2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress

# Optimizing MapReduce Partitioner Using Naive Bayes Classifier

Lei Chen[1,2], Wei Lu[1]✉, Liqiang Wang[2], Ergude Bao[1], Weiwei Xing[1], Yong Yang[1,2], Victor Yuan[3]
[1] School of Software Engineering,Beijing Jiaotong University, Beijing, China
[2] Department of Computer Science, University of Central Florida, Orlando, USA
[3] Department of Electrical Engineering, Southern Methodist University, Dallas, USA
Email: [1]{13112084,luwei,ergd,wwxing,12112088}@bjtu.edu.cn
[2]{lwang}@cs.ucf.edu [3]{fvyuang}@smu.edu

***Abstract*—Data locality and data skew on the reduce side are two essential issues in MapReduce. Improving data locality can decrease network traffic by moving reduce tasks to the nodes where the input data of reduce tasks is located. Data skew will lead to load imbalance among reducer nodes. Partitioning is an important operation of MapReduce because it determines the destinations of map output and could significantly affect the data amount of shuffle. Therefore, an effective partitioner can improve MapReduce performance by increasing data locality and decreasing data skew on the reduce side. Previous studies considering the two essential issues have ignored the fact that for different types of jobs, the priority of data locality and data skew on the reduce side may produce different effects on the execution time. In this paper, we propose a novel partitioner based on naive Bayes classifier, namely, BAPM, which achieves better performance through optimizing data locality and data skew by leveraging the naive Bayes classifier, i.e., considering job type and bandwidth as classification attributes. Our experiments are performed in a Hadoop cluster with 31 nodes and the results show that BAPM speeds up the computing performance of MapReduce by up to 19.26% compared to the native Hadoop.**

## I. Introduction

As a popular framework for processing big data, MapReduce [1] consists of two main stages: map stage, which transforms input data into intermediate data, namely <key,value> pairs, and reduce stage, which is applied to each list of values with the same key. As a critical feature, partitioning determines what reducer an intermediate data item will be sent to. Therefore, an ineffective partitioner will decrease data locality or cause data skew, consequently degrade system performance.

Data locality [2] in a distributed environment refers to that computing and data are preferably co-located on the same node in order to reduce network traffic. The current schedulers in native Hadoop [3] only consider data locality in map tasks and ignore data locality of reduce tasks. When a user uploads files to HDFS[4], the files are divided into blocks (64 MB by default), and each chunk is replicated across multiple machines. Data processing is co-located with data storage, i.e., when a file needs to be processed, the job scheduler consults a storage metadata service and then schedules a map process on that node so that data locality in the map phase is exploited efficiently. In state-of-the-art MapReduce systems, each map task processes one split of input data and generates a sequence of key-value pairs, which are referred to intermediate data, on which hash partitioning is performed. The key-value pairs with the same hash result, which we define as a hash partition, are assigned to the same reduce task. In the reduce stage, a reducer takes a partition as input and performs reduce operation on the partition. However, with hash function, there is a possibility of transferring a large amount of intermediate results to certain reducer nodes, which could cause massive network communication, furthermore, data locality might not be achieved and job execution time might be prolonged.

Data skew refers to the imbalance in the amount of data assigned to each task or the imbalance in the amount of work required to process the data [5]. The essential reason for data skew is that data sets in the real world are often skewed and we do not know data distribution beforehand, which causes the aforementioned hash function in native Hadoop to be inefficient in most cases. Therefore, balancing hash partition size, i.e., the size of the key-value pairs with the same hash result, is important for maintaining load balancing among the reducers.

In recent years, several approaches, such as [12][20], have been proposed to improve the performance of MapReduce by increasing data locality and decreasing data skew. However, these studies do not consider job type and network bandwidth when adjusting data locality or data skew. Based on the amount of intermediate data transmitted from Mappers to Reducers in shuffle phases, all MapReduce jobs can be divided into three different types: reduce-input-heavy, reduce-input-light, reduce-input-zero. We will describe this in details in Section IV. When the bandwidth is limited, for reduce-input-heavy jobs, data transmission will result in more overhead. Therefore, data locality is one of major factor that affecting job execution time. In contrast, when the bandwidth among nodes is relatively high, for reduce-input-light jobs, the data transmission cost will be less important. In this case, data skew, which could cause load imbalance among reducers, will be a major factor affecting the execution time. This has been verified in our experimens. Therefore, the preference for data locality or data skew may result in varying execution time when running different types of jobs at different bandwidths.

In this paper, we propose a novel partitioner for MapReduce, namely, BAPM, which improves the job execution time by properly selecting the preference of data locality or data skew in consideration of job type and bandwidth. Our contributions can be summarized as follows:

1. BAPM makes a proper choice between two algorithms, LRS (data **L**ocality **R**ather than **S**kew) and SRL (data **S**kew **R**ather than data **L**ocality), both consider data locality and data skew on reduce side, but with reversed preference. BAPM leverages naive Bayes classifier by considering job type and network bandwidth as classification attributes.

2. According to the amount of intermediate data transmitted from mappers to reducers, we classify a given MapReduce job into specific categories, which is then used by BAPM to determine appropriate algorithms (LRS or SRL) for the given job.

We evaluate BAPM in YARN(Hadoop 2.6.0) with bench of benchmarks. Based on our training sets, the selection accuracy of BAPM is 93.18%. Comparing with the native Hadoop, the improvement caused by BAPM reaches 19.26% on the average.

The rest of this paper is organized as follows. Section II reviews some related studies. Section III introduces the LRS and SRL algorithms. Section IV describes our BAPM in details. Section V describes the performance evaluation of BAPM. Finally, Section VI concludes the paper.

