Modulo Based Data Placement Algorithm for Energy Consumption Optimization of MapReduce System

Jie Song · HongYan He · Zhi Wang · Ge Yu · Jean-Marc Pierson

Abstract With the explosion of data production, the efficiency of data management and analysis has been concerned by both industry and academia. Meanwhile, more and more energy is consumed by the IT infrastructure especially the larger scale distributed systems. In this paper, a novel idea for optimizing the Energy Consumption (EC for short) of MapReduce system is proposed. We argue that a fair data placement is helpful to save energy, and then we propose three goals of data placement, and a modulo based Data Placement Algorithm (DPA for short) which achieves these goals. Afterwards, the correctness of the proposed DPA is proved from both theoretical and experimental perspectives. Three different systems which implement MapReduce model with different DPAs are compared in our experiments. Our algorithm is proved to optimize EC effectively, without introducing the additional costs and delaying data loading. With the help of our DPA, the EC for the *WordCount* (https://src/examples/org/apache/hadoop/examples/), *Sort* (https://src/examples/org/apache/hadoop/examples/sort) and *MRBench* (https://src/examples/org/apache/hadoop/mapred/) can be reduced by 10.9 %, 8.3 % and 17 % respectively, and time consumption is reduced by 7 %, 6.3 % and 7 % respectively.

Keywords Big data · Data placement · Energy consumption optimization · MapReduce

Jie Song (✉) · HongYan He · Zhi Wang
Software College, Northeastern University, Shenyang, China
e-mail: songjie@mail.neu.edu.cn

Ge Yu
School of Information Science & Engineering, Northeastern University, Shenyang, China

Jean-Marc Pierson
IRIT, Paul Sabatier University, Toulouse 31062, France

1 Introduction

In recent years, huge amount of data have been accumulative along with the development of digital society [1]. Both the industry and academia are in pursuit of efficient data management and analysis. As we know, MapReduce [2], which is widely applied in many areas, is one of the most efficient way to deal with Big Data. Meanwhile, with the increase of IT instruments and the expansion of clustering system, the high Energy Consumption (EC for short) becomes one of the tough problems, so the EC optimization is a hot topic in both academia and industry [3–11]. In this paper, we treated a MapReduce system as an integration of both hardware and software, and study EC optimization of it from a software perspective.

In the MapReduce programing model, a job is divided into two continuous phases which are named as the map phase and reduce phase, then the latter does not start until the former is complete. In the map/reduce phase, map/reduce tasks

(mappers/reducers for short) are distributed to nodes and executed in parallel. Thus we can infer that the map/reduce phase is complete when the slowest mapper/reducer finishes, while the other nodes are idle and waste energy. Such idleness is passive, temporal and fragmental. To reduce the passive idleness, our previous study proved a load balance is an intuitive solution which ensures the parallelism of mappers/reducers [3]. Based on this, we argue that a static and predefined "modulo based data placement algorithm" optimizes the EC of a MapReduce system. Our approach balances the load by static data-oriented distributing without needing data access pattern or re-configuration, and the basic idea is reducing the passive idleness and the wasted energy consumed during the idleness by improving parallelism of mappers/reducers, then improving parallelism by ensuring that mappers/reducers are completed simultaneously, then adjusting the execution time by a fair data placement, then achieving the fair data placement by a modulo based data placement algorithm. In fact, modulo operator is a simple and effective hush function. With the modulo operation in our algorithm, the data blocks can be distributed to each node fairly according to the nodes capability and data features. In a MapReduce system, data placement is the distribution state of data among nodes. The data placement is fair if it satisfies "fairness of size", "fairness of range", and "best adaptability" (see Section 3.3). We name these characteristics of fair data placement as data placement goals, and the algorithm to implement these goals as Data Placement Algorithm (DPA for short).

Taken mapper as an example, we explain why its execution time, which is the key to improving parallelism, is dominated by the data placement. In a MapReduce system, mappers are distributed to nodes for processing their local data, respectively, thus each node will perform the same mapper. The complexities of mappers on each node are exactly the same, so the execution time of a mapper is dominated by features of its input data, such as the data amount and value distribution. These characteristics of data are also features of data placement.

Hadoop, which implemented the MapReduce programing model, ensures the parallelism by allowing a faster node to fetch data from slower nodes remotely. In this case, the parallelism could not be worse than the situation where the slowest node processes a data block from the farthest node remotely while other nodes are idle. The cost of such strategy is remote I/O, the parallelism is improved but processors are also not fully utilized because they are waiting for the remote data access. We argue that the data size a task processes dominates its execution time in a MapReduce system, and in a Hadoop system the node does not only run tasks but also stores data. So the best situation, where all the tasks only process their local data and are completed simultaneously, could happen if the data are fairly placed among these nodes. The fair data placement, which is defined definition 6 to 8, means data placement is fair in terms of data amount, of access patterns, and of response time when data are processed.

Above all, it is possible to define a proper data placement that can improve the parallelism of nodes, reduce their idleness, and optimize their EC. Our contributions in this paper are studying the characteristics of data placement which optimizes EC of MapReduce systems, defining the data placement goals, and proposing a modulo based DPA. We prove that the MapReduce system with the proposed data placement consumes less energy than those with random data placement. Theoretical proof and experiments show that the DPA implements the data placement goals well in a heterogeneous system. The data placement is a classical topic. There are some relevant works which use data placement to achieve optimize energy though [9–13], all of them are dynamic optimization, such as turn off cluster nodes or right the size of resource allocations, and the cost of rebalancing or the overhead on the network cannot be ignored. However, in our work, it is a static way which balances the load not by dynamic tasks-oriented scheduling, but by static data-oriented distributing with the good adaptability, because the execution time of tasks is dominated by the data characteristics of the nodes.

The rest of this paper is organized as follows. Following the introduction, Section 2 introduces the related work. Section 3 analyzes the EC of MapReduce system and proposes the idea of EC optimization. Section 4 introduces the data placement model, and defines the goals of data placement on EC optimization. Section 5 proposes a modulo based DPA which is designed for heterogeneous systems. Section 6 proves that the modulo based DPA satisfies the desired goals. Section 7 explains the experiments and results. Section 8 summarizes this paper and presents the further works.

2 Related Work

There are some productive approaches, which are classified into three categories, have been proposed in the area of energy consumption (EC for short) optimization. First, energy can be saved by adjusting the CPU state [4, 5]. Second, energy is saved through scheduling jobs or tasks dynamically, then the idle node is shut down temporarily. It is also known as workload concentration [6]. Third, EC is reduced by improving the utilization of computer resources, then less energy is wasted. The former two are not suitable for MapReduce system [7]. The reasons are as follows. Firstly, in a highly utilized system, the energy gains through CPU scaling is low (less than 5 %) and also has a negative impact on performance. Secondly, nodes in MapReduce system are not only for computation but also for storing data. Shutting down or hibernating nodes makes the part of data unavailable. What's more, it is costly to migrate the virtual machines with their data. The latter one is suitable to the MapReduce system. Our approach reduces the idleness of nodes by a balanced data placement, and further reduces EC.

There are some relevant works which optimize energy through data placement. Maheshwari et al. proposed an algorithm that dynamically reconfigures the cluster based on the current workload and turns cluster nodes on or off when the average cluster utilization rises above or falls below administrator specified thresholds, respectively [8]. It is a dynamic data re-placement strategy. The energy is saved though shutting down nodes, and the cost of data transfer is remarkable. Xiong et al. dynamically righted the size of resource allocations to the parallelized tasks such that the effective hardware configuration matches the requirements of each task [9]. It is the dynamic optimization. But our approach statically allocates data into each node, and reduces the complexity of cluster operation, such as switching-off cluster and re-configuration. Moreover, the overhead on the network while rebalancing is high in Xiong's approach. Palanisamy et al. provided MapReduce clusters the locality-aware manner in order to reduce the data transfer overhead among nodes [10]. However, it did not consider the heterogeneity of MapReduce system. Our approach is at software-level which uses algorithm to allocate data into each node. Moreover, the overhead on the network while rebalancing is high in those approaches. In our paper, the adaptability is good. When a node crashes and data are re-allocated, the overhead on the network is only the amount of data in that node. Our work is inspired from earlier works of Pinheiro et al. where they present the problem of load balancing and unbalancing for power and capability in cluster-based systems [11]. The system configuration for a MapReduce cluster has a high impact on its energy efficiency [12, 13], and we argue that the data placement, even if static, also leverages energy efficiency.

