IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

VOL. E101-A NO. 5
MAY 2018

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. FUNDAMENTALS, VOL.E101–A, NO.5 MAY 2018

PAPER

Naive Bayes Classifier Based Partitioner for MapReduce

Lei CHEN[†a)], **Wei LU**[†], **Ergude BAO**[†b)], **Liqiang WANG**[††], **Weiwei XING**[†], ***Nonmembers***, ***and*** **Yuanyuan CAI**[†††], ***Member***

SUMMARY MapReduce is an effective framework for processing large datasets in parallel over a cluster. 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 reducer input data is located. Data skew will lead to load imbalance among reducer nodes. Partitioning is an important feature of MapReduce because it determines the reducer nodes to which map output results will be sent. Therefore, an effective partitioner can improve MapReduce performance by increasing data locality and decreasing data skew on the reduce side. Previous studies considering both essential issues can be divided into two categories: those that preferentially improve data locality, such as LEEN, and those that preferentially improve load balance, such as CLP. However, all these studies ignore 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 naive Bayes classifier based partitioner, namely, BAPM, which achieves better performance because it can automatically choose the proper algorithm (LEEN or CLP) by leveraging the naive Bayes classifier, i.e., considering job type and bandwidth as classification attributes. Our experiments are performed in a Hadoop cluster, and the results show that BAPM boosts the computing performance of MapReduce. The selection accuracy reaches 95.15%. Further, compared with other popular algorithms, under specific bandwidths, the improvement BAPM achieved is up to 31.31%.

key words: *MapReduce, hadoop, data locality, data skew, naive Bayes, bandwidth, job type*

1. Introduction

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

Data locality in a distributed environment refers to computing and data are preferably co-located on the same nodes in order to reduce network traffic. The current schedulers in native Hadoop [2] only consider data locality in map tasks and ignore the data locality of reduce tasks. When a user uploads files to HDFS [3], 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: when a file needs to be processed, the job scheduler consults a storage metadata service to get the host node for each chunk 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 the hash partitioning function defined in Equation (1) is performed. Because every intermediate data has a unique hash code, the key-value pairs with the same hash result, which we define as a hash partition, are assigned to the same reduce task.

$$Hash(Hashcode(Intermediate\ data)\ mod\ ReducerNum) \quad (1)$$

In the reduce stage, a reducer takes a partition as input and performs the reduce function on the partition. However, with the hash function, there is a possibility of transferring a large amount of intermediate results to improper reducer nodes, which could cause massive network communication, i.e., data locality might not be achieved and the 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 [4]. The fundamental reason for data skew is that data sets in the real world are often skewed and we do not know the distribution of the data beforehand, which cause the aforementioned hash function in native Hadoop to be inefficient in most cases. Therefore, balancing the hash partition size, which is defined as the size of the key-value pairs with the same hash result, is an important indicator for load balancing among the reducers.

In recent years, several approaches have been proposed to improve the performance of MapReduce by increasing the degree of data locality and decreasing the degree of data skew intensively. Based on the order in which data locality and data skew are considered, relevant studies can be divided into two categories. Studies in the first category consider data locality then data skew, e.g., the LEEN algorithm [5], in contrast, the other category considers data lo-

Manuscript received June 22, 2017.
Manuscript revised December 6, 2017.

[†]The authors are with School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China.
[††]The author is with Department of Computer Science, University of Central Florida, USA.
[†††]The author is with Beijing Key Laboratory of Big Data Technology for Food Safety, School of Computer and Information Engineering, Beijing Technology and Business University, Beijing, 100048, China.
a) E-mail: 13112084@bjtu.edu.cn
b) E-mail: baoe@bjtu.edu.cn

DOI: 10.1587/transfun.E101.A.778

cality based on data skew, e.g., the CLP algorithm [6]. However, these studies do not consider the job type and network bandwidth, according to which the preference for data locality or data skew should be adjusted. 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 and reduce-input-zero. We will describe this in details in Sect. 4.2. When the bandwidth is ineffective, for reduce-input-heavy jobs, data transmission will result in overhead. Therefore, data locality is the major factor that affects the job execution time. In contrast, when the bandwidth among nodes in a Hadoop cluster is relatively high, for reduce-input-light jobs, the data transmission cost will decrease obviously. In this case, data skew, which could cause load imbalance among reducers, will be a major factor that influences the execution time. We will verify this in the experimental section. Therefore, the preference for data locality or data skew may result in varying execution time when running different types of jobs at different bandwidths.

The available bandwidth in a Hadoop cluster depends on the resource configurations status of cluster and the free computing resources that working nodes can provide. The bandwidth between working nodes in Hadoop is relatively stable when the jobs on the nodes are running, but variable and unpredictable when the jobs running on the nodes change, for example, a running job is suspended or a new job begin 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 included in Hadoop, the Resource Manager will limit the bandwidth allocated to the Balancer in order to avoid obvious performance degradation [7]. Secondly, when the Resource Manager assigns a resource container to a task, it will select the node with sufficient resources (including bandwidth) from among all the nodes that hold the input data or its replications of the task, this will also prevent bandwidth mutation [8]. 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 specific bandwidth. In addition, besides job type and bandwidth, there are many other factors that influence the performance of a MapReduce job, such as CPU, memory and container of computing nodes, as well as the number of nodes. Therefore, giving a specific combination of job type and bandwidth, the preference between LEEN and CLP is determined by many factors. It would be better to use a learning technique to make the choice.

