Distributed Modeling in a MapReduce Framework for Data-Driven Traffic Flow Forecasting

Cheng Chen, Zhong Liu, Wei-Hua Lin, Shuangshuang Li, and Kai Wang

***Abstract*—With the availability of increasingly more new data sources collected for transportation in recent years, the computational effort for traffic flow forecasting in standalone modes has become increasingly demanding for large-scale networks. Distributed modeling strategies can be utilized to reduce the computational effort. In this paper, we present a MapReduce-based approach to processing distributed data to design a MapReduce framework of a traffic forecasting system, including its system architecture and data-processing algorithms. The work presented here can be applied to many traffic forecasting systems with models requiring a learning process (e.g., the neural network approach). We show that the learning process of the forecasting model under our framework can be accelerated from a computational perspective. Meanwhile, model fusion, which is the key problem of distributed modeling, is explicitly treated in this paper to enhance the capability of the forecasting system in data processing and storage.**

***Index Terms*—Distributed modeling, MapReduce, model fusion, traffic flow forecasting.**

I. Introduction

With the growth of utilizing more sophisticated sensing and communication technologies in urban traffic management systems, a massive amount of traffic data can now be acquired with little effort. As is well known, traffic is a nonstationary process. A traffic system within a network is a huge complex dynamic system [1], [2]. With the availability of data from various sources, it is important for traffic management systems to quickly process data into necessary forms or information, particularly in the situation for real-time operations. One of the applications that require such a capability is short-term traffic flow forecasting. Of many different approaches proposed for traffic forecasting, one particular approach is based on data mining, including methods based on the historical average [3], Kalman filter theory [4], neural network model [5], fuzzy-neural model [6], type-2 fuzzy logic model [7], and Bayesian network model [18]. The learning or training process embedded in these models is essentially data driven. Thus, this type of approach can also be classified as a data-driven approach. To achieve better performance, the parameters of models in this approach are usually quite frequently updated on a real-time basis with newly available data. The computational effort for this approach is often quite demanding as the computational complexity for these forecasting models typically increases with the increase in performance requirement or prediction accuracy. For some algorithms, it is desirable to make use of as much real-time data as possible to capture the nonstationarity behavior of traffic. Even for the training process in which historical data are largely used, it is still preferable to use more recent data since the traffic condition can be affected by many subtle changes in demand or roadway configurations (e.g., the change in demand resulting from the recent completion of a shopping mall in the neighborhood, etc.). The learning process should then keep up with the dynamic changes in the environment. Unfortunately, one predicament of data processing of the learning process in standalone mode is that the computation and storage burden of traffic forecasting systems can rapidly grow, resulting in actually degrading the performance of the forecasting system. This problem, however, can be solved by utilizing a distributed system. Under the distributed modeling approach, every block of newly collected data can be viewed as a new local model to the existing forecasting system. Taking the old global model as a local model, we can use model fusion to update the forecasting system online. The main issue in accomplishing this is how to effectively manage the computation time and storage space. In this paper, we propose a distributed model learning approach based on the technologies of cloud computing to explicitly deal with this issue.

Cloud computing is a new computing paradigm in the IT industry and has gradually become a trend for massive data processing. Aiming to use a unified framework to construct a large amount of low-end servers to handle various computing and storage requirements, it has the advantage of massive scalability and economies of scale. Hence, cloud computing holds promise in many fields, including transportation [8]. In this paper, we take advantage of cloud computing to solve the problems in the process of model learning. With the development

Manuscript received August 30, 2011; revised May 4, 2012; accepted June 13, 2012. Date of publication July 10, 2012; date of current version February 25, 2013. This work was supported in part by National Natural Science Foundation of China under Grant 70890084, Grant 60921061, and Grant 90920305 and in part by the Chinese Academy of Sciences under Grant 2F09N05, Grant 2F09N06, Grant 2F10E08, and Grant 2F10E10. The Associate Editor for this paper was L. Li.

C. Chen and S. Li are with the State Key Laboratory of Management and Control for Complex Systems, Chinese Academy of Sciences, Beijing 100190, China (e-mail: cheng.chen@ia.ac.cn; shuangshuang.li@ia.ac.cn).

Z. Liu is with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China (e-mail: liuzhong@nudt.edu.cn).

W.-H. Lin is with the Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ 85721 USA (e-mail: whlin@email.arizona.edu).

K. Wang is with the College of Mechatronics Engineering and Automation, National University of Defense Technology, Changsha 410073, China, and also with the State Key Laboratory of Intelligent Control and Management of Complex Systems, Chinese Academy of Sciences, Beijing 100190, China (e-mail: kai.wang_nudt@hotmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2012.2205144

of cloud computing, MapReduce, which is a programming model first created by Google [9], has received much attention and has been adopted in many different fields, such as the analysis of biological information [10], text processing [11], the analysis of social network [12], etc. Using a large amount of independent computing to handle the storage and processing of data partition, MapReduce is presented to support distributed computing with large-scale data sets on one cluster of computers. In this paper, we will integrate MapReduce into a traffic flow forecasting system to construct the data storage subsystem and data-processing subsystem. The main contribution of this paper is given here.