There are some relevant works which optimize energy through task scheduling. Chen et al. classified cluster into interactive zone which services interactive jobs and batch zone which services batchable and interruptible jobs [14]. Then the energy can be saved when the batch region transforms into a low-power state. However, there is a challenge when most jobs are large and batch jobs, which can stop the batch region transforming into low-power state. In MapReduce system, the data is processed in parallel. No matter what the types of jobs are, we partition them into the datasets with balance size in our algorithm. Therefore, our algorithm isn't influenced by the types of jobs. Yigitbasi et al. proposed an energy-aware scheduling strategy based on heterogeneity in MapReduce cluster, which schedules the task to the most efficient node [15]. This is a cluster-level optimization, and it negatively affects the performance, availability and fault-tolerance of the cluster. However, our approach is a software-level optimization whose cost is not the run-time performance but the additional computation when data is loaded.

There are some relevant works which optimize energy through migrating data dynamically. The Popular Data Concentration (PDC) migrates data across disks according to their access frequency or popularity [16]. GreenHDFS, which was developed by Yahoo!, saves energy by migrating files from active nodes to hibernated nodes [17]. Besides, the Massive Array of Idle Disks (MAID) technique uses extra cache disks to cache recently accessed data [18]. All these algorithms or strategies are dynamical approaches which rely on data access patterns. The systems, when applying these algorithms or strategies, migrate data frequently because applications have various data requirements and data access patterns. On the contrary, our idea is to build a pre-defined and balanced data placement

which improves the parallelism and reduces the I/O cost for any applications in a MapReduce system. Our approach is static, and the data placement is a predefined strategy which is independent from the data access patterns. The benefit is permanent once the goals of data placement are satisfied.

As far as data placement algorithm is concerned, the most popular goals of the traditional DPAs are promoting the parallelism, fault-tolerance and reliability of large-scale distribute storage systems. The most typical DPA is consistent hashing [19]. In this algorithm, every physical device is virtualized to $k\log|N|$virtual devices. k is a constant and N is the number of physical devices. The homogeneous system, whose DPA is implemented by the consistent hashing, has a good adaptability and load-balance. But in a heterogeneous system, the consistent hashing algorithm is inefficient. Brinkmann et al. introduced *levels* to improve the efficiency of consistent hashing algorithm in the heterogeneous storage area networks, to treat the data placement in non-uniform disks as that in uniform ones [20]. Chen Tao et al. proposed a DPA for the large scale network storage systems named CCHDP, which adapts the changes of storage scale and balances the load of each device [21]. In their algorithm, fewer virtual devices are introduced than consistent hashing in a heterogeneous environment, but the longer the system runs, the more complex it is to locate the specific data. Specially. in Hadoop, the data placement assumes that each node in this cluster has the same capacity, and distributes the blocks to each node averagely, which reduces data transfer and enhances the effectiveness. By default, each data block has three replicas in HDFS, and the one replica is placed on a node in the local rack, another on a node in a remote rack and the last on a different node in the same remote rack. But this strategy doesn't work well in the heterogeneous environment.

Some other goals of DPAs are reducing remote data access and then improving the performance of network, especially for the data intensive applications. For example, a data placement algorithm based on BEA and Kmeans was proposed by Yuan et al. [22]. Matrix is used to show the relationship between each data block and task, which stores the dependencies between data blocks. When distributing data, it will put the data which have strong dependence together, in order to reduce the data immigration. However, this algorithm doesn't consider the problem of load balance.

In comparison with the existing works about EC optimization and DPA. Our solution, which optimizes EC by a static DPA, is a novel one. What's more, some goals of the traditional DPA are not suitable for a MapReduce system. Consequently, according to the characteristics of MapReduce system, we propose three data placement goals and design a modulo based DPA for the EC optimization. Experimental results in Section 7 will prove the advantages of our solutions.

3 Energy Consumption Analysis

The first object we studied is MapReduce system. A MapReduce system is the integration of both hardware and software layer. The hardware layer of MapReduce system consists of a larger number of computers. The nodes store massive data and perform data analysis. The software layer of MapReduce system consists of a MapReduce framework (e.g. Hadoop MapReduce) and a distributed file system (e.g. HDFS). In the distributed file system, data are distributively stored in the nodes. In each node, the local data are processed before the remote data.

The second object we studied is energy consumption (EC). The EC of computers includes energy consumed by CPU, disk, memory, network card, video card, mainboard and other accessories. According to data provided by *msdn.microsoft.com* [23], the power consumed by CPU and I/O to disk and network are not ignored. They are 47.2 % and 23.5 % in the idle power usage, respectively. We exclude the EC of the monitor, keyboard, mouse and other devices for human-computer interaction. Generally, part of energy consumed by the hardware during the execution of the job in a MapReduce system, could be optimized by the software techniques. In this section, we analyze this part of energy consumption and propose an idea to optimize it.

Definition 1 Waiting Energy Consumption. In a MapReduce system, the energy is wasted when some nodes are in "passive idle" or "busy idle" state because they are waiting for other resources, such energy consumption is named as the Waiting Energy Consumption (waiting EC for short).

For example, a node is switched off when it is idle, otherwise it wastes energy. In another situation, a node also wastes energy when it is waiting for the computation results of other nodes; however, it is not exactly idle so we can not shut it down. Similarly, for the components of a computer (node), the CPU also brings waiting EC when it is waiting for local or remote data access. Therefore, when we reduce these waiting situations, waiting EC is minimized.

The definition of waiting EC matches the characteristics of MapReduce system well. In a MapReduce system, on the one hand, a job is divided into tasks (mappers and reducers) which are deployed on nodes, so it is possible that some nodes wait others because of weak parallelism among tasks. On the other hand, the most jobs in MapReduce are I/O intensive. Therefore, CPU idle is the main cause of waiting EC because the capability of CPU is much better than that of local and remote I/O. Our previous study has proved that increase parallelism and decrease remote I/O can reduce waiting EC efficiently.

Hadoop MapReduce ensures the parallelism by allowing a faster node fetch data from slower nodes remotely. For example, the parallelism could be no worse than the situation that the slowest node processes a data block from the farthest node remotely while the others are idle. The cost of such strategy is remote I/O, the parallelism is improved but waiting EC may not decrease because processors are waiting for the remote data access. The execute time consumption of a task is mainly dependent on its complexity and input scale. The complexities of all mappers are the same while the scale and location of inputs are dominated by the data placement. If the data are distributed to each node fairly according to the nodes capability and data features, then there is only local data access of each task and tasks will be completed simultaneously. The parallelism is ensured and remote I/O is minimized. The data placement goals, which are "fairness of size", "fairness of range" and "best adaptability", are explained in the next section.

4 Data Placement Model and Goals

In this section, we will define a data placement model which includes data set , data block, node and node Capability, then based on the mode, the goals to explain the "fair data placement" are given. In our paper, a MapReduce system C is the set of nodes $\{c_1, c_2,\ldots,c_n\}$which have different capabilities, namely it is a heterogeneous system, and n is the number of nodes.

Definition 2 Data Set and Data Block. The data set B, which contains many data blocks, is all the involved data for a specific job running in the MapReduce system. Let $B = \{b_0, b_1,\ldots,b_{m-1}\}$and each data block have the same size. m is the number of data blocks and the subscript is the fixed and the increasing index of data blocks which is encoded in an arbitrary order.

Under the definition 2, data placement is just a mathematical function. Let data blocks and nodes be two sets, and the DPA is a function of two sets because every block should be stored in a certain node, as is formally defined in Definition 5.