In this paper, we propose BAPM, a novel partitioner, which can not only make a proper choice between LEEN and CLP but also leverage the naive Bayes classifier by considering the job type and network bandwidth as classification attributes when a MapReduce job finishes map tasks. We consider the LEEN and CLP algorithms for the following reasons. First, the two algorithms consider both data locality and data skew on the reduce side, but with opposite preferences. Second, both algorithms take intermediate data as input. Hence, the inputs are very similar. Finally, the two algorithms show better performance than native Hadoop. The contributions of this study can be summarized as follows:

(1)We propose BAPM, which improves the job execution time by properly selecting LEEN or CLP in consideration of the job type and bandwidth.

(2)According to the amount of intermediate data that should be transmitted from mappers to reducers, we classify popular jobs, which could be coded using MapReduce model, into a special category. This is useful for our classifier.

(3)We conduct a performance evaluation using BAPM in a Hadoop cluster. Based on our training sets, the selection accuracy of BAPM reaches 95.15%. Comparing with other popular algorithms, the improvement caused by BAPM reaches 31.31%

The rest of this paper is organized as follows. Section 2 reviews some related studies. Section 3 briefly introduces the CLP and LEEN algorithms. Section 4 describes our BAPM in detail. Section 5 describes the performance evaluation of BAPM. Finally, Sect. 6 concludes the paper with a brief discussion on the scope for future work.

2. Related Work

There are many approaches for improving the data locality of Map tasks. Tan et al. [9] designed a resource-aware scheduler for Hadoop to mitigate the job starvation problem and improve the 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. EP [10] formulated a stochastic optimization framework to improve the data locality for reduce tasks with the optimal placement policy exhibiting a threshold-based structure. In presence of job arrivals and departures, EP assigned the Reduce tasks of the current job to proper nodes by taking fetching cost and data locality into account comprehensively.

Current MapReduce implementations have overlooked data skew, which is a major hurdle to achieve successful scale-up in parallel computing systems. SkewTune [11] adjusts the data partition dynamically: after detecting a straggler task, it repartitions the unprocessed data of the task and assigns them to new tasks in other nodes. LIBRA [12] is a system that implements a set of innovative skew mitigation strategies. LIBRA can handle data skew not only on the map side but also on the reduce side. First, LIBRA uses an efficient sampling method to achieve a highly accurate approximation for the distribution of the intermediate data. Then, it speeds up MapReduce by allowing reduce tasks to start copying as soon as the chosen sample map tasks are completed. Finally, it uses range partition to support the splitting of large keys when permitted by application semantics and

performs total order sorting of the output data.

Many studies have comprehensively investigated data locality on the map side as well as data skew. Hsu et al. [13] 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 the locality of map tasks, and it uses virtual machine mapping in the reduce phase in order to balance the workload among reduce nodes.

Some studies [5], [6] have comprehensively investigated data locality for reduce-side as well as data skew, which we will describe in Sect. 3. However, all the aforementioned approaches have ignored the fact that different sequences of data locality and data skew could affect the MapReduce execution time under varying network bandwidth. Our approach, i.e., BAPM, can solve this problem and improve performance.

3. LEEN and CLP

Because BAPM combines LEEN and CLP, we briefly introduce the two algorithms in this section, and we demonstrate their data locality and data skew characteristics through examples.

3.1 LEEN

In order to keep track of all intermediate key frequencies and key distributions, LEEN uses asynchronous map and reduce schemes. The partition function using KeyID as a unique ID is implemented on the intermediate key. Thus, LEEN 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. Finally, when all map tasks are done, all metadata files will be aggregated. Then, the keys will be partitioned into different DataNodes, which store the data on HDFS (Hadoop data file system), according to the LEEN algorithm. LEEN considers data locality ahead of data skew. The LEEN algorithm is described as follows:
(1) Suppose that there are m key-value pairs after the map phase and n Reducers in the cluster.
(2) The hosted data on each node is set to their initial values, which can be obtained from the frequency table.
(3) For a specific key $K_i (0 < i < m)$, in order to achieve the best locality, LEEN selects the node with maximum frequency of key K_i. Therefore, LEEN sorts all nodes in descending order according to the frequency of key K_i in every node $N_j (0 < j < n)$.
(4) LEEN compares the current node with the next node, having the second-hightest frequency. To address the load imbalance among reducer nodes, LEEN introduces the Fairness-Score. The Fairness-Score of each node is the variation in the expected hosted data among all nodes if the key_i is partitioned to this node. A lower value is preferred. If the Fairness-Score of the next node is better than that of the current one, the next node is accepted. Thus, LEEN recursively tries nodes with lower Fairness-Scores.
(5) After selecting the node, LEEN moves all <K_i, value> pairs in a cluster to the selected node N_j and calculates the new values of the hosted data on different DataNodes.
(6) Then, LEEN will continue to process the rest of the keys with the same strategy.