## II. Related Work

There are many approaches to improve data locality of Map tasks. Tan et al. [6] designed a resource-aware scheduler for Hadoop to mitigate the job starvation problem and improve overall data locality; it utilizes wait scheduling and random peeking scheduling for map tasks in order to optimize task placement.

Many studies have focused on the locality of reduce tasks. LARTS [7] attempts to collocate reduce tasks with the maximum required data computed after recognizing input data network locations and sizes. It adopts a cooperative paradigm that seeks higher data locality while circumventing scheduling delay, scheduling skew, poor system utilization, and low degree of parallelism.

MapReduce implementation in Hadoop 2.6.0 overlooks data skew, which obstructs scale-up in parallel computing systems. LIBRA [8] is a system that implements a set of innovative skew mitigation strategies. LIBRA can handle data skew not only on map side but also on reduce side. However, LIBRA can not tackle the problem of prolonging execution time caused by lower degree of data locality.

Many studies have comprehensively investigated data locality on map side as well as data skew. Hsu et al. [9] proposed a method for improving MapReduce execution in heterogeneous environments. The method achieves higher performance by dynamically partitioning data before the map phase in order to improve locality of map tasks, and it uses virtual machine mapping in the reduce phase in order to balance workload among reduce nodes.

Some studies [10][11][12][15][16] have comprehensively investigated data locality for reduce-side as well as data skew. However, all the aforementioned approaches have ignored the fact that for different types of MapReduce jobs, different consideration orders of data locality and data skew could affect MapReduce execution time under varying network bandwidth. Our approach, i.e., BAPM, can solve this problem and improve performance.

## III. LRS and SRL

To improve the performance of MapReduce jobs, we design two algorithms named LRS and SRL, which increase data locality and decrease data skew on reduce side, but with reversed preference. LRS is an extension of the LEEN algorithm in [10], and SRL is an extension of the CLP algorithm in [11]. In this section, we describe these two algorithms and then demonstrate their data locality and data skew characteristics through case studies. The variable notation used in this paper are shown in Table I.

TABLE I
Variable Notation

| Variable Name | Description |
|---|---|
| $K : \{key_0, ..., key_{m-1}\}$ | keys in $< key, value >$ pairs |
| $N : \{node_0, ..., node_{n-1}\}$ | the datanodes in the entire hadoop cluster |
| $C : \{C_0, C_1, ..., C_{n-1}\}$ | a set of clusters, each of which contains several $< key, value >$ pairs |
| $FK_i^j$ | the frequency of $key_i$ in the data node $node_j$ |
| $FK_i$ | total frequency of $key_i$ |
| $DataSkew_i^j$ | the data skew rate of all reduce nodes if data with $key_i$ are partitioned to $node_j$ |
| $hostedDataN_i^k$ | the amount of data hosted in $node_k$ if data with $key_i$ are partitioned |
| $Tran_i^j$ | the amount of data transferred to $node_j$ with $key_i$ |
| $TotalTran_i^j$ | the amount of data transmitted to $node_j$ with the keys in cluster $C_i$ |

### A. LRS

As shown in Algorithm 1, LRS considers data **L**ocality **R**ather prior to data **S**kew. LRS produces data files and a metadata file. The number of data files is the same as the number of keys. The metadata file contains a frequency table, which includes the number of records in each file and represents the key frequency. When all map tasks are done, all metadata files will be aggregated. Then, all keys are partitioned into different DataNodes according to the LRS algorithm, which works as follows:

1. Suppose that there are *m* key-value pairs after the map phase and *n* reducers in the cluster. Because the higher the frequency of a key, the greater it effects on data locality, we prioritize these high frequency keys. Line 1 sorts all keys in descending order according to their $FK_i$.

2. For a specific $key_i$(0≤i≤m-1), in order to achieve the best locality, LRS selects the node with the maximum frequency of $key_i$. Line 3 sorts all nodes in descending order according to the frequency of $key_i$ on every $node_j$ (0≤j≤n-1).

**Algorithm 1** LRS

```
Input:  K, N
Output:  partition(K,N);
 1: Sort keys in descending order based on $FK_i$
 2: for i=0 to m-1 do
 3:     Sort all nodes in descending order based on $FK_i^j$ (0 ≤ j ≤ n-1);
 4:     j=0;
 5:     while $DataSkew_i^j$ >$DataSkew_i^{j+1}$ do
 6:         j = j + 1;
 7:     end while
 8:     Partition($k_i$,$N_j$); //move all $< key_i, value >$ pairs to Reducer j
 9:     for k=0 to n-1 do
10:         Calculate $hostedDataN_i^k$;
11:     end for
12: end for
```

3. Lines 5-7 compare the current node with the node with the second-hightest frequency of $key_i$. To solve load imbalance among reduce nodes caused by data skew, LRS introduces $DataSkew_i^j$ to describe data skew rate of all reduce nodes if the data with $key_i$ are partitioned to $node_j$. $DataSkew_i^j$ is defined as Formula (1), where $Mean$ represents the mean of all $HostedDataN_i^j$ values. A lower value is preferred. LRS recursively looks for nodes with lower $DataSkew_i^j$ value.

$$DataSkew_i^j = \sqrt{\frac{\sum_{j=0}^{n-1}(hostedDataN_i^j - Mean)^2}{N}} \qquad (1)$$

4. After the node with the lower $DataSkew_i^j$ is determined, lines 8-11 move all <Key$_i$, value> pairs to the selected $node_j$ and calculate the new values of the hosted data on different DataNodes.

