), for main applications served by SLDP are data-intensive computing and the I/O resources are the most important resource. As a result, we can get VST_k^*, which is the collection of the DataNodes in VST_k ordered by IOPS. Where, $dn_{k,l}$ is the DataNode with l-th highest IOPS and in the k-th highest comprehensive performances tier of the whole Hadoop cluster.

Step 1.2. The total storage space of each VST_k^*, denoted by S_k, is accumulated by all disk space size of the DataNodes in this tier (line 4 in Algorithm 1), so that, SLDP can determined the data blocks collection that will be stored in it according to S_k.

Algorithm 1: DataNodes classification and ranking

```
Input: $DN$(the set with $n$ DataNodes), $\lambda$(the intercept factor)
Output: $VST^*$, $S$
1 $VST = \{\text{VST}_k|k \in [1, g], |\text{VST}_k| = T(k)\} \leftarrow \texttt{Haag}(DN, \lambda)$
2 for $k = 1 \rightarrow g$ do
3     $\text{VST}_k^* = \{dn_{k,l}|l \in [1, T(k)]\}$ ←sort all the $T(k)$ DataNodes
      in $\text{VST}_k$ by the IOPS performance parameter
4     $S_k$ ← accumulate the total space of DataNodes in $\text{VST}_k^*$
      (MB)
5 end
6 $VST^* = \{\text{VST}_k^*|k \in [1, g]\}$
7 $S = \{S_k|k \in [1, g]\}$
8 return $VST^*$, $S$
```

Step 2. SLDP sorts the hotness of all data blocks and calculates the replica factor of each data block according to Eq. (8) through Algorithm 2.

Algorithm 2: Rank hotness and calculate replica factor

```
Input: $B$(the collection with $m$ data blocks)
Output: $H$, $B^*$, $RF$
1 $H = \{h(b_i)|i \in [1, m]\}$ ←get hotness from the NameNode
  through the HoM module
2 $B^* = \{b_i^*|i \in [1, m]\}$ ← sort all $m$ data blocks in $B$ according to
  their hotness by the HoM module
3 $RF = \{rf(b_i^*)|i \in [1, m]\}$ ← figure out the replica factor for
  each block by Eq. (8) based on its hotness
4 return $H$, $B^*$, $RF$
```

Step 3. For each data block, its replica factor is dynamically changed according to the hotness of it in different time slot. We use Algorithm 3 to calculate the number of primitive data blocks stored in each $\text{VST}_k \in VST$, $k \in [1, g]$. Where α is the default block size (MB) and β is used to reserve some disk space.

Algorithm 3: Calculate the number of primitive data blocks stored in VST

```
Input: $S$(the space collection of $VST$), $RF$
Output: $D$(the number collection of primitive data blocks stored
        in $VST$)
1 $\alpha \leftarrow 128, \beta \leftarrow 0.9, i \leftarrow 1$
2 for $k = 1 \rightarrow g$ do
3     $D(k) \leftarrow i$, sumSize ← 0
4     while $S_k \times \beta > sumSize$ do
5         sumSize ←sumSize + $\alpha \times (rf(b_i^*) - 1)$
6         $i \leftarrow i + 1$
7     end
8     $D(k) \leftarrow i - D(k)$
9 end
10 $D = \{D(k)|k \in [1, g]\}$
11 return $D$
```

Step 4. SLDP distributes the data blocks across DataNodes in each $\text{VST}_k \in VST$, $k \in [1, g]$ circuitously like a slowly-swimming snake, as shown in Fig. 4.