We believe that LEEN considers data locality prior to data skew for the following reason. First, locality is considered in step 3 of the LEEN algorithm, before data skew in step 4. Specifically, when selecting a proper destination node for a special K_i, LEEN first sorts all nodes in descending order according to the frequency of K_i in every N_j. This ensures that the locality of the previous node is better than that of the next node. Then, LEEN considers the data skew through the Fairness Score. Therefore, regardless of which node is finally chosen, its locality is better than that of the subsequent nodes. Actually, from the example presented in Sect. 3.3 and experiments, we find that LEEN's improvement in data locality is greater than that in data skew.

3.2 CLP

CLP considers data locality based on data skew on the reduce side. The CLP partition algorithm consists of three main parts. In first part, CLP uses random sampling to analyze input data and uses an extra MapReduce job to gather information about data distribution. In the second part, CLP designs the same number of reducer nodes, and each data cluster is sent to one reducer. The data with the same key will be stored into the same data cluster. In order to balance the load of each reducer when processing skewed data, CLP provides a heuristic partition method to combine data into data clusters so that every data cluster has similar data size. The final part is the locality partition part, which assigns data clusters to suitable processing nodes according to data locality. CLP uses RP_k $(1 < k < n)$, which represents each of the data clusters produced in the second part, n is the number of reduce nodes. The algorithm in the final part is summarized as follows:
(1) CLP calculates the locality of key K_i in node N_j.
(2) CLP calculates the sum of the data size of each data cluster $RP_k (1 < k < n)$, which is represented by Sum_RP_k, and then sorts the data clusters in descending order of Sum_RP_k.
(3) The locality-partition part enters into n iterations. In this paper, we use Equation (2) to measure the data locality, where $Key_i^j Count$ denotes the number of values of local keys in Map Node j, and $\sum_{i=1}^{m} Key_i^j Count$ denotes the total number of data in Node j. In each iteration, CLP selects the best reducer in consideration of data locality and the impact factor of load balance. The data cluster at the head of RP will be sent to the best reducer.
(4) CLP removes the reducer selected from the reducer set and proceeds to process the remaining data clusters with the same strategy.

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

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

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 LEEN and CLP.

$$KL_i^j = \frac{Key_i^j Count}{\sum_{i=1}^{m} Key_i^j Count} \tag{2}$$

3.3 Example of LEEN and CLP

We present an example that describes different partitioning results on the same intermediate data, as shown in Fig. 1. The example considers three DataNodes and six keys, every DataNode is configured as a mapper node and reducer node concurrently. The numerical values in Fig. 1 represent the frequency of each key. Here, we assume that all records are of equal size.

This example demonstrates the inconsistency in key distribution, which affects data locality on the reduce side. As shown in Fig. 1, the key distribution is inconsistent among the nodes. The data transmission amounts will be noticeably different with various partitioning strategies. In this example, the data transmission amount is 44 records when running LEEN and 47 records when running CLP is, the former is smaller than the latter by 6.82%. When data size is large, LEEN can reduce data traffic significantly.

The example 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 the hash partitioning varies, the distribution of reducer inputs will be different. In this study, we define the data skew rate, expressed by Eq. (3), to describe the degree of balance of key frequency among all the Reducers, where FK_i^j denotes the frequency of $K_i (1 < i < m)$ in the data node $N_j (1 < j < n)$,and Mean denotes the mean of all the FK_i^j values. The data skew rate of the Reducer input is 2.16 when running LEEN and 0.82 when running CLP.

$$DSR = \sqrt{\frac{\sum_{j=1}^{n} (\sum_{i=1}^{k} FK_i^j - Mean)^2}{n}} \tag{3}$$

In this study, a larger data transmission amount leads to longer execution times. Lower data skew rates result in better performance. The results indicate that for LEEN, the improvement in data locality is greater than the alleviation of data skew. In contrast, for CLP, the alleviation of data skew is greater than the improvement in data locality.

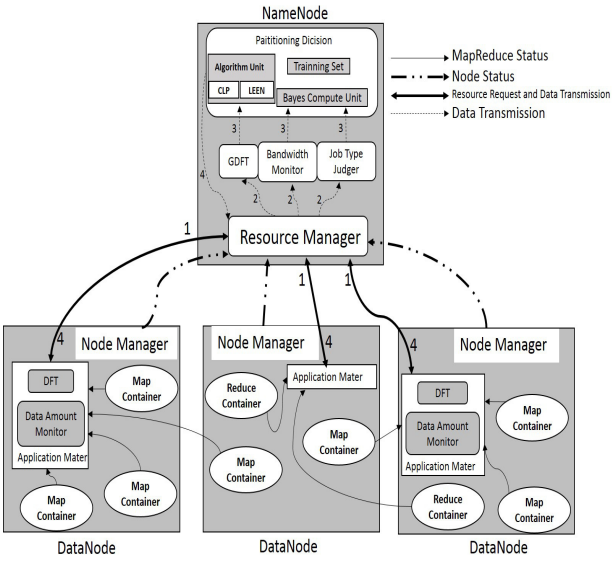


Fig. 2 The architecture of BAPM.