Definition 3 Node Capability. The node capability represents how efficient the node processes the data. Generally, it depends on the CPU and I/O performance. The devices, such as CPU, memory, disk and network are comprehensively considered. The capability of node c_j is represented by w_j. It is a normalized value. $w_j \in (0,1)$.

In this paper, we evaluate the node capability by benchmark tool unixbench which is a general-purpose benchmarking suit that has been designed to provide the basic performance evaluation for a unix-like system, and provide a single value to indicate the node capability [24]. We introduce a terms which are frequently used in the post sections.

Definition 4 Cumulative capability. In a heterogeneous system, the nodes are encoded from 1 to n(decimal code) in a certain (arbitrary) order. $\forall\ c_j \in \{c_1, c_2,\ldots,c_n\}$, the cumulative capability of node c_j, denoted as $\sum w_j$ is the sum of the capabilities of nodes from c_1 to c_j. $\sum w_j = w_0 + w_1 + \ldots + w_j$.

Definition 5 Data Placement Algorithm (DPA). A data placement algorithm maps blocks to nodes. $\forall b_j \in B$, data placement algorithm is a function $f_0{:}B \rightarrow C$. That maps b_i to a specific c_j, $\exists! c_j \in C$, $c_j = f_0(b_i)$. The number of blocks that are distributed to c_j is $|f_0^{-1}(c_j)|$.

In this paper, a DPA is a mapping function which maps B to C. Definition 5 is a general definition of DPA. For example, the DPA of Hadoop Distributed File System which divides files into blocks with an arbitrary size (64MB as default) and stores every block in one of the nodes randomly, is also satisfies the Definition 5. In Section 4, we describe the modulo based DPA, and denoted it as f_0, by which the data placement satisfied the "fairness of size", "fairness of range", and "best adaptability", are realized.

First, we distribute more data to the nodes with higher capability, and less data to the nodes with lower capability. We name this goal as "fairness of size ".

Definition 6 Fairness of Size (f-size). The data placement algorithm in a MapReduce system achieves fairness of size (f-size for short) if $\forall c_j \in$ C:

$$|f_0^{-1}(c_j)|/m = w_j/\sum w_n \quad (1)$$

In (1), m and n are the number of blocks and nodes, respectively. w_j is the node capability of c_j, $\sum w_n$ is cumulative capability of c_n, and $|f_0^{-1}(c_j)|$is the number of blocks that are distributed to c_j.

Generally, a MapReduce job includes two processes: querying data and analyzing the query results. Therefore, "fairness of size" can only ensure the equitable size of input data but not that of query results which will be analyzed later. Given a query with a random condition, we need another goal which is named as "fairness of range" to guarantee the equitable size of query results in each node.

Definition 7 Fairness of Range (f-range). The data placement algorithm in a MapReduce system achieves fairness of range (f-range for short) if $\forall c_j \in$C:

$$h(c_j)/m = w_j/\sum w_n \quad (2)$$

In (2), m and n are the number of blocks and nodes, respectively. w_j is the node capability of c_j, $\sum w_n$ is cumulative capability of c_n. Given a random query, $h(c_j)$ is the number of blocks in the query results of c_j. The DPA which achieves f-size and f-range ensures that the data been queried by mapper and analyzed by reducer are distributed among the nodes equitably.

We also consider the adaptability of DPA because adding and removing nodes are normal in a MapReduce system. For one reason, the hardware failure of a MapReduce system is common if the hardware layer adopts commodity cluster whose reliability can be low. For another reason, the number of nodes is shifting frequently for scale-out capability. Consequently, to maintain the f-size and f-range when a node is added or removed, the DPA moves the existing data to the new node, or moves the data from the removed node to the others. The data transferring through network is costly, so that minimizing data transferring is a must for achieving adaptability.

Definition 8 Best Adaptability (b-adapt). The best adaptability of data placement algorithm (b-adapt for short) ensures that the size of transferred data over network is equal to the size of data in the node which is newly added or removed in order to maintain the f-size and f-range. It satisfies the following two conditions:

1. When a new node $c_{n_{+1}}$ is added to the MapReduce system, then f_0 changes to f_0^+. They satisfy: $\forall b_i \in B$, if $f_0^+(b_i) \neq c_{n_{+1}}$, then $f_0^+(b_i) = f_0(b_i)$.
2. When a node c_j is removed from the MapReduce system, then f_0 changes to f_0^-. They satisfy: $\forall b_i \in B$, if $f_0(b_i) \neq c_j$, then $f_0^-(b_i) = f_0(b_i)$.

In summary, if the fairness of size is satisfied, each node can spend same time on data query; if the fairness of range is satisfied, each node can spend same time on data computation; if the Best Adaptability is satisfied, data transferring can be minimized. Therefore, DPA should be designed to satisfy goals of "fairness of size (f-size)", "fairness of data range (f-range)", and "best adaptability (b-adapt).

5 Modulo Based DPA

In this section, the modulo based DPA (f_0), which is designed for a heterogeneous system, is introduced. In a heterogeneous system, capability of nodes is various, and some nodes may be unavailable in a certain time.

Definition 9 Unavailable Factor. To indicate the timestamp of a node c_j being unavailable, unavailable factor η_j of node c_j is the latest index of block which is distributed to c_j. η_j is initialized as $+\infty$, maintained by the master node, and updated when c_j is unavailable.

On condition that there are many more blocks than the nodes ($n\langle\langle m$), the modulo based DPA is defi-

ned as the function f_0:$B \rightarrow C$: $\forall i \in [0,m\text{-}1]$, $\forall j \in [1,n]$, given $b_i$andc_j, the b_i is mapped to c_j ($f_0(b_i) = c_j$) if they are satisfy (3).

$$\begin{array}{ll} (1) & i\%\sum w_j - o_j \in [0, w_j) \\ (2)\ \text{and} & \forall k \in (j, n]\, i\%\sum w_k - o_k \notin [0, w_j) \\ (3)\ \text{and} & \eta_j \geq i \end{array} \quad (3)$$

We also define the computation for %:

$$x\%y = x - \lfloor x/y \rfloor \quad (4)$$

In (3), w_j and $\sum w_j$ is the capability and cumulative capability ofc_j, respectively. $w_j \in (0, 1)$. As we know, the result of $i\%x$ is always i if $x > i$, so that w_j is a normalized value for the effectiveness of modulo operation (see Definition 3).

In (3), o_j is a pre-defined, random and static offset of the node c_j. The random offset o_j ensures that blocks are selected randomly which contributes to both f-size and f-range (proved in Section 5). o_j is calculated as (5).

$$o_j = \begin{cases} p_j \times \sum_{k=1}^{j-1} w_k & (j > 1) \\ 0 & (j = 1) \end{cases} \quad (5)$$

The pseudo-code of the f_0 is shown as Algorithm 1.

In Algorithm 1, the nodes are traversed backwardly (line 4), and only condition (1) is verified in line 5. Because in (1), condition (2) is similar with condition (1). It makes sure that c_j is the last node from c_1 to c_n which satisfies condition (1). If a forward traversal is adopted, both condition (1) and condition (2) should be verified. Figure 1 shows an example of distributing 15 data blocks to 5 heterogeneous nodes. In the example, the nodes' weights are 0.1, 0.3, 0.5, 0.6 and 0.4 respectively; the nodes' p_j are 0, 0.7, 0.5, 0.3 and 0.2 respectively. The 3rd node crashes when block No. 12 has just been distributed, and then block No. 13 and 14 are imported to the system.

Algorithm 1 f_0: The modulo based DPA

Input: B,C
Output: distribute all the b_i in B to all the c_j in C
Function f_0
1. **Encoding** data blocks from 0 to m-1 (decimal code) in a certain (arbitrary) order, $B = \{b_0, b_1, \ldots, b_{m-1}\}$;
1. Encoding nodes from 1 to n(decimal code). in a certain (arbitrary) order, $C = \{c_1, c_2, \ldots, c_n\}$;
2. For $i = 0$ **To** $m-1$
3. For $j = n$ **To** 1
4. If $i\%\sum w_j - o_j \in [0, w_j)$ and $\eta_j \geq i$
5. *distribute* b_i *to* c_j
6. Continue
7. End If
8. End For
9. End For

For the 5th node, the blocks which satisfy equation (i%1.9-0.2×1.5)∈[0, 0.4) are No. 6, 8, 10 and 12. Therefore, block 6, 8, 10 and 12 are distributed to the 5th node.