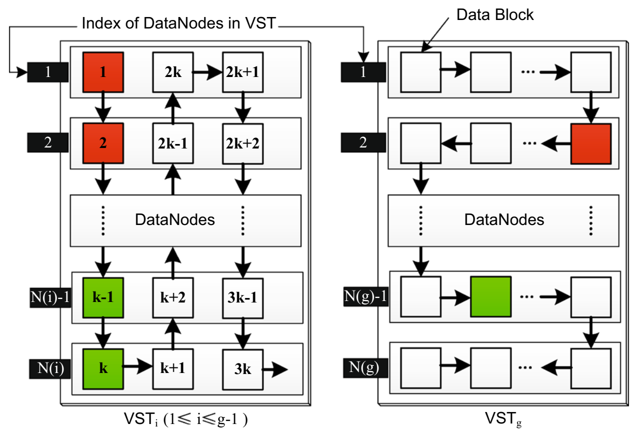

Fig. 4 Snakelike data placement diagram

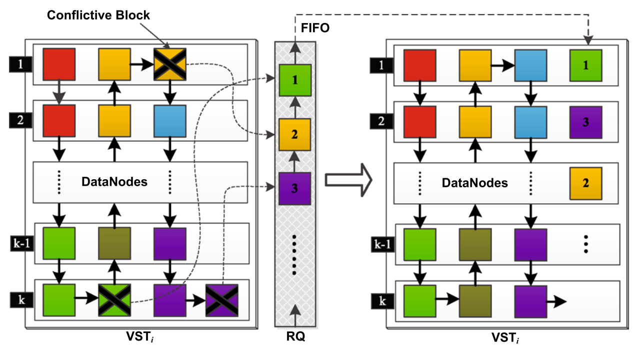

Fig. 5 Conflict situation and its solution during snakelike data placement

Algorithm 4: SLDP Strategy

```
   Input: $B^*$, $VST^*$, $D$, $RF$
   Output: $DP[n][m]$
1  $D(0) \leftarrow 0$, $D^*(0) \leftarrow 0$, $DP[n][m] \leftarrow 0$
2  for $k = 1 \rightarrow g-1$ do
3      $RQ \leftarrow \phi, l \leftarrow 1, d \leftarrow 0$
4      $D^*(k) \leftarrow \sum_{i=1}^{D(k)} (rf(b^*_{(i+D(k-1))})-1)$
5      for $j = (k-1)D^*(k-1)+1 \rightarrow D^*(k)$ do
6          doPlace $\leftarrow$ true
7          while doPlace is true do
8              if left space of $dn_{k,l}$ is enough then
9                  if $b^*_j$ is not in $dn_{k,l}$ then
10                     $DP[(k-1)T(k-1)+l][j] \leftarrow 1$
11                     if $d == 0$ then
12                         if $l == T(k)$ then $d \leftarrow 1$
13                         else $l \leftarrow l+1$
14                     else
15                         if $l == 1$ then $d \leftarrow 0$
16                         else $l \leftarrow l-1$
17                     end
18                 else
19                     $RQ \leftarrow b^*_j$
20                 end
21                 doPlace $\leftarrow$ false
22             else
23                 if $d == 0$ then
24                     if $l == T(k)$ then $d \leftarrow 1, l \leftarrow l-1$
25                     else $l \leftarrow l+1$
26                 else
27                     if $l == 1$ then $d \leftarrow 0, l \leftarrow 2$
28                     else $l \leftarrow l-1$
29                 end
30             end
31         end
32     end
33     for each $b^*_j \in RQ$ do
34         for $l = 1 \rightarrow T(k)$ do
35             if $b^*_j \notin dn_{k,l}$ and space is enough then
36                 $DP[(k-1)T(k-1)+l][j] \leftarrow 1$
37                 break
38             end
39         end
40     end
41 end
42 $D^{**}(0) \leftarrow 0$
43 for $l = 1 \rightarrow T(g)$ do
44     $D^{**}(l) \leftarrow \lfloor \text{leftSpaceOf}(dn_{g,l}) \times \beta/\alpha \rfloor$
45     for $j = (l-1)D^{**}(l-1)+1 \rightarrow D^{**}(l)$ do
46         $DP[(g-1)T(g-1)+l][j] \leftarrow 1$
47     end
48 end
49 return $DP[n][m]$
```

Step 4.1. The hottest block is placed in the first DataNode with highest IOPS performance in VST$^*_k$, $k \in [1, g-1]$, the second hottest block is placed in the second DataNode with the second highest IOPS performance, and the rest can be done in the same manner (line 5–32 in Algorithm 4, Fig. 4, left), except the last replica of each data block (done in Step. 4.3, Fig. 4, right).

SLDP can spread data blocks across the nodes in a heterogeneous Hadoop cluster to take full advantage of the high-performance DataNodes, and can also improve the hot data locality of MapReduce computation in the meantime. While benefiting in terms of performance, this design principle simplifies power-management to endow with energy proportional ability for Hadoop clusters. However, during the process of data placement in Hadoop clusters, SLDP may suffer from a "conflict" situation that occurred in the "corner" of the snake shape as illustrated in Fig. 5, left, for violating the restriction that a DataNode could not store a two same data blocks to ensure data durability (line 18 in Algorithm 4).

Step 4.2. In order to deal with such conflict situation, a "rescue queue" named RQ is purposely designed to store the "homeless" data blocks temporarily in each VST$^*_k$ (line 3 in Algorithm 4). And RQ will be handled after the remainder of blocks were placed successfully already. These homeless blocks residing in RQ are distributed into the DataNodes belong to VST$^*_k$ according to FIFO approach. As demonstrated in Fig. 5, right, the head block in RQ will be placed in the first DataNode, which is on the top of IOPS performance in VST$^*_k$ and has enough storage space as well as without storing the same block. Otherwise, the block will be placed to the next one DataNode that satisfies these preconditions. The rest can be done in the same manner recursively (line 33–40 in Algorithm 4).

Step 4.3. The last replica of each data block will be placed in the lowest-performance VST$^*_g$ separately like a transverse slowly-swimming snake (Fig. 4, right, line 42–48 in Algorithm 4). This might be necessary because, technically, it is possible to keep a behind-the-scenes replica for each data block to ensure fault-tolerant capacity. More importantly, the DataNodes in VST_g^* can be treated as inactive nodes that can be configured to power-off mode to save energy consumption. And once one of these DataNodes is waken to recover some data blocks, all the data blocks stored in it with similar hotness are available, that will improve the entire Hadoop performance apparently.

Through the above steps, every data block can be successfully placed into an appropriate DataNode in a heterogeneous Hadoop cluster according to its hotness.

3.5 PCS: power control scheme

SLDP spreads data blocks and their replicas across the DataNodes circuitously according to the hotness of them like a slowly swimming snake for higher MapReduce performance, load-balancing and resiliency, as described in Sect. 3.4. With all the data blocks being distributed into Hadoop clusters already, those with high performance DataNodes may be frequently participating in the reading, writing, or computation of a batch of data blocks in most of time. Such data placement mechanism not only improves the MapReduce framework performance, but also endows with energy-proportionality for the Hadoop clusters. That means SLDP makes it possible to generate significant periods of idleness for some DataNodes in the Hadoop clusters, and it can render usage of inactive power modes feasible.

When obtaining the data blocks placement scheme matrix by SLDP, we propose an optimal power control scheme (PCS), and implement a channel named PoC between the cluster's power controller and HDFS (see Fig. 2). According to our proposed PCS, the component PoC can scale the number of active DataNodes in each VST dynamically to meet the service demands of the data-intensive applications, and can minimize the energy consumption for the whole Hadoop runtime system.

3.5.1 Mathematical model of PCS

As mentioned above, we denote the n DataNodes set in a heterogeneous Hadoop cluster as DN, and the parameter set of power consumption of all DataNodes is denoted by PC. The m data blocks set is B. Then, based on the data placement result determined by SLDP, minimizing the energy consumption for the Hadoop clusters can be illustrated as obtaining a status vector SV about all DataNodes to keep the availability of all data blocks (i.e., there is an active replica for each block at least). Hence, we can present the mathematical model of PCS as follows:

$$\text{Minimize}\, PoC(PC, SV) = PC \times SV^T$$
$$\text{s.t.} \begin{cases} DP = DN \otimes B \leftarrow SLDP(DN, B) \\ \forall ba_i \geqslant 1 (ba_i \in SV \otimes DP, 1 \leqslant i \leqslant m) \\ \forall pc_i > 0 (pc_i \in PC, 1 \leqslant i \leqslant n) \\ \forall sv_i \in SV = \begin{cases} 1, \text{ active} \\ 0, \text{ inactive} \end{cases}, 1 \leqslant i \leqslant n \end{cases} \tag{9}$$

where the first constraint condition indicates this power control strategy is based on the data placement result of SLDP; the second constraint condition can guarantee the availability of all data blocks; the third one ensures the power cost of arbitrary DataNode is positive; and the last one restricts the status of every DataNode can only to be active (the value is 1) or inactive (the value is 0).