4. BAPM

In this section, we present BAPM, a new partitioner for Mapreduce. First, we discuss how to classify MapReduce jobs and measure bandwidth. Finally, we describe the naive Bayes classifier used in BAPM.

4.1 BAPM in YARN

This section describes the design of the proposed bandwidth-aware partitioner frame, namely, BAPM. The architecture of BAPM is shown in Fig. 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 Bandwidth Monitor obtains real-time information about bandwidth in the cluster (Sect. 4.2). The Job Type Judger will decide the type to which a job belongs (Sect. 4.3). The Partitioning Decision determines the partition algorithm that will be adopted (CLP or LEEN) (Sect. 4.4). The workflow of BAPM consists of 4 steps:

(1) In BAPM, the job type could be input by users or determined automatically. The Data Amount Monitor will compute the amount of input and output of each Map task, this is an essential preparative work for job type determination. DFT counts and records intermediate key-value pairs generated by a map function in every DataNode, the number of pairs corresponding to every key is represented as key frequency. After all the map tasks are completed, the information data, which consists of the job type that the user inputs or the statistical data from the Data Amount Monitor,

Table 1 Job type.

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

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 the information data, it transmits corresponding data to the GDFT and Job Type Judger, respectively. Then, the GDFT will summarize all the key frequencies in each DataNode into a table, which will be used in LEEN and CLP later. The style of GDFT is similar to the tables in Fig. 1. Using the rules described in Sect. 4.1.1, the Job Type Judger determines the job type by analyzing the data from the Data Amount Monitor.
(3) To select the partitioning algorithm (LEEN or CLP), the Partitioning Decision uses three modules: Bayes Compute Unit, Training Set, and Algorithm Unit. After the Partitioning Decision receives information data from the Bandwidth Monitor and the Job Type Judger, it implements a naive Bayes classifier by considering current bandwidth and job type as categorical attributes, and then chooses between LEEN and CLP. We describe this in details in Sect. 4.4. The Training Unit records the 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 information such as the number of times LEEN is selected, the number of times the job type is input-reduce-heavy when LEEN is selected, and so on. 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 algorithm is determined, the Algorithm Unit will generate the partitioning result using the table in GDFT, and the result will be transmitted back through the resource response messages from the Resource Manager to the Application Master.

4.2 Job Type

There are many problems that are suitable for the MapReduce framework. In many situations, users have to be aware of the system resources being used by the MapReduce job, especially inter-cluster network utilization. The execution time of jobs whose mappers produce a large number of key-value pairs is strongly dependent on network conditions in a cluster, therefore, it is necessary to categorize jobs with the amount of intermediate data transmitted from mappers to reducers. If the amount of output of the map phase is almost the same as 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 data transmitted to the reducers is obviously less than the mapper input for a job, 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 the rules stated above and the results are listed in Table 1.

4.3 Bandwidth Monitor

Bandwidth is an important issue in networks. The Bandwidth Monitor in BAPM gets current bandwidth of a link by monitoring the traffic through it ports. Bandwidth Monitor sends a port statistical message to a certain port. Then, a return message is received. From the return message, Bandwidth Monitor gets information such as the number of transmitted and received packets, the number of transmitted and received bytes, the duration of this process, and the number of bytes of the transmitted message. The result is divided by the duration time of the process, which gives the current bandwidth of this port. Moreover, we can obtain the remaining bandwidth of this port by subtracting the current bandwidth from the maximum bandwidth that can be configured. As we have described in Sect. 1, the bandwidth mutation does not happen in Hadoop, so the bandwidth values obtained by the Bandwidth Monitor are effective.

4.4 Bayes Compute Unit

Because the Naive Byes classifier is used in the Bayes Compute Unit, we introduce it here. Given a problem instance to be classified, represented by a vector $X=\{x_1,x_2,...,x_m\}$ containing m features, we assume that all features are independent, $C=\{y_1,y_2,...y_n\}$ denotes a category set that contains n categories. The naive Bayes classifier determines the category $y_i(1\leqslant i\leqslant n)$ 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 $P(y_i|X)(1\leqslant i\leqslant n)$ can be decomposed as

$$P(y_i|X) = \frac{\prod_{j=1}^{m} P(x_j|y_i)P(y_i)}{P(X)} \tag{4}$$

The workflow of the Bayes Compute Unit is shown in Fig. 3 and described as follows. In the preparation stage, we determine the features and their division. There are only two features in BAPM, job type and bandwidth, in vector X, x_1 denotes the job type, x_2 denotes the bandwidth. Obviously, the two features are independent. The division of x_1 includes the job types introduced in Sect. 4.3. The division of x_2, i.e., bandwidth, has been described in Sect. 4.2. In the category set C, y_1 denotes the LEEN algorithm and y_2