For the 4th node, the blocks which satisfy equation (i%1.5-0.3×0.9)∈[0, 0.6) are No. 2, 5, 8, 11 and 14, but No. 8 has been already distributed. Therefore, block 2, 5, 11 and 14 are distributed to the 4th node.

For the**3rd node**, the blocks which satisfy equation (i%0.9-0.5×0.4)∈[0, 0.5) are No. 3, 4, 5, 12, 13 and 14, but 5 and 12 have been distributed, besides, the 3rd node is unavailable when block 13 and 14 are imported to the system. Therefore, block 3 and 4 are distributed to the 3rd node.

For the 2nd node, the blocks which satisfy equation (i%0.4-0.7×0.1)∈[0, 0.3) are No.1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13 and 14, but No. 2, 3, 5, 6, 8, 10, 11 and 14 have been distributed. Therefore, block 1, 7, 9 and 13 are distributed to the 2nd node.

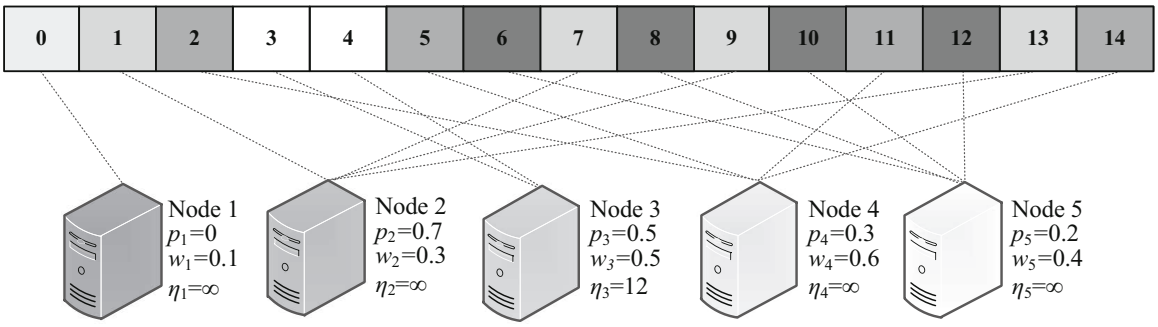


Fig. 1 An example of the modulo based DPA

For the 1st node, all the blocks satisfy the equation (i%0.1-0)∈[0, 0.1), but all the blocks except the No.0 have been distributed. Therefore, only block 0 is distributed to the 1st node;

According to f_0, blocks 3 and 4 are unreachable if the 3rd node crashes. To avoid this situation, a fault-tolerant mechanism is introduced in Section 6. This example doesn't achieve f-size and f-range because it doesn't satisfy $n\langle\langle m$. We prove that f_0 achieves f-size and f-range for $n\langle\langle m$ in Section 6.

6 Proof of Fairness and Adaptability

In this section, we prove that the modulo based DPA (f_0) achieves the f-size, f-range and b-adapt. For proving easily, another DPA, which is an equivalent of f_0, is proposed and named as f_0'. The pseudo-code of the f_0' is shown as Algorithm 2.

f_0' is explained as procedures of adding node one by one. According to this algorithm, there is only one node c_1 in the system in the beginning and all the data blocks are distributed to c_1. When a node is added, some data blocks are selected and redistributed to the new node. Such procedure is repeated until there are n nodes in the system. In conclusion, f_0 is a blocks oriented algorithm which traverses nodes descendingly, and f_0' is a node oriented algorithm which traverses blocks ascendingly. f_0 is more efficient than f_0', but they are equivalence. We plan to f_0' achieves f-size, f-range and b-adapt , so as f_0.

Proof **f_0' satisfies f-size.**

1) When j =1, there is only one node in the system. All the blocks are distributed to c_1. Thus, f_0' achieves the f-size.
2) Assuming that f_0' achieves f-size when there are j-1 nodes in the system, we can prove that f-size is achieved when a new node c_j is added. The proof is given as below.

Algorithm 2 f_0': Equivalent algorithm of the modulo based DPA

Input: B, C
Output: distribute all the b_i in B to all the c_j in C
Function f_0'
1. **Encoding** data blocks from 0 to m-1 (decimal code) in a certain (arbitrary) order, B ={b_0, b_1,…,b_{m-1}};
2. **Encoding** nodes from 1 to n(decimal code). in a certain (arbitrary) order, C ={c_1, c_2,…,c_n};
3. **For** i = 0 **To** m−1
4. **For** j = 1 **To** n
5. **If** $i\%\sum w_j - o_j \in[0,w_j)$ and $\eta_j \geq i$
6. **If** b_i has been distributed to {c_1, c_2,…,c_{j-1}}
7. remove the distribution of b_i
8. **End If**
9. *distribute* b_i *to* c_j
10. **Continue**
11. **End If**
12. **End For**
13. **End For**

Figure 2 shows the data placement of f_0' when node c_j is added. Based on the modulo operator (%), the range of results of $i\%\sum w_j$ is 0 to $i\%\sum w_j$-1 (i is block index). Therefore, all blocks are encoded and divided into$m/\sum w_j$ parts, and each part contains $\sum w_j$ data blocks except the last part. Moreover, w_j blocks will be selected in each part based on the condition$i\%\sum w_j - o_j \in[0,w_j)$.

∵ There are $m \cdot w_j/\sum w_j$ data blocks which are distributed to c_j.
∴ The number of data blocks distributed to c_j satisfies the (1).
∵ For each part, the probability of each data block being selected and re-distributed to c_j is equal.

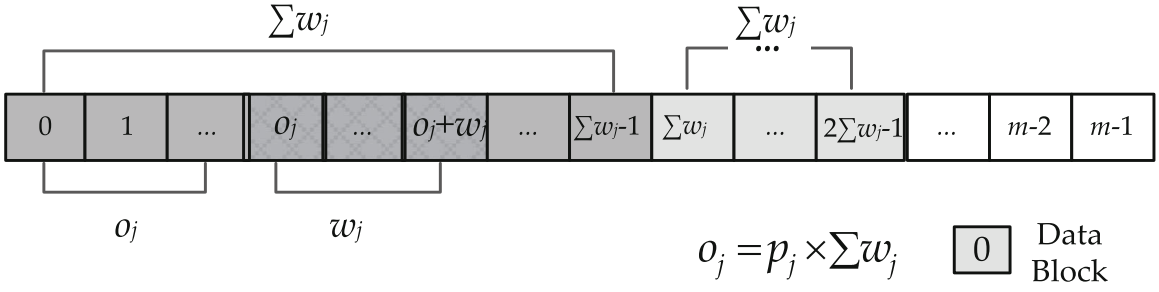


Fig. 2 The data placement of f_0' when the node c_j is added

$\therefore$ Each data block has the same probability of being re-distributed to c_j, so after adding c_j to $\{c_1, c_2, \ldots, c_{j-1}\}$, the re-distributed data blocks satisfy the (1).

$\therefore$ Hence the assertion 2) is satisfied.

In conclusion, $\forall j \in [1,n]$, the data placement satisfies (1) according to the mathematical deduction, and f_0' satisfies f-size. □

Proof **f_0' satisfies f-range**

$\because$ In f_0', a data block, which is distributed to a node, is randomly selected because of the pre-defined and randomo_j in (3).

$\because$ If a block is randomly distributed to a node, then for any queries, the query results are randomly distributed in nodes. That is, because of the randomness of data placement, the possibilities of a block being selected are definitely not same for different queries, but the possibilities of each node being selected are same for any query

$\therefore$ Equation 6 is satisfied.

$$|f_0'^{-1}(c_j)|/m = h(c_j)/m \tag{6}$$

In (6), $|f_0$'$^{-1}(c_j)|$is the number of blocks that are distributed to c_j, and $h(c_j)$ is the number of blocks that are selected in c_j.