3.5.2 Solution for PCS based on bipartite matching

Formally, the above problem can be explained by a relaxation of the maximum bipartite matching problem called *semi-matching* problem in the weighted case as follows. Let $G = (U \bigcup V, E)$ be a weighted bipartite graph, where U is a set of DataNodes and V is a set of data blocks. For any edge uv, let pc_{uv} be its weight. Each weight of an edge uv indicates power consumption it takes u to process data block v, as shown in Fig. 6.

Let n denote the number of DataNodes and m denote the number of primary data blocks in G. A set $M \subseteq E$ is a semi-matching if each block $v \in V$ is incident with exactly one edge in M. For any semi-matching M, we define the *cost* of M, denoted by cost(M), as follows. So, for any DataNode $u \in U$ in a semi-matching M, its cost with respect to M is

$$\text{cost}_M(u) = \sum_{(u,v) \in M} pc_{uv} \tag{10}$$

Intuitively, this is the total power consumption of processing data blocks assigned to u. Now, the cost of the semi-matching M is simply the summation of the cost over all $u \in U$ in M:

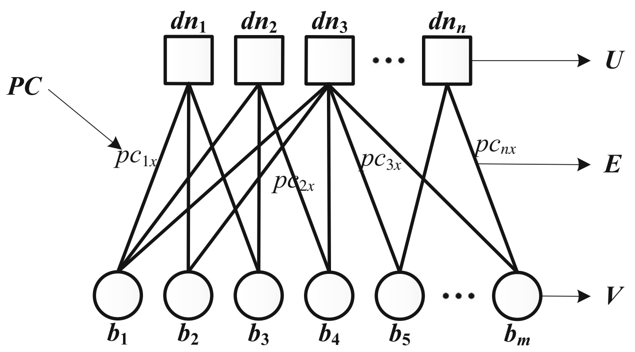

Fig. 6 A weighted bipartite graph for PCS

$$\mathrm{cost}(M) = \sum_{u \in U} \mathrm{cost}_M(u) \tag{11}$$

Then, the goal is to find an *optimal semi-matching* with minimum cost, and the set of DataNodes in it are the target nodes should be active and the rest of DataNodes in the Hadoop cluster should be deployed into sleep mode to reduce energy consumption. Since we all know, the Kuhn–Munkres algorithm (KM) is a classical and efficient solution for bipartite matching problem [26], based on that we propose an improved algorithm named *minPowerMatch* to determine the optimal semi-matching M for PCS as shown in Algorithm 5.

Algorithm 5: minPowerMatch

```
   Input: $G = (U \bigcup V, E)$ (a weighted bipartite graph, $|U| \leqslant |V|$)
1  if $|U| == |V|$ then
2      $M \leftarrow \mathrm{KM}(G)$
3  else
4      $k \leftarrow |V| - |U|, X \leftarrow \{x_1, x_2, \cdots, x_k\}$
5      $\hat{U} \leftarrow U \cup X$
6      $\hat{E} \leftarrow E \cup \{\langle x_i, v_i \rangle \,|\, i \in [1, |X|], j \in [1, |V|], pc_{x_i, v_j} = 0\}$
7      $\hat{G}(\hat{U} \cup V, \hat{E}) \leftarrow$ Transform $G$ according to the results of
       Step 5 and 6
8      $\hat{M} \leftarrow \mathrm{KM}(\hat{G})$
9      $M \leftarrow$ Eliminate $\hat{E}^{\dagger} = \{\langle x_i, v_j \rangle \,|\, x_i \in X\}$ from $\hat{M}$
10 end
11 Set all DataNodes ($\in M$) into active mode and the rest of
   DataNodes ($\notin M$) in the Hadoop cluster into sleep mode.
```

In this way, we can optimize power cost of the whole Hadoop cluster by managing the status of all DataNodes dynamically through *minPowerMatch* algorithm.

4 Evaluation results

In this section, we evaluate the performance of SLDP and the relevant algorithms from three perspectives with two real data-intensive applications. Before discussing the evaluation results, we will first describe the experimental setup.

4.1 Experimental setup

The experiments were conducted on a 252-node Hadoop cluster located in a data center of Southeast University (or SEU), whose overall architecture is illustrated in Fig. 7 including network and hardware configurations. This data center has been expanded three times for equipment renewal and modernization since 2011, and there are multiple generations of hardware in the cluster. The detailed configurations are shown in Table 1. Even two DataNodes have the same CPU model, they may own different disks or RAMs, that is, it is a typical heterogeneous Hadoop cluster.

We realized our work on the Apache Hadoop-2.3.0 release including the heterogeneity aware algorithm, hotness proportional replication, snakelike data blocks placement strategy and the power control function. We evaluated them with the $\mathrm{e}^-/\mathrm{e}^+$-Identify (similar to Word-Count) jobs and AMS-02 Physics Analysis jobs (to study the universe and its origin by searching for antimatter, dark matter while performing precision measurements of cosmic rays composition and flux, based on the raw data collected from the International Space Station or ISS and Monte Carlo simulation results) [23] [24] on the Hadoop cluster at SEU.

The experimental data set consisted of 3TB raw data acquired in February 2014 from ISS with its reconstructed data (about 15TB) and 8TB Monte Carlo simulations data produced at SEU over the same period. In this 26TB data set, there are approximately 1.3 billion events that recorded the information of the high-energy particles passed through AMS-02's sub-detectors (including category, energy, velocity, flight direction, etc.).

4.2 Hadoop cluster performance evaluation

The data locality determined by HDFS' data placement strategy is an important factor for MapReduce performance. Especially in a heterogeneous environment, the current default data placement policy of HDFS can cause severe performance degradation. In order to verify the effectiveness and efficiency of our proposed SLDP and relevant algorithms, we had run two different MapReduce jobs that have different data access patterns. Similar to Word-Count application, the first one job is called $\mathrm{e}^-/\mathrm{e}^+$-Identify procedure whose purpose is to count the frequency and energy of electron and positron recorded in all data blocks respectively. The second one is designed to analyze the actual spectrum structure and fluctuation of $\mathrm{e}^-/\mathrm{e}^+$ whose energy are greater than 100 GeV, named AMS-02 Physics Analysis job. All results obtained from these two jobs are significant to deduce the origin of those superfluous positrons in the universe.