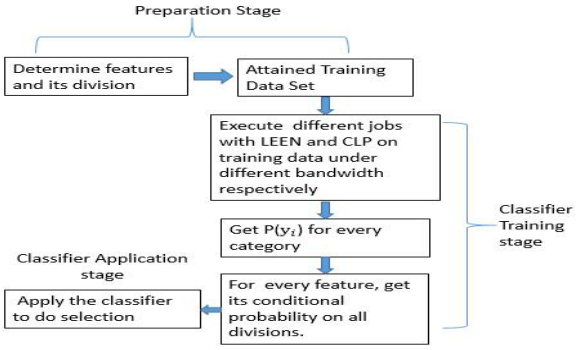
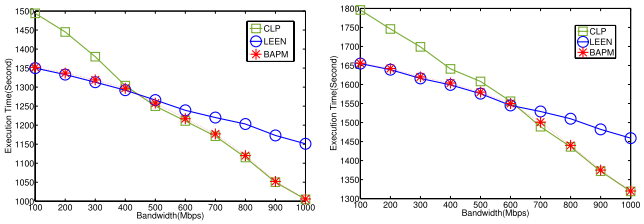


Fig. 3 The flow chart of Bayes classifier.



(a)Execution Time When Running Grep (b)Execution Time When Running Join

Fig. 4 Execution time of LEEN, CPL and BAPM for Grep and Join under various bandwidths.

denotes CLP. After obtaining the training data set, BAPM enters the Classifier Training stage.

In this stage, firstly, BAPM respectively executes a special MapReduce job, WordCount or Sort, with two algorithms, LEEN and CLP, on a training data set under a specific bandwidth. Then, the algorithm with the shorter execution time will be determined as the selected algorithm in this case. In our experiments, BAPM conducts the same work 30 times repeatedly under each of different bandwidths. Finally, BAPM obtains P(y_i) for every category. and the conditional probability of every feature in all divisions.

When a MapReduce job with testing data set arrives, BAPM uses the trained Naive Bayes classifier, to perform the classification. The reasons for employing the naive Bayes classifier to perform selection are as follows. First, the naive Bayes classifier works well because the categorical attributes, bandwidth and job type, are independent. Secondly, compared with other classifiers, the naive Bayes classifier shows efficient and stable classification when the number of features and categories are not large [15]. In our BAPM, we have only two features and two categories. Thirdly, because BAPM is able to compute and record the results in a component namly the Training Unit (introduced in Sect. 4.2) each time, we can efficiently obtain the previous probability using the naive Bayes classifier in the next instance.

5. Evaluation

5.1 Experiment Environment and Preparation

In this study, all experiments are performed on a homogeneous Hadoop cluster running a stable version of Hadoop 2.6.0. Our experiments are executed on 7 servers with 16x86 4 cores and 16 GB of RAM. The servers are interconnected by an Ethernet switch with 1 Gbps links. We evaluate BAPM performance in a virtual cluster comprising 31 virtual machines (VMs). A VM is deployed on a server to act as the master node (Namenode). We also deploy five VMs on each of the six PMs, hence, the cluster size is 30 nodes (DataNodes), the maximum number of tasks on each DataNode is set to 6. All the virtual machines are configured with 2 CPU cores and 1 GB memory. We configure the HDFS chunk size to be 64 MB. Because 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 train the Bayes classifer by running WordCount and TeraSort on a 8 GB synthetic data set of data skew rate 0.2. which is described in Sect. 4.4. BAPM's training stage is conducted before the classification application stage, so there should be no training time spent for any specific MapReduce job, and therefore, we do not take it into consideration when calculating the total time.

In order to ensure accuracy, in this paper, we perform each group of experiments at least 10 times and take the mean value as the final result so as to reduce the influence of the variable environment.

5.2 Selection Accuracy of BAPM

To verify BAPM's improvement performance by doing proper selection from LEEN and CLP, we run BAPM for Grep and Join, which represent reduce-input-light and reduce-input-heavy jobs, respectively, on a 12 GB synthetic data set of data skew rate 0.6.

Figure 4 shows execution time of LEEN, CLP and BAPM for Grep and Join under various bandwidths. LEEN exhibits less execution time than CLP under relatively small bandwidths; while under large bandwidths, CLP outperforms LEEN. Therefore, there is a turning point on bandwidth between LEEN and CLP to obtain optimal performance. BAPM can find such a tuning point on bandwidth. We notice that BAPM's polyline tends to overlap the polyline representing the algorithm that gets the shorter execution time between LEEN and CLP when bandwidth is far away from the turning point, such as 200 Mbps and 900 Mbps. This can be explained as the distinct difference of execution time between the LEEN and CLP in those cases lead to a high conditional probabilities of characteristic attributes in naive Bayes, which makes our BAPM select the proper algorithm accurately. In contrast, when the bandwidth is near to the tuning point, such as 400 Mbps and 500 Mbps in Fig. 4(a), the BAPM's polyline is slightly different with the lowest polyline, which means that there is no obvious distinction in terms of execution time between the two algorithms in those cases, and it leads to the conditional probabilities of characteristic attributes in naive Bayes be-

Table 2 The statistical table of experiments.