5. Then, LRS continues to process the rest of the keys with the same strategy.

LRS optimizes the existing algorithm LEEN [10], the improvements include: (1) LEEN assumes the amount of output data on each map node are equal, this is not always true. Our LRS obtains the actual intermediate data on each map node by the Data Amount Monitor, which will be described in Section IV.A. (2) LEEN sorts the keys in descending order accroding to their FLK, which is defined as formula (2), where $mean'$ represents the mean of $FK_i^j$ values. This sorting provides a tradeoff between data skew rate of all reducers (the numerator in Eq. (2)) and data locality (the denominator in Eq. (2)). However, the numerator considers the data skew rate of all keys rather than all reducers. In addition, for a specific key $i$, finding the maximum of $FK_i^j$ needs traversing all nodes, therefore, the computational complexity of the sorting is m×n, which is more time-consuming than our LRS's sort in step 1. To reduce time complexity, sort in this paper is done on multiple nodes concurrently.

$$FLK_i = \frac{\sqrt{\frac{\sum_{j=1}^{n}(FK_i^j - mean')^2}{N}}}{max_{1\le j\le n}FK_i^j} \qquad (2)$$

## B. SRL

As shown in Algorithm 2, SRL considers data **S**kew **R**ather prior to data **L**ocality on the reduce side. SRL creates a cluster set C, which contains the same number of clusters as the number of reducers. The data with the same key will be put into the same data cluster, and each data cluster is sent to one reducer. The SRL consists of three main phases:

**Algorithm 2** SRL

```
Input:  K, N, C
Output:  partition(K,N);
 1: Obtain input data distribution using random sampling;
 2: Sort the keys in descending order based on their $FK_i$ ($0 \le i \le m-1$);
 3: i=0;
 4: while i ≤ m-1 do
 5:     $key_i \to C_{min}$; //put the $< key_i, value >$ pairs into the data cluster $C_{min}$
 6:     Update the amount of data in cluster $C_j$ ($C_j$ ($0 \le j \le n-1$));
 7:     i = i + 1;
 8: end while
 9: Sort C in descending order based on their data volume
10: i = 0; //cluster number
11: while i ≤ n-1 do
12:     for j=0 to n-1 do //node number
13:         for every $key_p$ in $C_i$ do
14:             $Tran_p^j$ = $FK_p$ - $FK_p^j$;
15:             $TotalTran_i^j$ += $Tran_p^j$;
16:         end for
17:     end for
18:     partition($C_i$, $node_j$) with the smallest $TotalTran_i^j$;
19:     Remove $N_j$;
20:     i = i + 1;
21: end while
```

1.In first phase (line 1), SRL uses the random sampling method to analyze input data and uses an extra MapReduce job to gather information about data distribution.

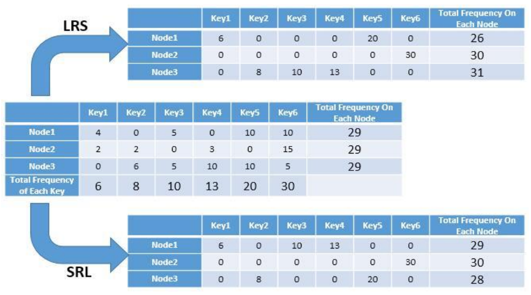
2.In the second phase (line 2-7), in order to balance the load of each reducer when processing skewed data, SRL provides a heuristic partition method to combine <key,value> pairs into data n clusters so that every data cluster has similar data size. $C_{min}$ in line 5 represents the cluster $C_j$ ($0 \le j \le n-1$) containing the smallest amount of data in cluster set C.

3.The final phase (lines 9-21) assigns data cluster to a suitable reducers considering data locality. With the same reason described in LRS, line 9 sorts the cluster set C in descending order based on the data amount of the clusters. We improve data locality on reduce side by decreasing data transfer in shuffle phase. Lines 13-16 calculate total data transfer when partitioning <key, value> pairs in cluster $C_i$ to $node_j$. Line 18 selects the node to which the data traffic with the keys in cluster $C_i$ in shuffle is minimal. Line 19 removes the Reducer node selected from the Reducer set.

SRL optimizes the existing algorithm CLP [11], both algorithms have same three phases, the difference between them is in Phase-3, during which the data locality on reduce side is improved. The drawbacks of CLP is it considers load balance repeatedly in Phase-2 and Phase-3, which causes unnecessary overhead. Our SRL improves performance of MapReduce jobs by considering the load balance and data skew in Phase-2 and Phase-3 respectively.

## C. Comparison of LRS and SRL

We compare different partitioning results on the same intermediate data by LRS and SRL, as shown in Figure 1.



|  | Key1 | Key2 | Key3 | Key4 | Key5 | Key6 | Total Frequency On Each Node |
|---|---|---|---|---|---|---|---|
| Node1 | 4 | 0 | 5 | 0 | 10 | 10 | 29 |
| Node2 | 2 | 2 | 0 | 3 | 0 | 15 | 29 |
| Node3 | 0 | 6 | 5 | 10 | 10 | 5 | 29 |
| Total Frequency of Each Key | 6 | 8 | 10 | 13 | 20 | 30 |  |

LRS

|  | Key1 | Key2 | Key3 | Key4 | Key5 | Key6 | Total Frequency On Each Node |
|---|---|---|---|---|---|---|---|
| Node1 | 6 | 0 | 0 | 0 | 20 | 0 | 26 |
| Node2 | 0 | 0 | 0 | 0 | 0 | 30 | 30 |
| Node3 | 0 | 8 | 10 | 13 | 0 | 0 | 31 |

SRL

|  | Key1 | Key2 | Key3 | Key4 | Key5 | Key6 | Total Frequency On Each Node |
|---|---|---|---|---|---|---|---|
| Node1 | 6 | 0 | 10 | 13 | 0 | 0 | 29 |
| Node2 | 0 | 0 | 0 | 0 | 0 | 30 | 30 |
| Node3 | 0 | 8 | 0 | 0 | 20 | 0 | 28 |

Fig. 1. Example of LRS and SRL

The case study considers three DataNodes and six keys; every DataNode is configured as a mapper node and a reducer node. The numerical values in Figure 1 represent the frequency of each key. Here, we assume that all records are of equal size.