During the evaluation, we compared three data placement mechanisms of HDFS, that is, (1) *default-Strategy*: the default data placement strategy of current HDFS release; (2) *non-EC-SLDP*: the snakelike data placement strategy presented by this paper without regard to energy conservation; (3) *EC-SLDP*: the snakelike data placement strategy proposed by this paper taking energy conservation into account. Based on this, we had run the two MapReduce jobs as mentioned above on the Hadoop cluster at SEU that deployed the three data placement mechanisms separately. The numbers of each job were set from 500 to 3000 with 500 increment each time, and the default data block size α was configured as 128 MB/256 MB respectively. Figure 8 reveals the specifics about final experimental results.

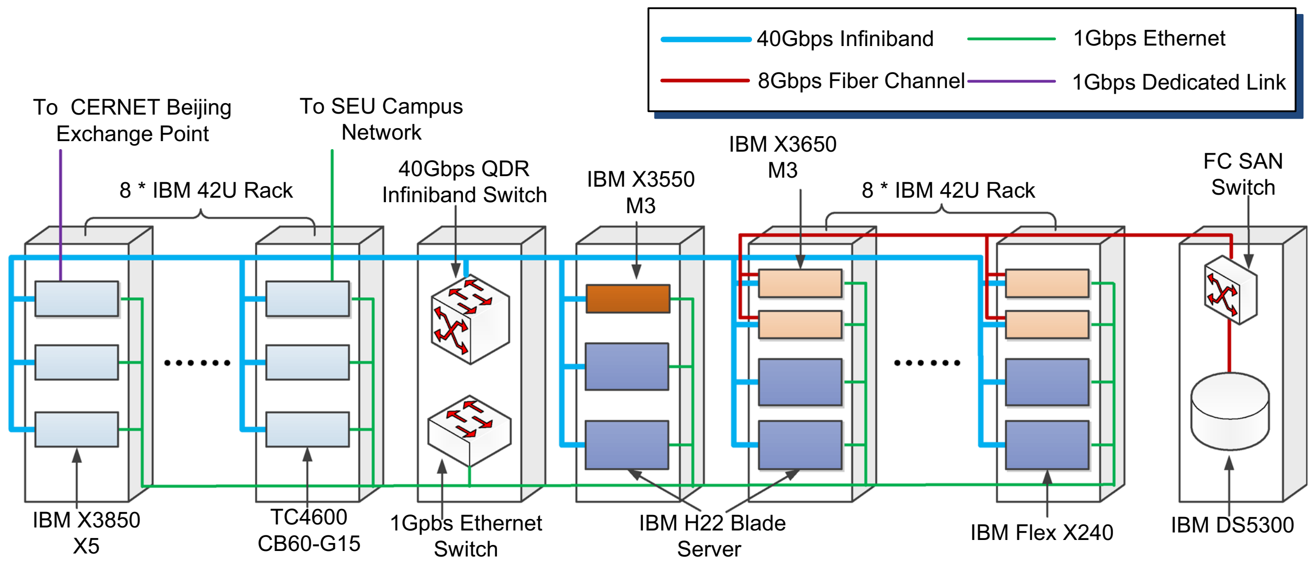

Fig. 7 Overall architecture of a Hadoop cluster at Southeast University

Table 1 Different configurations for the DataNodes in the Hadoop cluster at Southeast University

CPU			RAM	Disk		
Model	Speed	Cores		Type	IOPS	Size
X5650	2.66 GHz	6	24 GB	7.2K SATA	70	2 TB
X3850	2.66 GHz	6	64 GB	10K SAS	100	1 TB
X240	2.70 GHz	12	256 GB	SLC SSD	5000	120 GB
E5-2670	2.60 GHz	8	64 GB	15K SAS	150	300 GB

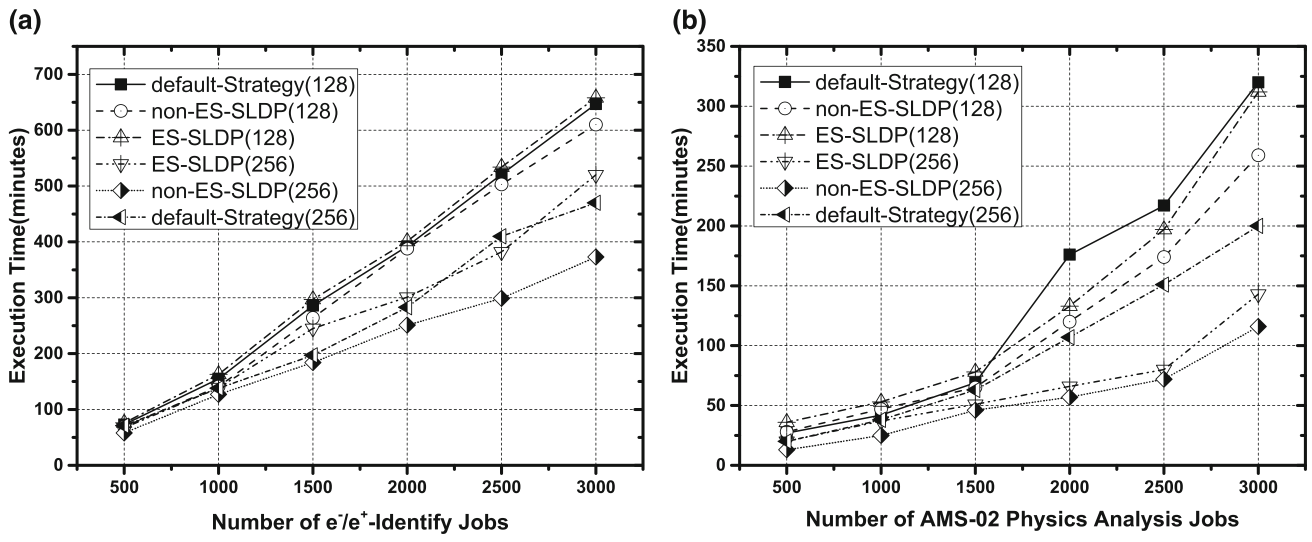

Fig. 8 Performance of two MapReduce jobs based on three different data placement strategies

From Fig. 8a, we can find out that the three data placement mechanisms have little effect on the e^{-}/e^{+}-Identify MapReduce job, except that the *ES-SLDP*'s performance is less impressive than the others. That's because this kind of job has to scan the whole raw data set every time during processing, and there is no evidence of hot data blocks such that it couldn't take full advantages of SLDP.