Testing DataSet		Testing Data Set											
Job Type		Grep						Join					
Bandwidth		200 Mbps			900 Mbps			200 Mbps			900 Mbps		
Algorithm		LEEN	CLP	BAPM	LEEN	CLP	BAPM	LEEN	CLP	BAPM	LEEN	CLP	BAPM
Execution Time(second)	Map	923	926	923	930	921	921	924	926	924	933	921	921
	Shuffle	281	390	281	66	72	72	401	526	401	97	111	111
	Reduce	138	125	138	182	59	59	310	293	310	436	290	290
	Overhead of BAPM	\	\	4	\	\	3	\	\	3	\	\	3
	Total	1342	1441	1346	1178	1052	1055	1635	1745	1638	1466	1322	1325
Selection Accuracy		95.15%						93.98%					

ing almost equal. Therefore, our BAPM can perform proper selection between the two algorithms, but not easily. Nevertheless, this can be improved along with the increase in the number of times that our Training Data Set is processed.

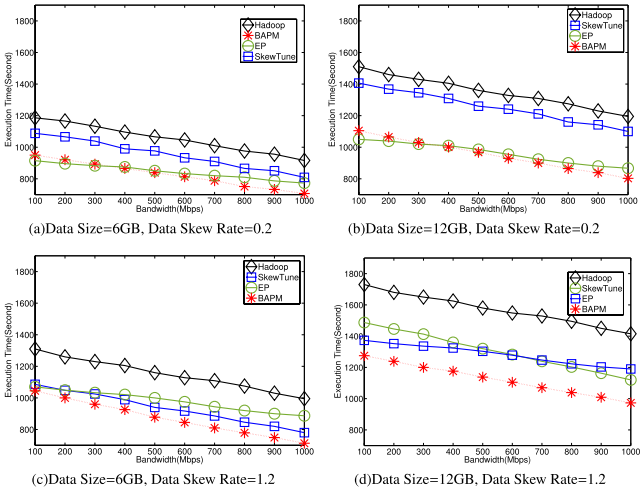
We record the execution time in every phase of MapReduce. Table 2 shows the results under bandwidths of 200 Mbps and 900 Mbps. For Grep, when the bandwidth is 200 Mbps, LEEN is 109 seconds faster than CLP in the shuffle phase, and is only 13 seconds slower than CLP. This indicates that data locality has more impact on execution time than load balance, so LEEN should be in favor and BAPM does select it. Similarly, BAPM selects the proper algorithm in all other cases. Generally, BAPM achieves 95.15% and 93.98% selection accuracy for Grep and Join, respectively. The improvement in execution time reaches 10.34% and 9.58% when LEEN and CLP are selected, respectively.

BAPM cannot lead to large performance degradation even in the case of selection failure. First, the probability of selection failure is higher when the bandwidth is set to be close to the turning point. In this case, the execution times of LEEN and CLP are close, therefore, even if BAMP selects the wrong algorithm, it will not cause much performance degradation. Secondly, when BAPM fails to select an algorithm under a bandwidth that is significantly larger or smaller than the turning point, the execution time is much longer than that in the case of a correct selection. Nevertheless, because LEEN and CLP achieves greater improvements in execution time than native Hadoop, which will be described in Sect. 5.3, BAPM can not cause large performance degradation. From Table 2, we also find the overhead of BAPM is trivial because the conditional probabilities have been computed and stored in the training stage, and only result querying is needed in the classification application stage.

5.3 Evaluate BAPM by Running Various Types of Jobs

In this section, we run BAPM for Grep and Join, which represent reduce-input-light and reduce-input-heavy jobs on various data sets, respectively. We compare BAPM with SkewTune [11], EP [10] and native Hadoop. SkewTune focuses on the performance degradation caused by data skew in MapReduce and EP improves data locality of Reduce tasks.

Figure 5 shows execution time of Skew-Tune, EP, Hadoop and BAPM for Grep on various data sets. In



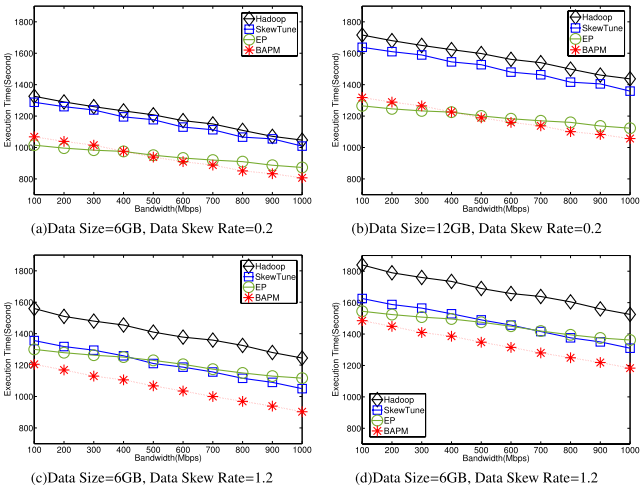
(a)Data Size=6GB, Data Skew Rate=0.2 (b)Data Size=12GB, Data Skew Rate=0.2

(c)Data Size=6GB, Data Skew Rate=1.2 (d)Data Size=12GB, Data Skew Rate=1.2

Fig. 5 Execution time of Skew-Tune, EP, Hadoop and BAPM for Grep on various data sets.