We demonstrate inconsistency in key distribution, which affects data locality on the reduce side. As shown in Figure 1, the key distribution is inconsistent among nodes. The data transmission amounts will be noticeably different under various partitioning strategies. In this case study, the data transmission includes 37 records when running LRS and 47 records when running SRL; the former is smaller than the latter by 21.28%.

The case study also shows the variation in the intermediate key frequencies, which could cause load imbalance among the reducers. The total key frequency on each node is 29, but the frequency of each key varies (6, 8, 10, 13, 20, and 30). When hash partitioning varies, the distribution of reducer inputs will be different. The Fairness of the Reducer input is 2.16 when running LRS and 0.82 when running SRL.

In this study, a larger data transmission leads to longer execution time. Lower Fairness results in better performance. For LRS, the improvement in data locality is greater than the alleviation of data skew. In contrast, for SRL, the alleviation of data skew is greater than the improvement in data locality.

## IV. BAPM

In this section, we present BAPM, a new partitioner for Mapreduce. First, we describe BAPM in detail. Next, because BAPM considers job type and network bandwidth as classification attributes for the naive Bayes classifier, we discuss how to classify MapReduce jobs. Finally, we describe the naive Bayes classifier used in BAPM.

### A. *BAPM in YARN*

The architecture of BAPM is shown in Figure 2. In particular, each Data Amount Monitor records the input data amount and output data amount of each map task. Each Data Frequency Table (DFT) creates a table that records the value of each key in every DataNode after the map phase. The global DFT (GDFT) summarizes all DFT data in each DataNode. The Job Type Classifier classifies jobs into several categories

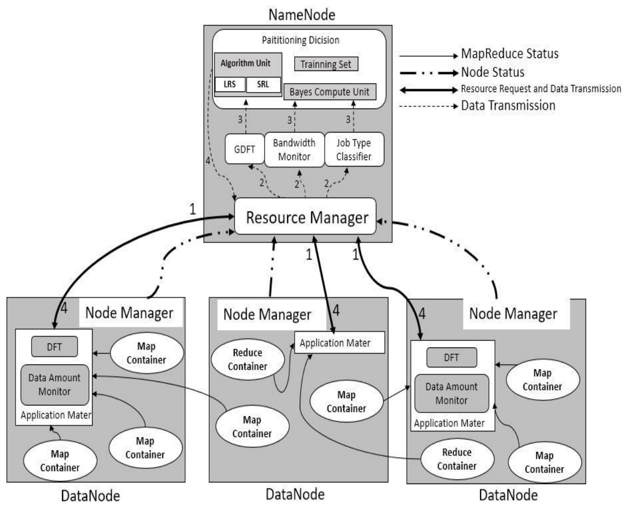


Fig. 2. The Architecture of BAPM

(IV.B). The Bandwidth Monitor obtains real-time bandwidth in the cluster (IV.C). The Partitioning Decision determines the partition algorithm (LRS or SRL)(IV.D). The workflow of BAPM consists of 4 steps:

1. In BAPM, job type could be given by users or classified automatically. The Data Amount Monitor computes the amount of input and output of each Map task, which is essential for job type classification. DFT counts and records intermediate <key,value> pairs generated by map functions in every DataNode; the number of pairs with the same key is denoted as a key frequency. After all map tasks are completed, the profiling data, which consists of job type that can be input by users or the statistical data from the Data Amount Monitor, and the key frequencies in DFT, will be transmitted from the Application Master to the Resource Manager through heartbeat messages.

2. When the Resource Manager receives profiling data, it transmits them to the GDFT and Job Type Classifier. Then, the GDFT summarizes all key frequencies in each DataNode into a table, which will be used in LRS and SRL later. The format of GDFT is similar to the tables in Figure 1. Using the rules described in IV.B, the Job Type classifier determines job type by analyzing the data from the Data Amount Monitor.

3. To select partitioning algorithms (LRS or SRL), the Partitioning Decision uses three modules: Bayes Compute Unit, Training Set, and Algorithm Unit. After the Partitioning Decision receives profiling data from the Bandwidth Monitor and the Job Type classifier, based on current bandwidth and job type as categorical attributes, a naive Bayes classfier chooses LRS or SRL, which is detailed in IV.D. The Training Unit records the profiling information of all MapReduce jobs that have been run before, including job type, bandwidth, and the partitioning algorithm adopted. In order to speed up probability calculation, in the Training Set, we record statistical

TABLE II
JOB TYPE

| Type | Job |
|---|---|
| Reduce-input-light | Numerical Summerization, Top N, Distinct Counting |
| Reduce-input-heavy | Inverted Indexes, Structured to Hierarchical, Partitioning, Sort(containing TeraSort), Shuffling, Join |
| Reduce-input-zero | Filting, Bining, Cartesian Product, Permutation |

information such as the number of invocation times of LRS, the number of times of input-reduce-heavy job types under LRS. We also add current job information to the Training Set and update the statistics after the partitioning algorithm is determined; this will enable BAPM to further increase the selection accuracy.

4. When the partitioning strategy is determined, the Algorithm Unit will generate partitioning results using the table in GDFT, and the result will be transmitted back through resource response messages from the Resource Manager to the Application Master.