$\because$ The f_0' achieves f-size, then (7) is satisfied.

$$|f_0'^{-1}(c_j)|/m = w_j/\sum w_n \tag{7}$$

$\therefore$ Equation 8 is satisfied.

$$h(c_j)/m = w_j/\sum w_n \tag{8}$$

In conclusion, f_0' achieves f-range. □

Proof **f_0' satisfies b-adapt.**

To prove that the f_0' has the best adaptability (b-adapt), we consider the following three situations: 1) scale-out: adding a node to the system, 2) scale-in: removing a node from the system, 3) Fault-tolerance: a node crashes unpredictably.

For scale-out: when a node c_j is added, some data blocks are selected and re-distributed to c_j. The process of scale-out is same with one iteration of f_0' (line 4 to 12 in Algorithm 2), so that f-size and f-range are not broken. Meanwhile, the transferred data blocks over network, which are selected and re-distributed to c_j,are minimum. In conclusion, f_0' satisfies b-adapt.

For scale-in: Nodes are ordered by their indexes. The process of removing the last node can be treated as the inverse-process of adding a new node in f_0', so both f-size, f-range are still satisfied. The transferred data blocks over network are equal to the data blocks which are selected and re-distributed to $c_m$when $c_m$is added, thus they are minimum. In conclusion, if we only remove the last node (node with the largest index), f_0' satisfies b-adapt. The "removing the last" is a constriction of scale-in, but it is not a strict one because scale-in is an active adjustment on clusters; administrator can decide which server is removed.

For fault-tolerance: we analyze the following three situations when node c_j $(j<n)$ crashes and data blocks in c_j (e.g. b_i) are lost. According to the definition of f_0':

1) If both c_j and b_i can be recovered, or we can find a c_j' $(w_j' = w_j)$ to replace c_j, then it simply distributes b_i to c_j', and adds c_j' to cluster, then the data placement is recovered to the one before c_j crashed. Both f-size and f-range are not broken. The transferred data blocks over network, which are blocks on c_j,are minimum.
2) If $c_j$and b_i are lost permanently, then the data placement of the other nodes is still satisfies f-size, f-range and b-adapt. There are not data blocks transferred over network. As it is defined in definition 9, η_j is maintained by the master node, so it is accessible since c_j is lost. η_j is not $+\infty$ anymore, and has been updated to the last index of block onc_j. The index j of c_j is also reserved. Based on it, client is notified that $c_j$is unavailable when b_i is queried, and for the new data blocks imported after c_j crashes, they are distributed to the remaining nodes according to the condition (3) of (3).
3) If b_i is recovered but c_j is crashed permanently, b_i is imported to the remaining system again as a newly-encoded data block. For the remained system, b_i is a new data block. Thus, the data placement still achieves f-size, f-range after c_j crashed and b_i is re-distribute. The transferred data blocks over network, which are blocks on c_j, are minimum

For both of three situations above, it is easy to deduce that when a node is added or removed, to maintain the f-size and f-range, the data blocks transferred over network are equal to the data

blocks in the node which is newly added or removed. In conclusion, f_0' satisfies b-adapt.

All in all, according to the Proof 1 to 3, it is proved that f_0 achieves f-size, f-range and b-adapt. □

7 Algorithms Comparisons

Section 6 proves that f_0 is satisfied with data placement goals, but it can not prove that f_0 is better than classic DPAs. In order to verify the effectiveness of f_0, the comparisons are performed in the simulation environment where the results can be drawn without truly distributing data blocks. The four algorithms as competitors are briefly described below:

1) *Consisitent hash*(f_1) [19]: every node is virtualized to $k\log|N|$virtual devices. k is a constant and N is the number of node. Then mapping these virtual devices into the unit ring.
2) *Dynamic interval mapping*(f_2) [25]: it divides unit interval into some sub-intervals based on the weights of devices, and there is a mapping between devices and sub-intervals. When loading data, the data which fall into an interval are distributed to the corresponding devices. When adding devices, the sub-intervals are divided into smaller intervals based on the weights, then distribute these smaller intervals to added devices(named cluster) and transfer the data which fall into these intervals into added devices. We increase the number of clusters in order to decrease the impact of heterogeneity.
3) *An algorithm based on BEA and K-Means*(f_3) [22]: each data set is interdependent based on task requirements. When distributing, it will put the data which have the strong independence into the same node; in this way, it reduces the time of moving data between nodes. However, it doesn't take load balance into account.
4) *Greedy algorithm*(f_4) [26]: it is a common DPA. The DPA firstly satisfies the requirement of node with the largest weight, then that of the second node until all the data are distributed. It is simple and fast. However, it is unfair to the nodes which have the small weights if the amount of data is not huge enough.

We simulate 100 nodes whose performance values are random integer ranging from 5 to 150. We adopt the well-known *Grep* cases [31] in the simulation. The artificial data set consists of 300000 data blocks and 0.3 billion of words (1000 words per block). When generating the data, both query condition (2 characters like "*AB*") and words (50 characters) are strictly selected from 26 English letters. A 50-length string contains 49 different 2-characters-substrings, and the probability that a 2-characters-substring matches the query condition is about $1/26^2$, so the hit rate is about $49/26^2 \approx 0.072$.

We compare the f-size and f-range between these algorithms because adaptability of f_0 has been proved to be the best in Section 6. In the experiment of f-size, if the number of data blocks on each node has the linear correlation with node's capabilities, then f-size is satisfied. In the experiment of f-range, a random query simulator is designed to get the selected data blocks of each block. If the amount of selected data on each node has the linear correlation with node's capabilities, then f-range is satisfied. Pearson correlation coefficient is a measure of the linear correlation (dependence) between two variables, giving a value between $+1$ and -1 inclusive, where 1 is total positive

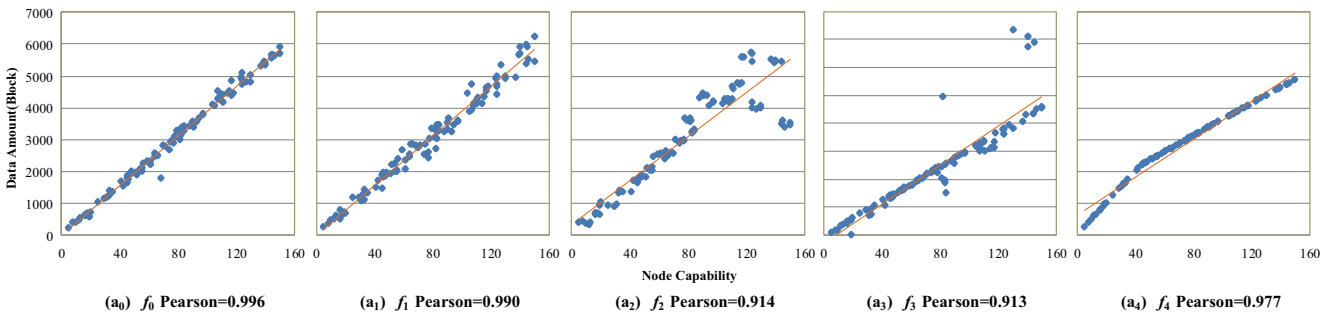


Fig. 3 The comparison of f-size among five DPAs

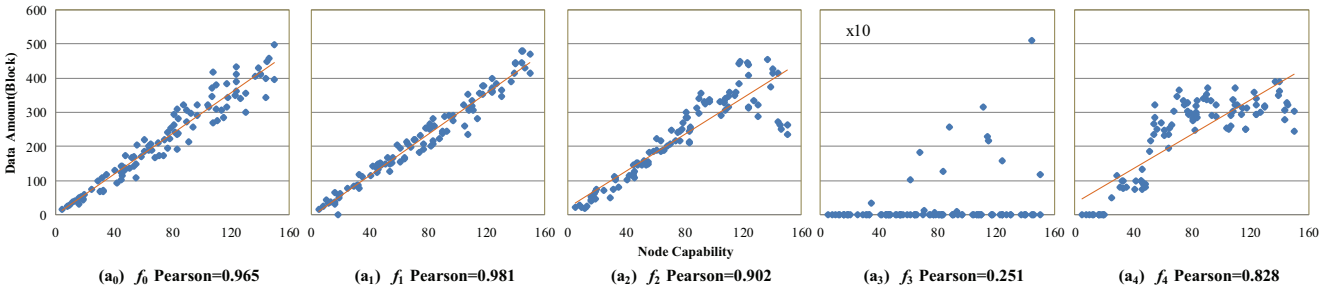


Fig. 4 The comparison of f-range among five DPAs

correlation, 0 is no correlation, and −1 is total negative correlation. In experiments, pearson correlation coefficient is adopted to compare the f-size and f-range of DPAs.