Fig. 5(a), size of the data set is 6 GB and data skew rate is 0.2. BAPM and EP have the smallest execution time compared with the other two algorithms under all bandwidths. This is because both EP and BAPM seek to improve data locality, which is a major affecting factor with a relatively low data skew rate. For BAPM and EP, when the bandwidth is relatively low, EP achieves a little faster performance, this is because BAPM takes extra overhead to do data sampling and selection from LEEN and CLP. Nevertheless, when the bandwidth is large, BAPM outperforms EP. This is because BAPM also considers data skew, which becomes more important when the bandwidth grows. In Fig. 5(b), the data set size is changed to 12 GB, and the data skew rate is kept at 0.2. All the algorithms spend more time, but maintain similar relative difference as in Fig. 5(a). In Figs. 5(c) and 5(d), the data skew rate is changed to 1.2, and the sizes are 6 GB and 12 GB, respectively. BAPM outperforms all the other algorithms under all bandwidths. This is because BAPM comprehensively considers both data skew and data locality. Specifically, compared with Hadoop, SkewTune and EP, the improvement range of BAPM is 11.1–28.6%, 10.9–19.8% and 5.4–16.7%, respectively.

Figure 6 shows execution time of Skew-Tune, EP, Hadoop and BAPM for Join on the same four data sets we have described in the previous paragraph. We notice that all the algorithms maintain similar relative difference as in Fig. 5, but spend more time. This can be explained as Join is a reduce-input-heavy job, it generates much data traffic than



(a)Data Size=6GB, Data Skew Rate=0.2 (b)Data Size=12GB, Data Skew Rate=0.2

(c)Data Size=6GB, Data Skew Rate=1.2 (d)Data Size=6GB, Data Skew Rate=1.2

Fig. 6 Execution time of Skew-Tune, EP, Hadoop and BAPM for Join on various data sets.

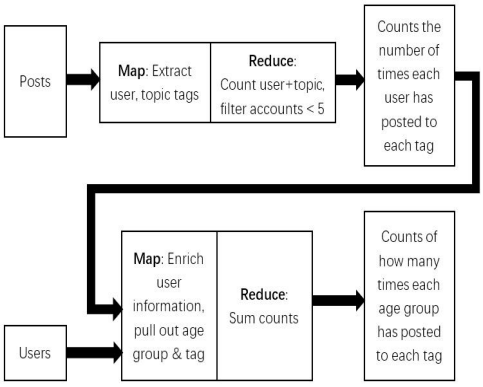


Fig. 7 MapReduce job chain.

Grep. In conclusion, compared with Hadoop, SkewTune and EP, the improvement range of BAPM is 19.5–31.3%, 15.6–22.8% and 13.5.4–18.7%, respectively.

5.4 Evaluate BAPM on a Social Network Analysis Application

We run BAPM on a social network analysis application on two data sets of www.stackoverflow.com, *Posts* and *Users* of sizes 10.58 GB and 3.55 GB. Given these two data sets, the intent of the application is to count how many times each age group of *Users* has posted to each tag in *Posts*. The application contains two MapReduce jobs. As Fig. 7 shows, the first job is a reduce-input-light job similar to WordCount, while the second job is a reduce-input-heavy job running Join to enrich user information. BAPM is compared with SkewTune, EP and naive Hadoop under various bandwidths, and the results are shown in Fig. 8. Specially, we notice that when the bandwidth is 200 Mbps, the first job of BAPM, i.e., *BAPM(job1)* in Fig. 8, is slower than *EP(job1)*, the reason is described in Sect. 5.3. This disadvantage is offset by *BAPM(job2)*. Finally, the total duration of BAPM is the shortest in this case. Overall, BAPM has the fastest performance under all the bandwidths for all the jobs. Specifically, BAPM is 8.66%–29.58% faster than the other algorithms.

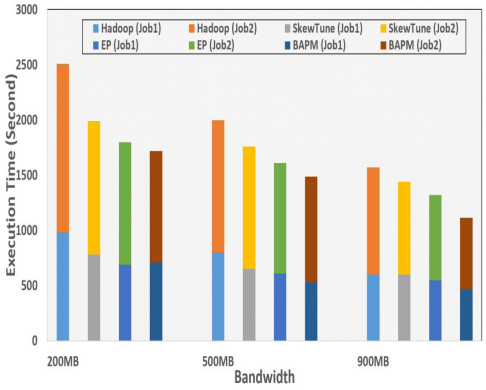


Fig. 8 Execution time of Skew-Tune, EP, Hadoop and BAPM on real data sets.