## B. Job Type

If mappers in a job produce a large number of <key,value> pairs, its execution time is mainly determined by network conditions in a cluster; therefore, we can categorize jobs based on the amount of intermediate data transmitted from mappers to reducers. If the amount of map output is approximately equal to or greater than the input data for a job, we refer to this job as a reduce-input-heavy job. In contrast, if the amount of transmitted data is obviously less than the mapper input, we refer to this job as a reduce-input-light job. In particular, there is a certain type of jobs that do not contain the reduce phase; we refer to them as reduce-input-zero jobs. We classify popular MapReduce jobs using this rule and the results are listed in Table II. In this study, WordCount and Sort in our experiment represent reduce-input-light job and reduce-input-heavy job, respectively.

As the combiner can always be utilized in WordCount, the intermediate result is much smaller than the mapper input data; therefore, WordCount is classified into the reduce-input-light type.

TeraSort uses map/reduce to sort the data into a total order. TeraSort is a standard map/reduce sort, except for a custom partitioner that uses a sorted list of N-1 sampled keys that define the key range for each reduce. In particular, all keys such that sample[i-1] <= key < sample[i] are sent to reducer i. This guarantees that the output of reduce i are less than the output of reduce i+1. To speed up the partitioning, the partitioner builds a two-level trie tree that quickly indexes into a list of sample keys based on the first two bytes of the key. Map tasks mark the number of reducer that input data should be transmitted to. All Reducers do local sort in parallel and output the final total order results. Because all marked data will be transmitted over the network in shuffle phases, TeraSort is a reduce-input-heavy job.

## C. Bandwidth Monitor

Bandwidth is an important issue in networks [17][19][21]. The Bandwidth Monitor in BAPM uses iPerf3, a tool for measuring the maximum achievable bandwidth on IP networks. The bandwidth between working nodes in Hadoop is relatively stable when the running jobs on the nodes remain stable, but variable and unpredictable when the jobs change, for example, a running job is suspended or a new job begins to run. However, this variation is not mutable for the following reasons. First, the Resource Manager in Hadoop continuously monitors all resources including bandwidth in the entire cluster; it tries to prevent bandwidth mutation from happening as much as possible. For example, when users start the Balancer in Hadoop, the Resource Manager will limit the bandwidth allocated to the Balancer in order to avoid obvious performance degradation [13][20][18]. Secondly, when the Resource Manager assigns a resource container to a task, it will select the node with sufficient resources (including bandwidth) among all nodes that hold the input data; this will also prevent bandwidth mutation [14]. Therefore, the bandwidth is stable in shuffle phases of a MapReduce job. This also makes it possible to improve the performance of a MapReduce job by increasing data locality and mitigating the data skewness under special bandwidth.

## D. Bayes Compute Unit

Given a problem instance to be classified, denoted by a vector X={$x_1$,$x_2$,...,$x_m$} with $m$ features, we assume that all features are independent, C={$y_1$,$y_2$,...$y_n$} denotes a category set with $n$ categories. The naive Bayes classifier determines the category $y_i$(1⩽i⩽n) to which the instance X will be assigned. In other words, it will find the maximum among the conditional probability set {P($y_1$|X), P($y_2$|X),...P($y_n$|X)}. Under the independence assumptions, using Bayes' theorem, the conditional probability P($y_i$|X)(1⩽i⩽n) can be decomposed as

$$P(y_i|X) = \frac{\prod_{j=1}^{m} P(x_j|y_i)P(y_i)}{P(X)} \tag{3}$$

The workflow of the Bayes Compute Unit is shown in Figure 3. In the preparation stage, we determine the features and their values. There are two features in BAPM, job type and bandwidth. In vector X, $x_1$ denotes the job tupe, $x_2$ denotes the bandwidth. Obviously, the two features are independent. In the category set C, $y_1$ denotes LRS and $y_2$ denotes SRL. In training stage, firstly, BAPM executes a MapReduce job, with two algorithms (LRS and SRL) on a training data set under a special bandwidth. The algorithm with shorter execution

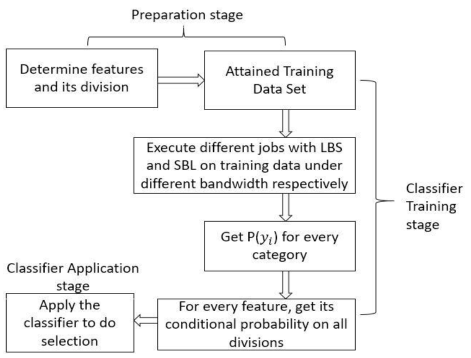

Preparation stage
Determine features and its division
Attained Training Data Set
Execute different jobs with LBS and SBL on training data under different bandwidth respectively
Get P($y_i$) for every category
For every feature, get its conditional probability on all divisions
Classifier Training stage
Classifier Application stage
Apply the classifier to do selection

Fig. 3. The Workflow of Bayes Classifier

**Algorithm 3** BAPM

```
 1: $V_{init}$ = the volume of the input data before Map phase;
 2: $V_{inter}$ = the volume of the intermediate data after Map phase;
 3: **if** $V_{inter}$ <0.8$V_{init}$ **then**
 4:     Job_Type = 0; //reduce_input_heavy job
 5: **else**
 6:     Job_Type = 1; //reduce_input_light job
 7: **end if**
 8: bw = the bandwidth values got from the Bandwidth Monitor;
 9: trainResult = the statistics of training data set;
10: partitioner = Bayes(JobType, BW, TrainResult);
11: **if** partitioner = 0 **then**
12:     SRL;
13: **else**
14:     LRS;
15: **end if**
```

time will be chosen as the designated algorithm. BAPM will repeat the procedure under different bandwidths. Finally, BAPM obtains P($y_i$) for every category, and the conditional probability of every feature in all divisions.