7.1 Fairness of Size

In this experiment, we compare the f-size among five algorithms. The comparison results are shown in Fig. 3. The axis y represents the number of data blocks; axis x represents the capability of each node. The stronger of linear relation between node capability and the number of data block is, the better f-size is. The f-size of f_0 is much better than f_2, f_3, f_4 and slightly better than f_1. After calculating, the coefficients of f_0, f_1, f_2, f_3 and f_4 are 0.996, 0.990, 0.914, 0.913 and 0.977 respectively. Therefore, the proposed f_0 has the best f-size.

The f-size of f_0 is slightly better than that of f_1, and the former is more independent to the number of nodes. The fairness of f_1 is dominated by the number of nodes, and when more nodes are introduced, time consumption of f_1 increases. On the contrary, the time consumption and f-size of f_0 are independent from the number of nodes. Moreover, f-size of f_2 is weak, and it requires additional space to store the mapping interval. The f-size of f_3 is worse because it only considers the dependency of data. f_3 ensures the dependent data are in the same node, and after clustering the data block into groups according to their similarities, it is traditional bin packing problem. f_4 does not take the overall conditions into consideration, so there is no data distributed to nodes which have smaller weights since the data blocks are not huge enough.

7.2 Fairness of Range

In this experiment, we compare the f-range among five algorithms under the same data, query and node capability. The comparison results are shown in Fig. 4. The axis y represents the number of selected data blocks; axis x represents the capability of each node. Notice that the scale of y- axis in Fig. 4-a_3 is from 0 to 6000, and which in Fig. 4-a_0, a_1, a_2 and a_4 are all from 0 to 600. It is obvious that the f-range of f_0 and f_1 are far better than that of other three. The Pearson correlation coefficients of f_0, f_1, f_2, f_3 and f_4 are 0.965, 0.981, 0.902, 0.251 and 0.828 respectively. Therefore, the f-range of f_0 is slightly worse than f_1 and far better than the other three.

Although the f-range of f_1 is slightly better than f_0, the time consumption of f_1 is three time larger than that of f_0. For proposed f_0, the slight weakness of f-range is offset by performance advantage. For f_2, f-range is dominated by f-size because Fig. 4-a_2 accords with Fig. 3- a_2. f_3 ensures the queried data are centralized in several nodes, so only a few nodes are involved in query results, and f_4 does not take f-range into consideration.

Table 1 The testbed of experiments

Brand	CPU	Memory	Disk	Net work	Job Tracker	Task kTracker
TongFang Z900	Inter Core i5 2.80GHz	8GB	1TB	1000M	1	5
TongFang C3001	Inter Core Pentium(R) 4 3.00GHz	1GB	80GB	100M	0	4

Table 2 The differences of three MapReduce system

MapReduce System	MapReduce Module	DFS Module	DPA	Data Location
Hadoop	Hadoop MapReduce	Hadoop DFS	Random	Local & Remote
LocalHadoop	Modified Hadoop MapReduce	Local File System	Random	Local
NeoHadoop	Modified Hadoop MapReduce	Local File System	Modulo based	Local

Moreover, from Fig. 3-a_0, we can see that, f-size will not be worse with the increase of data blocks; based on Fig. 4-a_0, f-range is hardly influenced by the node capacity, because of the linear relation between node capability and the number of data block proved by Fig. 3-a_0, so f-range is also hardly influenced by data blocks. Namely, our algorithm cannot be affected by the number of data blocks.

8 System Evaluation

In this section, we plan several experiments to evaluate the effect of energy optimization and cost of the modulo based DPA. The experiments are performed in a Hadoop MapReduce system and distributed environment.

8.1 Setup

The MapReduce system consists of 6 nodes, including a *JobTracker* (master node) and 10 *TaskTracker*s (slave nodes). Because the optimization is mainly for the slave nodes, *TaskTracker*s consist of two types of computers whose capabilities are quite different. The testbed is shown as Table 1.

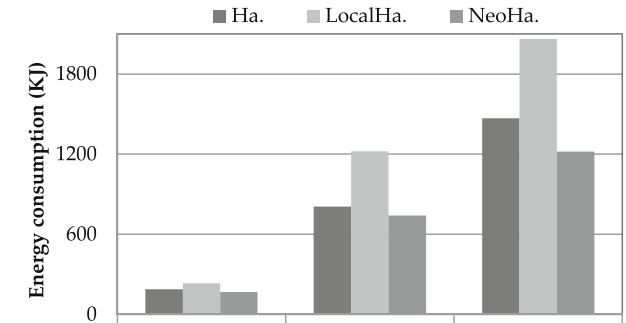
To estimate the EC optimization, three different kinds of MapReduce systems, which are explained in Table 2, are compared in the experiments. Base on the Hadoop MapReduce, we built LocalHadoop and NeoHadoop as new implementations of MapReduce. Hadoop adopts the random DPA, by which data block is placed to the nodes randomly, and a node will process the data blocks of other nodes remotely after it finishes processing its local data blocks. LocalHadoop adopts the random DPA too. It is built on the local file system, thus there is no remote data access, and tasks only process their local data. NeoHadoop, which is built on the local file system with the modulo based DPA, is an extension of LocalHadoop. With the same inputs, logic and the outputs of a job, the differences of three systems are data placement and data location. We compare the EC and time consumption of nodes in Hadoop, LocalHadoop and NeoHadoop during the execution of test case, and study the EC optimization effect of the proposed DPA.

Table 3 Measurement approaches of experimental results

Name	Unit	Measuring approach
Energy consumption	Kilo Joule	The PowerBay power-meter, whose power precision is ±0.01˜0.1W, maximum is 2200W, measuring frequency is 1.5-3 second, is used to measure EC [30].
Disk I/O	MB	Accumulative value of disk I/O per second of each node
Remote I/O	MB	Accumulative value of network I/O per second of each node, then divided by 2 because data been sent and received are equal.
CPU workload	GHz	Accumulative value of multiplying CPU frequency and CPU usage per node.

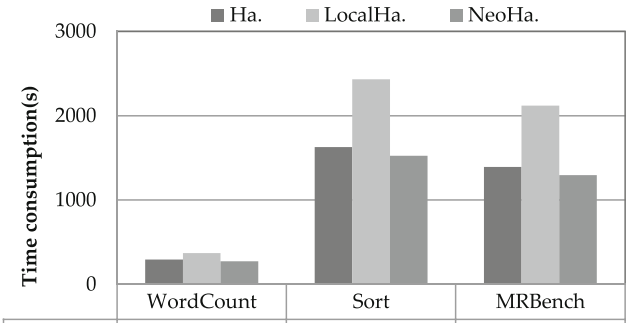
Three typical and well-known jobs, *WordCount* [27], *Sort* [28] and *MRBench* [29] are chosen as test cases. *MRBench* is treated as an interactive job. *WordCount* and*Sort* are treated as an I/O intensive and CPU intensive job respectively. Since our algorithm has no relationship with the amount of data, for simplicity, the data size of *WordCount* and *Sort* are 2GB per node, the number of concurrent job is 100 for*MRBench*. We compare and analyze both the energy consumption, time consumption and resource consumption of three different MapReduce systems. The measured value is accumulated by values of 10 *TaskTracker*s. The unit and measuring approaches are list in Table 3.