6. Conclusions

Locality and data skew on the reduce side are two important performance factors in MapReduce. Experiments have shown that the processing order of the two factors affects the execution time under different bandwidths. In this study, we first 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 (LEEN or CLP) under various bandwidths. Then, we divide MapReduce jobs into three types according to the amount of intermediate data transmit from the map nodes to the reduce nodes in a cluster after the map phase. To quantify the performance of BAPM, we conduct experiments in a Hadoop cluster under different bandwidths, and use BAPM to execute various benchmarks and a social network analysis application with different testing data sets. Our experimental results show that BAPM can achieve up to 95.15% selection accuracy and 31.31% improvement in execution time under specific bandwidths.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (No. 61100143, No. 61272353, No. 61370128, No. 61428201), Beijing Natural Science Foundation (No. 4184084), Humanity and Social Science Youth Foundation of Ministry of Education of China (No. 17YJCZH007), Research Foundation for Youth Scholars of Beijing Technology and Business University (No. QNJJ2017-17).

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol.51, no.1, PP.107–113, 2008.
[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," IEEE 26th Symposium on Mass Storage

Systems and Technologies (MSST), pp.1–10, 2010.
[3] H. Zhang, L. Wang, and H. Huang, "SMARTH: Enabling multi-pipeline data transfer in HDFS[C]," 2014 43rd International Conference on Parallel Processing. IEEE, pp.30–39, 2014.
[4] M. Kawarasaki and H. Watanabe, "System status aware Hadoop scheduling methods for job performance improvement," IEICE Trans. Inf. & Syst., vol.E98-D, no.7, pp.1275–1285, July 2015.
[5] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. 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.
[6] Y. Chen, Z. Liu, T. Wang, and L. Wang, "Load balancing in MapReduce based on data locality," Algorithms and Architectures for Parallel Processing, pp.229–241, 2014.
[7] H. Huang, J.M. Dennis, L. Wang, and P. Chen, "A scalable parallel LSQR algorithm for solving large-scale linear system for tomographic problems: A case study in seismic tomography[J]," Procedia Computer Science, vol.18, pp.581–590, 2013.
[8] V. Subramanian, L. Wang, E.-J. Lee, and P. Chen, "Rapid processing of synthetic seismograms using Windows azure cloud[C]," Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, pp.193–200, 2010.
[9] J. Tan, X. Meng, and L. Zhang, "Coupling task progress for mapreduce resource-aware scheduling," IEEE Conference on INFOCOM, pp.1618–1626, 2013.
[10] J. Tan, S. Meng, X. Meng, and L. Zhang, "Improving reducetask data locality for sequential mapreduce jobs[C]," INFOCOM, 2013 Proceedings IEEE. IEEE, pp.1627–1635, 2013.
[11] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating skew in mapreduce applications[C]," Proc. 2012 ACM SIGMOD International Conference on Management of Data, ACM, pp.25–36, 2012.
[12] Q. Chen, J.T. Yao, and Z. Xiao, "LIBRA: Lightweight data skew mitigation in MapReduce," IEEE Trans. Parallel Distrib. Syst., vol.26, no.9, pp.2520–2533, 2014.
[13] C.H. Hsu, K.D. Slagter, and Y.C. Chung, "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.
[14] L.A. Adamic and B.A. Huberman, "Zipf's law and the Internet," Glottometrics, vol.3, no.1, pp.143–150, 2002.
[15] S. Diersen, E.-J. Lee, D. Spears, P. Chen, and L. Wang, "Classification of seismic windows using artificial neural networks[J]," Procedia Computer Science, vol.4, pp.1572–1581, 2011.
[16] H. Huang, L. Wang, E.-J. Lee, and P. Chen, "An MPI-CUDA implementation and optimization for parallel sparse equations and least squares (LSQR)[J]," Procedia Computer Science, vol.9, pp.76–85, 2012.

Lei Chen is a Ph.D candidate in computer science of Beijing Jiaotong University. He received the M.S. degree in 2011 in the School of Computer Science at Taiyuan University of Technology, China. His research interests are in the areas of distributed data processing and cloud computing.

Wei Lu received his Ph.D degree from Sichuan University in 2006. Currently, He is a professor and the dean of School of Software Engineering in Beijing Jiaotong University. He serves as a member of Software Engineering Professional Education Committee with the Ministry of Education in China. His research interests in service computing and network.

Ergude Bao received his Ph.D degree from Department of Computer Science and Engineering, University of California, Riverside, United States, in 2014. He is currently an associate professor in School of Software Engineering at Beijing Jiaotong University. His research interests include design of algorithms and intelligent systems, and parallel computation.

Liqiang Wang received Ph.D degree in Computer Science from Stony Brook University in 2006. He is an Associate Professor in the Department of Computer Science at the University of Central Florida. His research interest is the design and analysis of parallel systems for big-data computing, which includes two aspects: design and analysis. He received an NSF CAREER Award in 2011.

Weiwei Xing received the B.S. degree in computer science and Ph.D degree in signal and information processing from Beijing Jiaotong University, in 2001 and 2006 respectively. Currently, she is a professor at School of Software Engineering, Beijing Jiaotong University. Her research interests mainly include intelligent information processing and artificial intelligence.

Yuanyuan Cai received her Ph.D degree from Beijing Jiaotong University in 2016. Currently, She is a lecturer in the Department of Information Management, School of Computer and Information Engneering at Beijing Technology and Business University. Her research interests focus on semantic computation, information retrivel and natural language processing