When executing a MapReduce job that has been explored, BAPM uses Algorithm 3 to choose the proper algorithm from LRS and SRL to optimize the performance of the job. In Algorithm 3, lines 1-7 determine the job type and line 8 measures the bandwidth. Line 9 obtains the statistics of training data set. Lines 10-15 use the Naive Bayes classifier to perform the classification.

## V. Evaluation

### A. Experiment Environment

All experiments are performed on a homogeneous Hadoop cluster running Hadoop 2.6.0. The cluster consists of 7 identical servers, each equipped with 12 x86 64 cores Xeon processors and 128 GB of RAM, running RHEL5 with kernel 2.6.22. The servers are interconnected by an Ethernet switch with 1 Gbps. We evaluate BAPM performance in a virtual cluster comprising 31 virtual machines (VMs). A VM is deployed on one server to act as the master node (Namenode). We also deploy five VMs on each of the other six servers; hence, the cluster size is 30 nodes (DataNodes), the maximum number of tasks each DataNode is set to 6. All virtual machines are configured with 2 virtual CPU cores and 1 GB memory. We configure the HDFS chunk size to be 64 MB. Because the

TABLE III
Data Set of Experiments

| Data Set | Training Data Set | Testing Data Set |
|---|---|---|
| Size | 12G | 12G |
| Key Frequencies Variation | 105% | 187% |
| Keys Distribution | 126% | 201% |

bandwidth is a critical attribute of the naive Bayes classifier, we vary the bandwidth (from 100 Mbps to 1 Gbps at intervals of 100 Mbps) in our cluster with OpenFlow.

We generated training data sets, which embodies the variation in key frequencies and inconsistency in key distribution comprehensively. To control the two issues, we modify the existing methods so that files can be assigned to specific map nodes because the native Hadoop randomly selects destination nodes among available map nodes when writing files from the local disk to HDFS. We use up to 100 different keys of the same length (64 B) to avoid variation in value size, and obtain intermediate data from map function on each map node as our testing set, as shown in Table III. We ran WordCount and Sort, which represent reduce-input-light and reduce-input-heavy job, respectively, under varying bandwidths (from 100 Mbps to 1000 Mbps at intervals of 100 Mbps interval) using LRS and SRL.

### B. Evaluation of BAPM

We conduct experiments under different bandwidths using BAPM to execute WordCount and TeraSort with Testing Data Set 1 shown in Table III. In order to ensure accuracy, we perform each group of experiments at least 10 times and take the mean value as the final result to reduce the influence of the variable environment. The definitions of data locality and data skew rate have been introduced in Section III.

In order to investigate the accuracy of our BAPM, we perform two groups of experiments on WordCount and TeraSort with the Training Data Set, which shows variation of key frequencies and inconsistency in key distribution comprehensively.

We first introduce the method used to evaluate whether a selection is accurate. When BAPM completes a MapReduce job, it records the execution time and the selected algorithm (LRS or SRL). To verify if the selection is accurate, we execute the same job with the other algorithm and compare their execution time. We vary the bandwidth from 100 Mbps to 1000 Mbps with 100 Mbps as the intervals. Without the loss of generality, we only show the results in 100 Mbps and 900 Mbps in Table III. The selection accuracy of our BAPM are 92.04% and 93.18% when running WordCount and TeraSort with Dataset1, respectively. We also evaluate the improvement in execution time due to BAPM. The bandwidth under which both LRS and SRL have the equal execution time is referred to as **cross point** in this paper. When the bandwidth is more than the cross point, compared with LRS, the improvement of BAPM ranges from 0.91% to 5.60% for WordCount and from 0.90% to 5.10% for TeraSort. When the bandwidth is smaller than the cross point, compared with SRL, the improvement

TABLE IV
THE RESULTS OF EXPERIMENTS

| Testing DataSet | | Testing Data Set 1 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job Type | | WordCount | | | | | | TeraSort | | | | | |
| Bandwidth | | 200Mbps | | | 900Mbps | | | 200Mbps | | | 900Mbps | | |
| Algorithm | | LRS | SRL | BAPM | LRS | SRL | BAPM | LRS | SRL | BAPM | LRS | SRL | BAPM |
| Execution Time(second) | Map | 923 | 926 | 923 | 930 | 921 | 921 | 926 | 926 | 926 | 933 | 921 | 921 |
| | Shuffle | 311 | 389 | 311 | 88 | 92 | 92 | 399 | 466 | 399 | 101 | 108 | 108 |
| | Reduce | 138 | 124 | 138 | 145 | 118 | 118 | 310 | 289 | 310 | 318 | 281 | 281 |
| | Overhead of BAPM | \ | \ | 4 | \ | \ | 3 | \ | \ | 3 | \ | \ | 3 |
| | Total | 1372 | 1439 | 1376 | 1163 | 1131 | 1134 | 1635 | 1681 | 1638 | 1352 | 1310 | 1313 |
| Performance Improvement | Compared With LRS | 0.91%-5.60% | | | | | | 0.88%-6.80% | | | | | |
| | Compared With SRL | 0.90%-5.10% | | | | | | 1.11%-7.10% | | | | | |
| | Compared With Hadoop | 8.90%-17.10% | | | | | | 8.11%-19.26% | | | | | |
| Selection Accuracy | | 92.04% | | | | | | 93.18% | | | | | |