However, SLDP is more important for AMS-02 Physics Analysis jobs as illustrated in Fig. 8b. The reason is that, this kind of job is focusing on the blocks containing the elec-

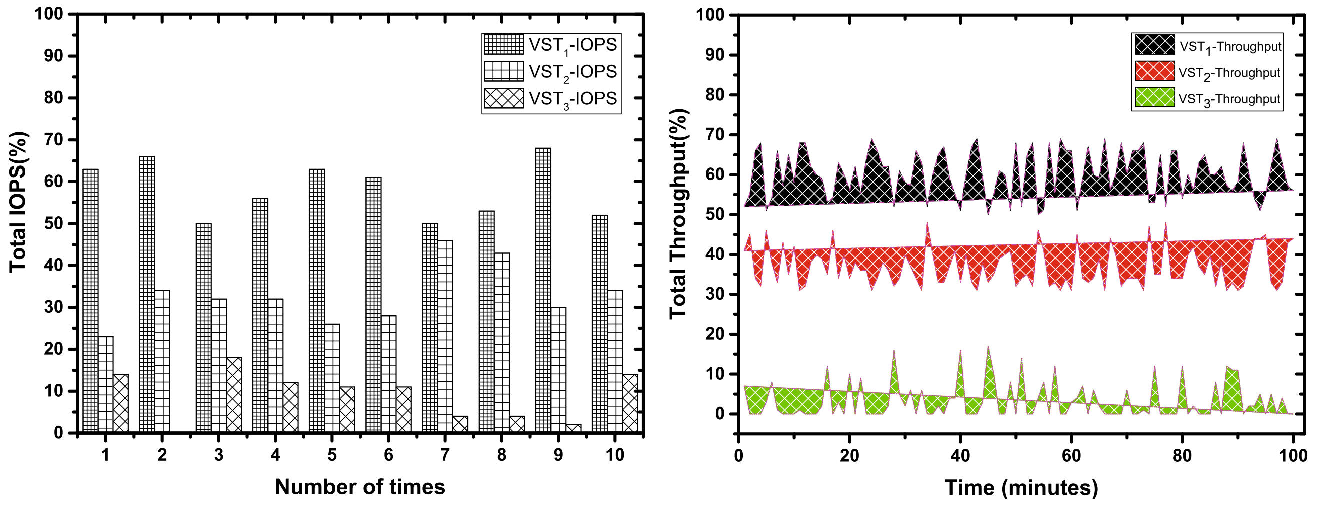

Fig. 9 IOPS and throughput performances of three different VSTs

trons and positrons whose energy are greater than 100 GeV, which means these jobs' data access patterns are skewed and may generate hot data spots. In addition, we noticed that the MapReduce performance is better when the default block size was set to 256 MB than to set as 128MB from Fig. 8a, b. This interesting phenomenon indicates that there is an opportunity to optimize the MapReduce performance, by figuring out the optimal value of the block default size with performing a large number of experiments in our future work.

Furthermore, through out the whole evaluation, all DataNodes in the Hadoop cluster at SEU are divided into three VSTs by using Haag algorithm. To be precise, there are ten DataNodes in VST_1 and 74 DataNodes in VST_2, and the rest 168 DataNodes are belong to VST_3. The ranking about comprehensive performances of them are $VST_1 > VST_2 > VST_3$. When we tested the MapRreduce performance of our Hadoop system, we also measured the accumulated IOPS and throughput performance of all DataNodes in each VST. We have built into our test scripts a 30-second pause between the individual tests. This pause is used to avoid distortion throughput writing tests measurements for flash-based storage (i.e., SSD). The final results are demonstrated in Fig. 9.

As shown in Fig. 9, we can conclude that, on average, VST_1 contributes more than 58 % of the total I/O performance (including both IOPS & throughput measurements) for the entire Hadoop cluster, VST_3 can only provide about 9 % of the whole performance, and the rest 33 % are offered up by VST_2. Obviously, it is because the storage devices deployed on DataNodes in VST_1 are all the highest performance SSDs and those attached to DataNodes in VST_3 are all the lowest performance 7200 RPM SATA drives, meanwhile, the DataNodes in VST_2 employ 10K/15K SAS disk drives. All of the above also offer another perspective on the effectiveness of our proposed Haag algorithm.

4.3 Hadoop cluster energy consumption evaluation

The current default data placement strategy of HDFS will make all DataNodes being active all the time to ensure data availability, so that the Hadoop clusters are short of the energy-proportional ability. Fortunately, based on the proposed SLDP, even a heterogeneous Hadoop cluster can become an energy-efficient pioneer.

In this subsection, we validated the power consumptions of the Hadoop cluster at SEU using snakelike data placement strategy (named "PCS-Strategy"), and compared it with other three different energy-saving strategies (one is adopting the default placement strategy named "Default-Strategy", and the others named "CS-Strategy" and "All-In-Strategy" are proposed by [27] and [28] respectively).

During the test, we had run a six groups of AMS-02 Physics Analysis jobs on the Hadoop cluster continuously, and used a functional electric meter to record power cost consumed by all nodes. The results are shown in Fig. 10.

From Fig. 10, we can observe that the power consumption was degraded most when the Hadoop cluster using our energy control policy which is based on SLDP mechanism, and the fluctuation is more gradual than the other strategies during the experiment. The power consumption ratio decreased (about 30 %) is so impressive by comparison to the default strategy used by HDFS. In addition, the load of Hadoop cluster was keeping in heavy pressure during the test, and we can find out that the energy saving of "All-In-Strategy" is nearly negligible and no better than the default strategy. Consequently, we hold the opinion that, this policy is more applicable to the batch jobs with longer intervals.