Let disk I/O, remote I/O and CPU workload of Hadoop system be one unit, respectively. Each corresponding value of LocalHadoop and NeoHadoop is normalized to dimensionless ratio.



	WordCount	Sort	MRBench
Ha.	188.1	807.34	1468.04
LocalHa.	232.87	1220.52	2063.7
NeoHa.	167.51	739.73	1218.34

(a) Energy Consumption



	WordCount	Sort	MRBench
Ha.	294	1628	1392
LocalHa.	369	2433	2121
NeoHa.	272	1525	1294

(b) Time Consumption

Fig. 5 The comparison of energy consumption and time consumption of WordCount, Sort and MRBench on three systems

8.2 Energy Analysis

In this section we evaluate the effect of EC optimization. The results are shown in Fig. 5. We conclude that NeoHadoop is more efficient than Hadoop for the *WordCount* [27], *Sort* [28] and *MRBench* [29]. EC is reduced by 10.9 %, 8.3 % and 17 % respectively, and time consumption is reduced by 7 %, 6.3 % and 7 % respectively. Optimization on EC is better than that on time consumption. What's more, LocalHadoop, even there is no remote data access, is less efficient than Hadoop in both EC and time consumption.

LocalHadoop only processes local data, and the data are placed randomly, as a consequence the higher-capability nodes wait the lower-capability nodes in a heterogeneous system. In our testbed, the difference of capability is huge, so that both waiting time and waiting EC are also huge. Hadoop avoids the waiting time between nodes by processing data remotely, but CPU also brings waiting EC when it is waiting for remote data access. The cost of remote data access is the main reason of waiting EC in Hadoop. The parallelism of NeoHadoop is ensured through a fair data placement, so that the NeoHadoop is better than LocalHadoop in avoiding waiting time between nodes, and is also better than Hadoop in avoiding remote data access.

As we know, the idle power of computer is a constant, and EC is calculated by multiplying the power and time, therefore if waiting EC is reduced, time consumption is also reduced. The optimization effect of EC is better than that of time consumption because we reduce both the waiting EC and remote data access, meanwhile the network I/O equipment (net card) works with a lower power within a shorter period, then the power of the node in NeoHadoop is less than that in Hadoop.

Besides, the effect of EC optimization on MRBench is better than that on other two cases. MRBench consists of many small jobs but the other two cases consist of one big job. Each job causes waiting EC in Hadoop and LocalHadoop. On the contrary, NeoHadoop avoids waiting EC greatly. So that our approach has better optimization effect on MRBench.

8.3 Resources Analysis

In this section, we compare the accumulative resource consumption of Hadoop, LocalHadoop and NeoHadoop under the different test cases. In order to

compare results easily, the resource consumption of Hadoop is set to one unit, and the corresponding resource consumptions of LocalHadoop and NeoHadoop are normalized proportionally. Figure 6 shows the experimental results. The axis *y* represents the accumulative resource consumption (ration), axis *x* represents different systems. Because of the relevance between algorithm and resource consumption, we can only compare resource consumption of three systems with the same task, but not those with different tasks.

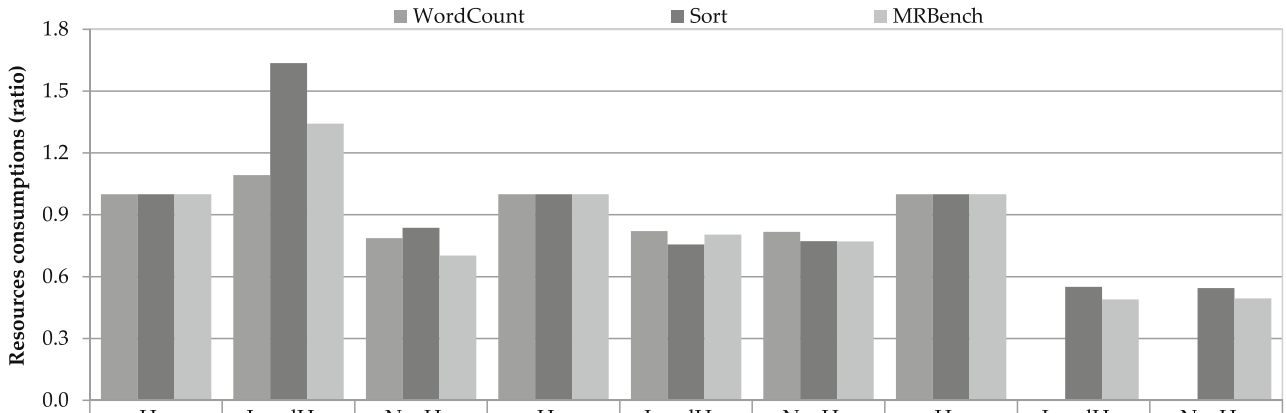
Firstly, the CPU consumption is analyzed. The CPU consumption presents the amount of waiting EC, that is, for the same algorithm and data size, the more CPU consumption is, the longer CPU waiting is (CPU usage is not zero when it is idle), and the more waiting EC is. As shown in Fig. 6, no matter which task is performed, the CPU consumption of NeoHadoop is the least, and that of LocalHadoop is the most. We can conclude that: the huge network I/O, but not the poor parallelism, is the dominating reason of CPU's idleness in Hadoop; and the situation is opposite in LocalHadoop; but both parallelism and I/O cost are optimized in NeoHadoop.

Secondly, we analyze the local I/O consumption. For each node, no matter it processes data locally or remotely, it causes local I/O consumption in one node, and if remote data is processed, additional remote I/O is consumed. Thus, local I/O consumption represents how many data have been processed (also some I/O consumed by virtual memory and intermediate result). So that local I/O consumption of three systems are almost same no matter what kind of jobs are executed. The local I/O consumption of LocalHadoop is slightly larger than that of two systems because job on Local Hadoop runs longer and extra logs consumed some local I/O operations.

Thirdly, we analyze the remote I/O consumption. Both NeoHadoop and LocalHadoop confirm that task only processes local data, so that the tiny I/O is consumed by *shuffle* operation in MapReduce and communications between *JobTracker* and *TaskTracker*. It proves that NeoHadoop with proposed data placement reduces the remote I/O cost greatly. As shown in Fig. 6, the I/O consumption of both NeoHadoop and LocalHadoop is equal with all test cases, and it is half of I/O consumption of Hadoop.

8.4 Cost Analysis

The modulo based DPA brings extra operations in the data loading process. In this experiment, we evaluate the data loading performance of three systems (Hadoop, LocalHadoop and NeoHadoop). Firstly, data loading is executed only once, thus DPA is not executed repeatedly. The data query and process benefit from the fairness of data placement. The proportional cost is very low. Secondly, during the data loading of NeoHadoop and LocalHadoop, it treats data block as



	Ha.	LocalHa.	NeoHa.	Ha.	LocalHa.	NeoHa.	Ha.	LocalHa.	NeoHa.
	CPU			Disk			Network		
WordCount	1.000	1.093	0.786	1.000	0.821	0.818	1.000	0.002	0.002
Sort	1.000	1.636	0.837	1.000	0.756	0.772	1.000	0.551	0.544
MRBench	1.000	1.343	0.702	1.000	0.803	0.770	1.000	0.489	0.494

Fig. 6 The comparison of accumulative resource consumption (normalized by Hadoop) of three systems with WordCount, Sort and MRBench

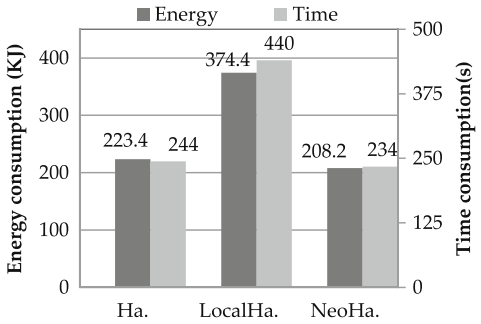


Fig. 7 The loading time and EC of three different MapReduce systems

the smallest unit. Calculating the destination of a data block is much faster than transferring the data block through network. If the data loading is performed by two threads, one of which takes the responsibility of placing data and the other takes the responsibility of loading data, then the former thread never blocks the latter one. Therefore, the data loading performance is mainly dominated by I/O performance. The cost of modulo based DPA is ignorable. We compare both EC and time consumption of loading same data set in three systems, and show the results in Fig. 7.