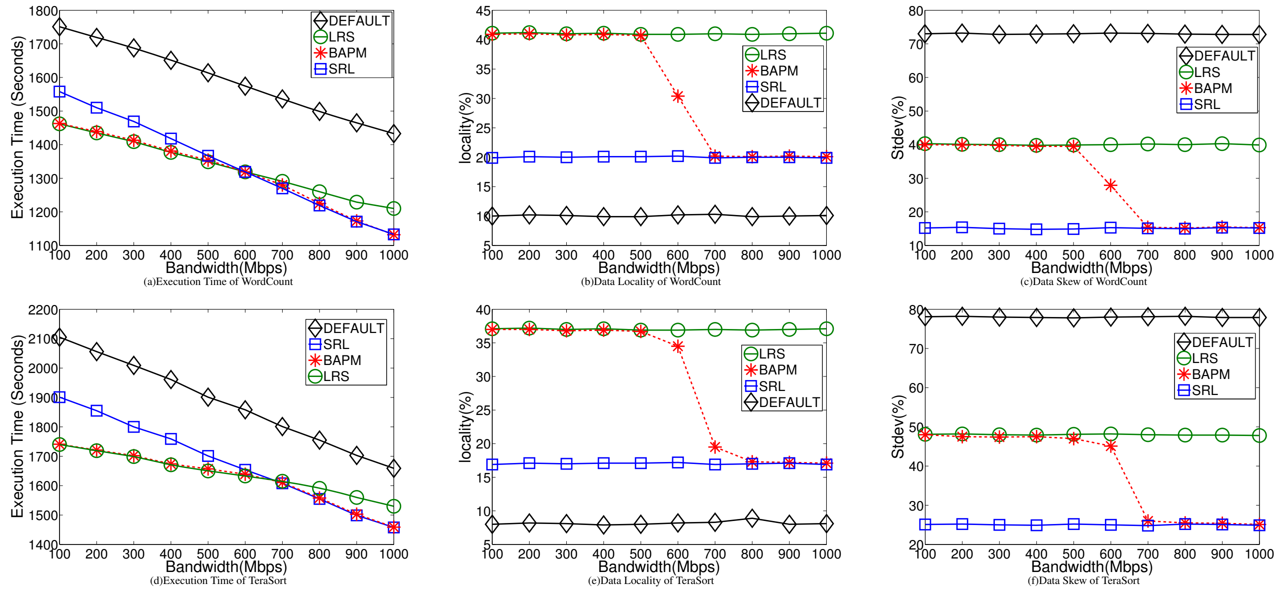


Fig. 4. Evaluation of BAPM

ranges from 0.88% to 6.80% and from 1.11% to 7.10% for the two types of jobs, respectively.

In Figure 4 (a), the two lines that represent LRS and SRL intersect when the bandwidth is set to 608 Mbps; thus, 608 Mbp is a cross point. From Figure 4(a) and (b), we can find that the cross points in experiments are disparate because they are subject to bandwidth, data sets, and job type. Therefore, users can not easily know where the cross point would appear. However, our BAPM can find the cross point accurately so as to reduce the execution time by choosing the LRS on the left of the cross point, and choosing SRL on the right of the cross point. We also discover that BAPM tends to overlap the polyline representing the algorithm that gets the shorter execution time between LRS and SRL when bandwidth is far away from the cross point, such as 200 Mbps and 900 Mbps. This can be explained as the distinct difference of execution time between the LRS and SRL in those cases lead to the conditional probabilities of characteristic attributes in naive Bayes are very high; this makes our BAPM select the proper algorithm more accurately. In contrast, when the bandwidth is set to near to the cross point, such as 500 Mbps and 700 Mbps in Figure 4(a), the polyline of BAPM is slightly different from the fastest algorithm (LRS or SRL). Therefore, when the bandwidth is near to the cross point, the values of LRS and SRL are very close, which means that there is no obvious distinction in terms of execution time between the two algorithms in those cases, and the conditional probabilities of characteristic attributes in naive Bayes are almost equal. Therefore, our BAPM performs proper selection between the two algorithms less accurately. Nevertheless, this can be improved along with more experiments. Figures 3(b)(c)(e)(f) describe the variation of data locality and data skew when running different benchmarks with data set1.

Figure 4 also shows that for any specific data set, the cross point of TeraSort (648 Mbp) is higher than that of WordCount (608 Mbp). This can be explained as the data amount transferred in TeraSort is greater than the amount in WordCount; this causes the degree of dependence on bandwidth in TeraSort jobs is much higher than that in WordCount. We record the execution time in every phase of MapReduce under specific bandwidths in Table IV. We define the Contribution Rate CR[i] to describe the ratio of duration change in phase i, which can be one of map, shuffle, and reduce, in total time change. As shown in Table IV, for Dataset1 on WordCount, when the bandwidth is set to 200 Mbps, CR[shuffle]=$\frac{311-389}{1372-1439}$=$116.4\%$, CR[reduce]=$\frac{138-124}{1372-1439}$=$-20.89\%$. The absolute value of CR[shuffle] is greater than the absolute value of CR[reduce], which verifies our analysis. Because SRL is good at dealing with load imbalance, and the overhead of data transmission is relatively small, BAPM selects SRL and achieves better performance. Overall, BAPM boosts computing performance.

The reasons why BAPM does not lead to large performance degradation even in the case of selection failure are as follow. First, the probability of selection failure is higher when the bandwidth is close to the cross node. However, it does not cause much performance degradation due to the execution time of LRS and SRL are close in this case. Secondly, when BAPM fails to select an algorithm under a bandwidth that is significantly larger or smaller than the cross node, the execution time is much longer than that in the case of a correct selection. Takes the experiment shown in Figure 4(b) for instance, the largest performance degradation is $4.61\%(\frac{1734-1675}{1675})$. Nevertheless, because LRS and SRL achieve greater improvements in execution time than the native Hadoop, BAPM does not cause large performance degradation. We also find that BAPM results in overhead when it selects the proper algorithm using the naive Bayes classifier and updates the statistics. The overhead can be measured by subtracting the execution times of the Map, Shuffle and Reduce phases from the total execution time. The overhead is trivial because the conditional probabilities of the characteristic attributes have been computed and stored in the Training Set module before the job begins to run. As shown in Table 3, the overhead is less than $0.23\%$ ($\frac{OverheadofBAPM}{TotalExecutionTime}$), which could be ignored.