Besides that, we also calculated the power consumption of each VST during the evaluation. Although the I/O performance of VST_1 is highest, it consumes the least energy cost,

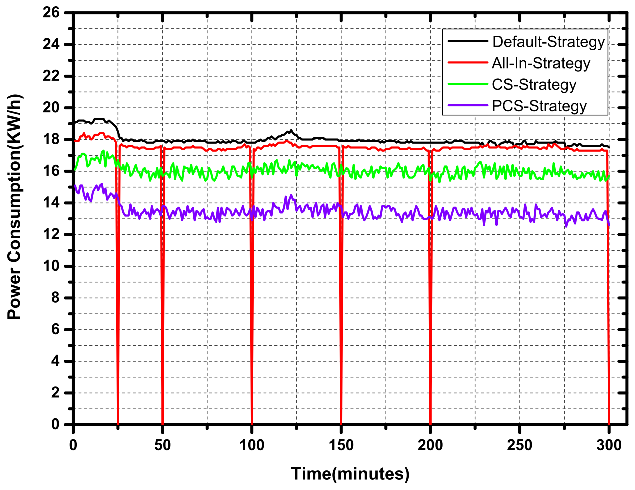

Fig. 10 Cluster power consumption with different energy saving strategies

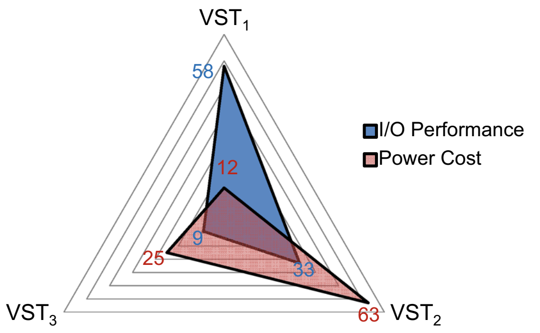

Fig. 11 I/O performance and power consumption statistical proportions of three different VSTs

as depicted in Fig. 11, only about 12 % of the whole Hadoop cluster power cost. It is because of that SSDs adopted by DataNodes in VST$_1$ use significantly less power at peak load than hard drives used in other VSTs, and their energy efficiency can deliver less power strain on system. This is also why we have to identify the difference between DataNodes by using our Haag algorithm, and we can take full advantage of the high-performance VSTs in our heterogeneous Hadoop cluster.

4.4 Hadoop cluster storage utilization evaluation

With the rapid growth of volume and velocity of big data, the storage costs are playing an increasingly large proportion of the data center cost. Improving the storage space efficiency is an important means of mitigating the TCO of data centers. For the default replica factor of three, HDFS's replica placement strategy may result in an unnecessary massive waste of storage space when there is a lot of inactive data. To take the data storage system at SEU for the AMS-02 experiment as example, there is about 25TB new experimental data needs to be stored into our data center every month. During the data processing and analysis, we need 75TB excess disk space per month if the Hadoop system using HDFS's default replication strategy. It is a heavy burden on our budge for equipment spending.

However, we had observed that about 20 % of data files archived in our system are active objects, and the other 80 % are "lazy boots" as illustrated in Fig. 12, left. Moreover, we also found out that, there are about 20 % of files of the active data are extremely popular that accessed by AMS-02 colleagues frequently. This motivates us to present the following replication policy in HDFS. For the 20 % $\times$ 20 % = 4 % called "red-hot" data blocks, we set their replica factors (or *rf* for short) as 4, the rest 16 % of active data blocks' *rf* as 3, and the other (80 %) cold data's *rf* as 2.

Based on such a simplified hotness-proportional replica factor strategy, the excess disk space we have to extend monthly in our data center at SEU just only needs 25TB $\times$ [(20 % $\times$ 20 %) $\times$ 4 + (20 % $\times$ 80 %) $\times$ 3 + 80 % $\times$ 2] = 56 TB. That means, we can save 25.33 % disk overhead every month, and can reduce about 250TB disk space consumption every year. Consequently, the storage capital expenditure savings can reach a quarter of million dollars per year, if the unit price of enterprise hard disk is 1$ per GB. More importantly, Fig. 12, right depicts a fact that, the efficiency of our storage system for AMS-02 experiment [23] is improved significantly (probably be around 30 %), since we adopted this replication policy at the end of January of year 2014.

5 Related work

Increasing evidence shows that heterogeneity problems must be tackled in MapReduce frameworks. Zaharia et al. implemented a new scheduling algorithm (LATE) in Hadoop to improve MapReduce performance by speculatively executing tasks that hurt response time the most [9]. LATE scheduling algorithm takes the heterogeneity assumptions into consideration, but has poor performance due to the static manner in computing the progress of the tasks. Asymmetric multi-core processors address the I/O bottleneck issue, using double buffering and asynchronous I/O to support MapReduce functions in clusters with asymmetric components [29]. Work by Fadika et al. [30] presented MARLA, a load-adaptive MapReduce framework espousing a task oriented approach to MapReduce application processing faced with the source of cluster heterogeneity. MARLA creates a multitude of tasks born from input splits, many times greater in number than the sum of its nodes. This approach allows the participating nodes to request tasks at their own pace. [31] proposed resource stealing which enables running tasks to steal the unutilized resources and return them when new