The loading performance of the three systems are close. Thus, the modulo based DPA does not bring much cost to data loading. What's more, loading capability of NeoHadoop is slightly better than that of Hadoop because nodes with higher I/O capabilities get more data blocks, and the more data blocks these nodes receive, the faster the data load.

9 Conclusions and Future Works

In this paper, we propose a modulo based Data Placement Algorithm (DPA) for optimizing Energy Consumption (EC for short) of MapReduce system. Firstly, we propose a novel idea for optimizing EC of a MapReduce system, based on which we define the objectives of EC optimization. We also demonstrate the waiting EC of MapReduce system and its optimization approach. Then, we propose the data placement goals for optimizing EC and the modulo based DPA to implement these goals. Finally, the correctness of proposed DPA is proved from both theoretical and experimental perspectives. It satisfies the goals of "Fairness of size (f-size)", "Fairness of Range (f-range)" and "Best Adaptability (b-adapt)"; It optimizes EC of MapReduce system efficiently; eventually; It does not bring the additional cost to the data loading.

The EC optimization approach we proposed in this paper can be well applied to the most of MapReduce based Big Data analysis systems. In short, with the modulo of DPA, the EC for the *WordCount*, *Sort* and *MRBench* can be reduced by 10.9 %, 8.3 % and 17 % respectively, and time consumption is reduced by 7 %, 6.3 % and 7 % respectively. In MapReduce, the node failure happens in some cases. Then the cost for data transmission is unimaginable. If the data on failed node have the replica on the normal node, then data transmission can be reduced a lot. However, the current DPA just thinks of the data in a specified job without replica. The future works also need to take data replication mechanisms into consideration. For example, the replica should be distributed to the node in another rack where adopts different DPA. Data replication mechanisms also needs the support of adaptability, however, in our algorithm, adaptability is not enough which cannot remove an arbitrary node. So we will also make some improvements on DPA which will make DPA be more adaptive.

Acknowledgments Supported by the National Natural Science Foundation of China under Grant No. 61202088 and 61433008; the Fundamental Research Funds for the Central Universities under Grant No. N120817001 and N130417001; the Science Foundation of China Postdoctor under Grant No. 2013M540232; the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20120042110028.

References

1. Labrinidis, A., Jagadish, H.V.: Challenges and opportunities with big data. PVLDB **5**(12), 2032–2033 (2012)
2. Dean, J., Ghemawat, S.: Mapreduce: a flexible data processing tool. Commun. ACM (CACM) **53**(1), 72–77 (2010)
3. Song, J., Liu, X., Zhu, Z., Zhao, D., Yu, G.: A Novel Task Scheduling Approach for Reducing Energy Consumption of MapReduce Cluster. IETE Tech. Rev. **31**(1), 65–74 (2014)
4. Elnozahy, E.N., Kistler, M., Rajamony, R.: Energy-efficient server clusters. PACS 2002, 179–196
5. Lee, K.G., Bharadwaj, V., Sivakumar, V.: Design of fast and efficient Energy-Aware Gradient-Based scheduling algorithms heterogeneous embedded multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. (TPDS) **20**(1), 1–12 (2009)
6. Da Costa, G., Dias de Assunção, M., Gelas, J.-P., Georgiou, Y., Lefèvre, L., Orgerie, A., Pierson, J.-M., Olivier, R.,

Sayah, A.: Multi-facet approach to reduce energy consumption in clouds and grids: the GREEN-NET framework. e-Energy 2010, 95–104
7. Lang, W., Patel, J.M.: Energy management for MapReduce clusters. PVLDB **3**(1), 129–139 (2010)
8. Maheshwari, N., Nanduri, R., Varma, V.: Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework. Future Generation Comp. Syst. (FGCS) **28**(1), 119–127 (2012)
9. Xiong, W., Kansal, A.: Energy efficient data intensive distributed computing. IEEE Data Eng. Bull. (DEBU) **34**(1), 24–33 (2011)
10. Palanisamy, B., Singh, A., Liu, L., Jain, B.: Purlieus: locality-aware resource allocation for MapReduce in a cloud. In: SC, pp. 58:1–58:11 (2011)
11. Pinheiro, E., Bianchini, R., Enrique, V.C., Heath, T.: Load balancing and unbalancing for power and performance in cluster-based systems. In: Workshop on compilers and operating systems for low power, pp. 182–195 (2001)
12. Chen, Y., Keys, L., Katz, R.H.: Towards Energy Efficient MapReduce. Technical Report of EECS Department University of California, Berkeley. Available via http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-109.pdf (2008)
13. Pinheiro, E., Bianchini, R., Carrera, E.V., Heath, T.: Dynamic cluster reconfiguration for power and performance: Compilers and operating systems for low power (book), Kluwer Academic Publishers Norwell, ISBN:1-4020-7573-1, 75–93 (2003)
14. Chen, Y., Alspaugh, S., Borthakur, D., Katz, R.: Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis. In: 7th ACM Europan Conference on Computer System, pp. 43–56 (2012)
15. Yigitbasi, N., Datta, K., Jain, N., Willke, T.: Energy efficient scheduling of MapReduce workloads on heterogeneous clusters. In: Green Computing Middleware (ACM), pp. 1–6 (2011)
16. Pinheiro, E., Bianchini, R.: Energy conservation techniques for disk array-based servers. ICS 2004, 68–78
17. Kaushik, R.T., Abdelzaher, T.F., Egashira, R., Nahrstedt, K.: Predictive data and energy management in GreenHDFS. IGCC 2011, 1–9
18. Colarelli, D., Grunwald, D.: Massive arrays of idle disks for storage archives. SC 2002, 1–11
19. Karger, D.R., Lehman, E., Leighton, F.T., Panigrahy, R., Levine, M.S., Lewin, D.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. STOC 1997, 654–663
20. Brinkmann, A., Salzwedel, K., Scheideler, C.: Efficient, distributed data placement strategies for storage area networks (extended abstract). SPAA 2000, 119–128
21. Tao, C., Nong, X., Fang, L., et al.: Clustering-Based And consistent Hashing-Aware data placement algorithm. J. Softw. **21**(12), 3175–3185 (2010). (in Chinese)
22. Yuan, D., Yang, Y., Liu, X., Chen, J.: A data placement strategy in scientific cloud workflows. Future Generation Comp. Syst. (FGCS) **26**(8), 1200–1214 (2010)
23. Profiling Energy Usage for Efficient Consumption. https://msdn.microsoft.com/en-us/library/dd393312.aspx. 2016/6/1
24. Unixbench https://code.google.com/p/byte-unixbench/
25. Liu, Z.: Efficient, balanced data placement algorithm in scalable storage clusters. Journal of Communication and Computer, 2007, (7):8-17
26. Ronald, L.: Graham: Bounds on Multiprocessing Timing anoMalies. SIAM J. Appl. Math. (SIAMAM) **17**(2), 416–429 (1969)
27. WordCount program: Available in Hadoop source distribution: https://src/examples/org/apache/hadoop/examples/WordCount
28. Sort program: Available in Hadoop source distribution: https://src/examples/org/apache/hadoop/examples/sort
29. MRBench program: Available in Hadoop source distribution: https://src/examples/org/apache/hadoop/mapred/MRBench
30. Jie, S., Li, T., Zhi, W., Zhiliang, Z.: Study on energy-consumption regularities of cloud computing systems by a novel evaluation model. Computing, 1–19 (2013)
31. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM (CACM) **51**(1), 107–113 (2008)