## VI. Conclusions

Locality and data skew on the reduce side are two important factors affecting MapReduce. Experiments have shown that the processing order of the two factors will affect the execution time under different bandwidths. In this study, we propose a bandwidth-aware partitioner, namely, BAPM, which employs the naive Bayes classifier by considering bandwidth and job type as classification attributes for proper selection of the algorithm (LRS or SRL) under various bandwidths. To evaluate the performance of BAPM, we conduct experiments under different bandwidths, and evaluate BAPM using the benchmark WordCount and TeraSort with different testing data sets. Our experimental results show that BAPM can achieve up to 93.18% selection accuracy and reduce execution time by 6.85% and 7.10% when LRS and SRL are used, respectively.

## Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (No.61100143, No.61272353, No.61370128, No.61428201), Program for New Century Excellent Talents in University (NCET-13-0659), Beijing Higher Education Young Elite Teacher Project (YETP0583) and the National Science Foundation of USA (No. 1622292).

## References

[1] Dean J, Ghemawat S., *MapReduce: simplified data processing on large clusters*, Communications of the ACM, vol.51, no.1, PP.107-113,2008.
[2] Zhang H, Huang H, Wang L. *MRapid: An efficient short job optimizer on hadoop*, IEEE Syposium on Parallel and Distributed Processing (IPDPS), pp.459-468, 2017.
[3] Shvachko K, Kuang H, Radia S, *The hadoop distributed file system*, IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp.1-10, 2010.
[4] Zhang H, Wang L, Huang H. *SMARTH: enabling multi-pipeline data transfer in HDFS*, IEEE 43rd International Conference on Parallel Processing, pp.30-39, 2014.
[5] Kawarasaki M, Watanabe H. *System Status Aware Hadoop Scheduling Methods for Job Performance Improvement*, IEICE TRANSACTIONS on Information and Systems, vol.98, no.7, pp.1275-1285, 2015.
[6] Tan J, Meng X, Zhang L. *Coupling task progress for mapreduce resource-aware scheduling*, IEEE Conference on INFOCOM, pp.1618-1626, 2013.
[7] Hammoud M, Sakr M F, *Locality-aware reduce task scheduling for MapReduce*, IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp.570-576, 2011.
[8] Chen Q, Yao J T, Xiao Z, *LIBRA: Lightweight Data Skew Mitigation in MapReduce*, IEEE Transactions On Parallel And Distributed Systems. vol.26, no.9, pp.2520-2533, 2014.
[9] Hsu C H, Slagter K D, Chung Y C, *Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications*, Future Generation Computer Systems, vol.53, no.1, pp.43-54, 2015.
[10] Shadi Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He, Li Qi, *LEEN: Locality/fairness-aware key partitioning for mapreduce in the cloud*, IEEE Second Conference on Cloud Computing Technology and Science (CloudCom), pp.17-24, 2010.
[11] Chen Y, Liu Z, Wang T, Wang L, "Load Balancing in MapReduce Based on Data Locality," Algorithms and Architectures for Parallel Processing. pp.229-241, 2014.
[12] Lei Chen, Wei Lu, Xiaoping Che, Weiwei Xing, Liqiang Wang, and Yong Yang, *MRSIM: Mitigating Reducer Skew In MapReduce*, IEEE 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp:379-384, 2017.
[13] Huang H, Wang L, Lee E J and Chen po, *An MPI-CUDA implementation and optimization for parallel sparse equations and least squares (LSQR)*, Procedia Computer Science, vol.9, pp.76-85, 2012.
[14] Guo P, Huang H, Chen Q, Wang liqiang, Lee En-Jui and Chen Po, *A model-driven partitioning and auto-tuning integrated framework for sparse matrix-vector multiplication on GPUs*, ACM Conference on TeraGrid Extreme Digital Discovery, pp.2, 2011.
[15] Diersen S., Lee E. J., Spears D., Chen P., Wang L., *Classification of seismic windows using artificial neural networks*. Procedia computer science, vol.4, pp.1572-1581,2011.
[16] Huang H., Dennis J. M., Wang L., Chen P., *A scalable parallel LSQR algorithm for solving large-scale linear system for tomographic problems: a case study in seismic tomography*, Procedia Computer Science, vol.18, pp.581-590, 2013.
[17] Subramanian V., Ma H., Wang L., Lee E. J., and Chen P, *Rapid 3d seismic source inversion using windows azure and amazon ec2*, IEEE World Congress on Services (SERVICES), pp.602-606, 2011.
[18] Ma H., Diersen S. R., Wang L., *Symbolic analysis of concurrency errors in openmp programs*, IEEE 42nd International Conference on Parallel Processing (ICPP), PP.510-516.2013.
[19] Subramanian V., Wang L., Lee E. J., and Chen P., *Rapid processing of synthetic seismograms using windows azure cloud*, IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), pp.193-200, 2010.
[20] Lu W, Chen L, Yuan H, Wang L, Xing W, and Yang Y, *Improving MapReduce Performance by Using a New Partitioner in YARN*, The 23rd International Conference on Distributed Multimedia Systems, Visual Languages and Sentient Systems, pp.24-33, 2017.
[21] Guo P., Wang L, *Accurate crossarchitecture performance modeling for sparse matrixvector multiplication (SpMV) on GPUs*, Concurrency and Computation: Practice and Experience, vol. 27(13), pp. 3281-3294, 2015.