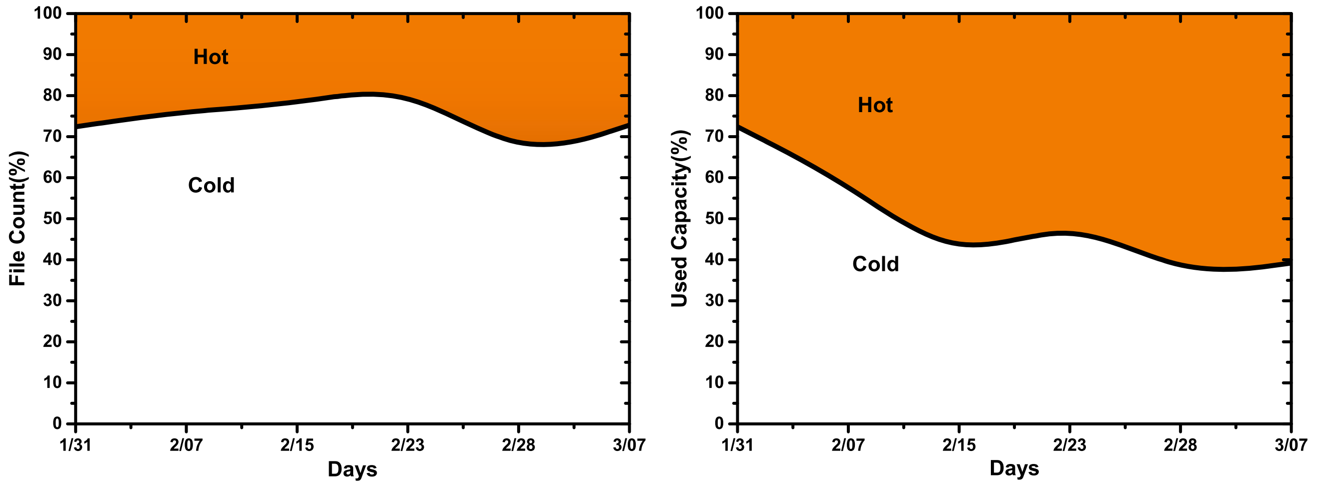

Fig. 12 Percentage of the hot and cold data and the associated file count in a Hadoop cluster at Southeast University

tasks were assigned. Besides that, they presented a Benefit Aware Speculative Execution, which predicts the benefit of running new speculative tasks and greatly eliminates unnecessary runs to alleviate the issue, that the current mechanism adopted to trigger speculative execution by heterogeneous Hadoop created many unnecessary speculative tasks that were killed soon after creation as the original tasks complete earlier. Although the above techniques can improve MapReduce performance of heterogeneous clusters, they do not take into account data locality and data movement overhead.

Data locality management is one of the features that have worse effect on the MapReduce performance [32], which is determined by HDFS' data placement strategy. Bad data locality management increases the execution time of MapReduce jobs. Some of approaches [18,33,34] have been recently introduced to improve the data locality management in heterogeneous MapReduce clusters. These approaches have some limitations. The first approach [18] of data locality management did not consider the data replication during data placement. If a node crashes, its input files are lost because there was no duplication of the input files. Additionally, it does not automatically support the fault tolerance feature. This approach can be enhanced by handling the redundancy issue of data allocation in the cluster, and designing a dynamic distribution mechanism for multiple data intensive applications. The second approach [33] of data locality management did not consider the workload heterogeneity. This approach can be enhanced by developing more effective technique for transmitting the data sets in the Hadoop environments. Also, this approach can be enhanced by a more accurate estimation of the transmission time. The third approach named ADAPT [34] dispatched data blocks based on the availability of each node and can reduced network traffic, improve data locality, and optimize the application performance, but will incur extra overheads to the existing Hadoop framework as it is an add-on feature of Hadoop.

All of the above approaches have an impact on energy efficiency as even idle machines remain powered on to ensure data availability. In order to address the issue of power conservation for clusters of nodes that run MapReduce jobs, Leverich et al. [27] presented a modified design for Hadoop that allows scale-down of operational clusters. They proposed the notion of "Covering Subset" during block replication that at least one replica of a data block must be stored in the covering subset. This ensures data availability, even when all the nodes not in the covering subset are turned off. Lang et al. [28] proposed a new energy management for MapReduce clusters called the All-In Strategy (AIS). AIS uses all the nodes in the cluster to run a workload and then powers down the entire cluster. But this policy is more applicable to the batch jobs with longer intervals. Vasić et al. [35] presented a design for energy aware MapReduce and HDFS, where they leverage the sleeping state of machines to save power. Their work also proposeed a model for collaborative power management where a common control platform acts as a communication channel between the cluster power management and services running on the cluster. Maheshwari et al. in [36] proposed an energy efficient data placement and cluster reconfiguration algorithm that dynamically scales the cluster in accordance with the workload imposed on it to address the problem of energy conservation for large datacenters that run MapReduce jobs. However, the efficacy of this approach would depend on how quickly a node can be activated and the efficiency of storage space was still very low.

6 Conclusions and future work

The existing data placement strategy and replication scheme of HDFS can noticeably reduce MapReduce performance and may cause increasingly energy dissipation in heterogeneous environments. Besides that, HDFS employs an inflexible replica factor acquiescently for each data block, which will give rise to unnecessary waste of storage space when there is a lot of inactive data in Hadoop system. In this paper, we propose a novel snakelike data placement strategy (named SLDP) to solve these problems. The evaluation results show that SLDP is energy-efficient, space-saving and able to improve MapReduce performance in a heterogeneous Hadoop cluster.

In the future, we will focus on exploring more optimization techniques to make the power control policy more intelligent and efficient. We also want to figure out the optimal value of the block default size by performing a large number of experiments, thus the MapReduce performance can be fine-grained optimized based on that. Moreover, we plan to develop a dedicated Hadoop subsystem (called `amsHadoop`) based on Apache Hadoop-2.3.0 release by integrating our approaches proposed in this study to serve AMS-02 experiment data production and physics analysis better.

Acknowledgments This work is supported by National Natural Science Foundation of China under Grant Nos. 61320106007, 61202449, 61572129, 61502097, 61370207, National High-tech R&D Program of China (863 Program) under Grant No. 2013AA013503, China Fundamental Research Funds for the Central Universities under Grant No. 1109007115, Jiangsu research prospective joint research project under Grant Nos. BY2012202, BY2013073-01, Jiangsu Provincial Key Laboratory of Network and Information Security under Grant No. BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grant No. 93K-9, and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

1. Armbrust, M., Fox, A., Griffith, R., et al.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. IBM Big Data: [Online]. http://www.ibm.com/big-data/
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
4. Apache Hadoop: [Online]. http://hadoop.apache.org
5. Shvachko, K., Kuang, H., Radia, S., et al.: The Hadoop distributed file system. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, May 2010, pp. 1–10
6. Hadoop Wiki: Applications powered by Hadoop. [Online]. http://wiki.apache.org/hadoop/PoweredBy
7. Hadoop Distributed File System Architecture Guide: [Online]. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
8. White, T.: Hadoop-the Definitive Guide: Storage and Analysis at Internet Scale, 3rd edn. O'Reilly Media Inc, Sebastopol, CA (2012)
9. Zaharia, M., Konwinski, A., Joseph, A. D., et al.: Improving MapReduce performance in heterogeneous environments. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, Dec 2008, pp. 29–42
10. Mauch, V., Kunze, M., Hillenbrand, M.: High performance cloud computing. Futur. Gener. Comput. Syst. **29**(6), 1408–1416 (2013)
11. Amur, H., Cipar, J., Gupta, V., et al.: Robust and flexible power-proportional storage. In: Proceedings of the ACM Symposium on Cloud Computing, June 2010, pp. 217–228
12. Carns, P.H., Walter, I., Ligon, B., et al.: PVFS: a parallel virtual file system for Linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, Oct 2000, pp. 317–327
13. Microsystems, S.: Lustre file system: high-performance storage architecture and scalable cluster file system. Technical Report, Lustre File System White Paper (2007)
14. Ghemawat, S., Gobioff, H., Leung, S.: The Google file system. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, Oct 2003, pp. 29–43
15. Lin, M., Wierman, A., Andrew, L.L.H., et al.: Dynamic right-sizing for power-proportional data centers. IEEE/ACM Trans. Netw. **21**(5), 1378–1391 (2013)
16. Barroso, L.A., Holzle, U.: The case for energy-proportional computing. Computer **40**(12), 33–37 (2007)
17. Nan, Zhu, Xue, Liu, Jie, Liu, et al.: Towards a cost-efficient MapReduce: mitigating power peaks for Hadoop clusters. Tsinghua Sci. Technol. **19**(1), 24–32 (2014)
18. Xie, J., Yin, S., Ruan, X., et al.: Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In: Proceedings of the IEEE International Parallel & Distributed Processing Symposium, Workshops, April 2010
19. Apache Hadoop: Enable support for heterogeneous storages in HDFS. [Online]. https://issues.apache.org/jira/browse/HDFS-2832
20. Jain, K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. (CSUR) **31**(3), 264–323 (1999)
21. Iri, M.C., Ignjatovi, J., Bogdanovi, S.: Fuzzy equivalence relations and their equivalence classes. Fuzzy Sets Syst. **158**(12), 1295–1313 (2007)
22. Kaushik, R.T., Bhandarkar, M.: GreenHDFS: towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster. In: Proceedings of the 2010 International Conference on Power Aware Computing and Systems, June 2010, pp. 1–9
23. AMS-02 Experiment: [Online]: http://www.ams02.org/
24. Collaboration, A.M.S.: First result from the alpha magnetic spectrometer on the international space station: precision measurement of the positron fraction in primary cosmic rays of 0.5-350 GeV. Phys. Rev. Lett. **110**(14), 1–10 (2013)
25. Myung, J.: Tutorial on maximum likelihood estimation. J. Math. Psychol. **47**(1), 90–100 (2003)
26. Jittat, F., Bundit, L., Danupon, N.: Faster algorithms for semi-matching problems. ACM Trans. Algorithms **10**(3), 14–37 (2014)
27. Leverich, J., Kozyrakis, C.: On the energy (in) efficiency of Hadoop clusters. ACM SIGOPS Oper. Syst. Rev. **44**(1), 61–65 (2010)
28. Lang, W., Patel, J.M.: Energy management for MapReduce clusters. PVLDB **3**(1–2), 129–139 (2010)
29. Rafique, M.M., Rose, B., Butt, A.R., et al.: Supporting MapReduce on large-scale asymmetric multi-core clusters. ACM SIGOPS Oper. Syst. Rev. **43**(2), 25–34 (2009)
30. Fadika, Z., Dede, E., Hartog, J., et al.: MARLA: MapReduce for heterogeneous clusters. In: Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 2012, pp. 49–56
31. Guo, Z., Fox, G.: Improving MapReduce performance in heterogeneous network environments and resource utilization. In: Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 2012, pp. 714–716
32. Guo, Z., Fox, G., Zhou, M.: Investigation of data locality in MapReduce. In: Proceedings of the 12th IEEE/ACM International

Symposium on Cluster, Cloud and Grid Computing, May 2012, pp. 419–426
33. Zhang, X., Feng, Y., Feng, S., et al.: An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In: Proceedings of the International Conference on Cloud and Service Computing, Dec 2011, pp. 235–242
34. Jin, H., Yang, X., Sun, X.H. et al.: ADAPT: availability-aware MapReduce data placement for non-dedicated distributed computing. In: Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems, June 2012, pp. 516–525
35. Vasić, N., Barisits, M., Salzgeber, V., et al.: Making cluster applications energy-aware. In: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, June 2009, pp. 37–42
36. Maheshwari, N., Nanduri, R., Varma, V.: Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework. Future Gener. Comput. Syst. **28**(1), 119–127 (2012)

Runqun Xiong is a lecturer in School of Computer Science and Engineering, Southeast University, China. He received his B.S. degrees in applied mathematics from Southeast University in 2005, and received his Ph.D. degree in Computer Science from Southeast University in 2015. His current research interests include distributed and parallel computing, cloud computing and massive data storage management.

Junzhou Luo is a full professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He received his B.S. degree in applied mathematics from Southeast University in 1982, and then got his M.S. and Ph.D. degree in computer network both from Southeast University in 1992 and in 2000 respectively. His research interests are next generation network, protocol engineering, network security and management, cloud computing, and wireless LAN. He is a member of both IEEE and ACM, and co-chair of IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design.

Fang Dong is currently an associate professor in School of Computer Science and Engineering, Southeast University, China. He received his B.S. and M.S. degrees in Computer Science from Nanjing University of Science & Technology, China in 2004 and 2006, respectively, and received his Ph.D. degree in Computer Science from Southeast University in 2011. His current research interests include cloud computing, big data processing and task scheduling.