1) We propose a method of distributed modeling under a MapReduce framework for data-driven traffic flow forecasting to meet the computation and data storage requirement for this particular application. For the purpose of presentation, the proposed framework is to be discussed in the context of the Bayesian network approach for traffic flow forecasting since the Bayesian approach shares many common features of information utilization in the network setting. The framework developed is general enough that can be applied to other data-driven traffic forecasting approaches as well.
2) We have addressed the three important problems that would often arise in the distributed learning model but not in the stand-alone learning model. These three problems are given as follows:
   a) selection of the number of components in the local Gaussian mixture model (GMM);
   b) realization of model fusion in an effective manner in terms of reducing computation time and improving accuracy;
   c) realization of merging several local models into a global model;
3) We have implemented the Expectation Maximization (EM) algorithm in the MapReduce framework.
4) We have identified the best load of the slave server for the computing clusters, which is useful for the determination of the number of servers in the ideal situation.

The remainder of this paper is organized as follows: In Sections II, we first present an architecture based on MapReduce for distributed modeling. Distributed modeling and model fusion will then be discussed in detail, respectively, in Sections III and IV. In Section V, some outcomes of the experiments on our traffic flow forecasting system are presented and discussed. Section VI provides a summary and a brief discussion of the direction of future work.

II. Architecture Based on MapReduce for Distributed Modeling

Many existing traffic forecasting algorithms were developed, considering only a single site, instead of a whole region. In the former case, the data processing part is not an issue. Recently, more and more research efforts have been devoted toward developing systems for region-wide traffic flow forecasting [18], [22]. When it comes to a whole region, the data issue can be very pronounced. It is very important to study the utilization of information in such a network.

Consider a region consisting of only 100 intersections in Parallel Traffic Management System [13] (PtMS) in Guangzhou, China. Each intersection of this region produces nearly 28-MB traffic flow training samples each year. These training samples only contain 4-D traffic flow information. Suppose that the traffic flow information only involves the vehicle flow rate in different directions of different intersections. When we consider the use of a historical data set of the past three years, the magnitude of the data considered can grow to nearly 7.5 GB. If we consider a single model learning algorithm, such as the EM algorithm, 4 s will be needed for processing data for a single intersection per day. It follows that the time needed for traffic forecasting in the entire region can take up to as many as five days. If we use a neural network or fuzzy-neural approach, the time needed for the entire training process can very quickly go up. Moreover, the training sample set will get bigger and bigger with the expansion of the area and the use of additional traffic information, such as turning movements and queue lengths. Therefore, it is necessary to shorten this process.

To expedite the training process, we design our forecasting system based on a MapReduce framework for data storage and data processing. In this section, we first give a brief introduction to the processing of a MapReduce job and then present in detail the architecture of a general MapReduce-based traffic flow forecasting system.

A. Overview of a MapReduce Framework

MapReduce, as one famous parallel programming model, has two important phases: 1) map phase and 2) reduce phase. The input and output of jobs both have the form of key value pair. The detail of data processing in the map and reduce phases can be designed differently by users with respect to different requirements. As shown in Fig. 1, the input data will first be split into data partition for each map phase. The data partition will then be processed in the map phase into an intermediate data block. The intermediate data block also takes the form of key value pairs and will be further partitioned into several parts with respect to different key values. Those that have the same key value will be sorted into the same slave node in the reduce phase.

B. General MapReduce Framework of Modeling in Traffic Flow Forecasting System

Based on the MapReduce framework in [15], we show in Fig. 2 a general architecture of the traffic flow forecasting system. In Step 0, according to different distributed traffic flow forecasting algorithms, an engine will be started to handle the task of modeling. Then, traffic flow data will be split into several pieces with reasonable size and then stored in a single distributed file system (see Step 1.1). At the same time, the engine will manage a mapping of files in Step 1.2. After that, the engine will handle management of mappers and reducers. The engine is responsible to send the map tasks to mappers (see Step 2.1). Corresponding algorithms and split data will

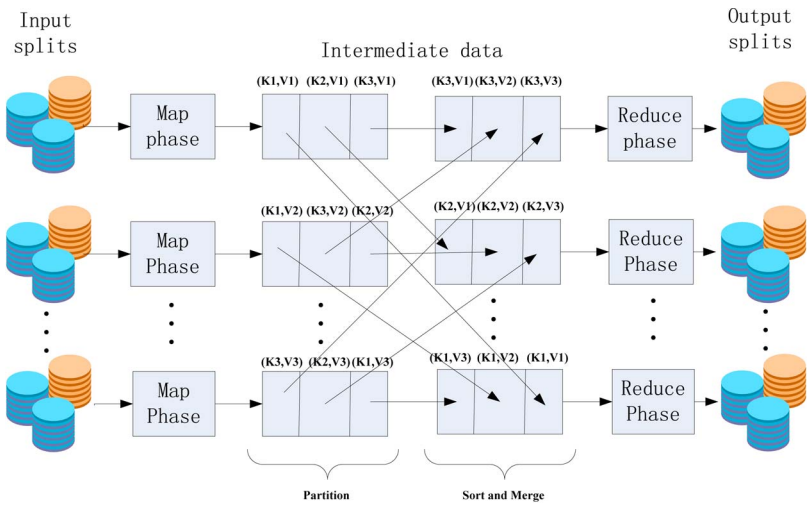


Fig. 1. Framework for processing a MapReduce job.

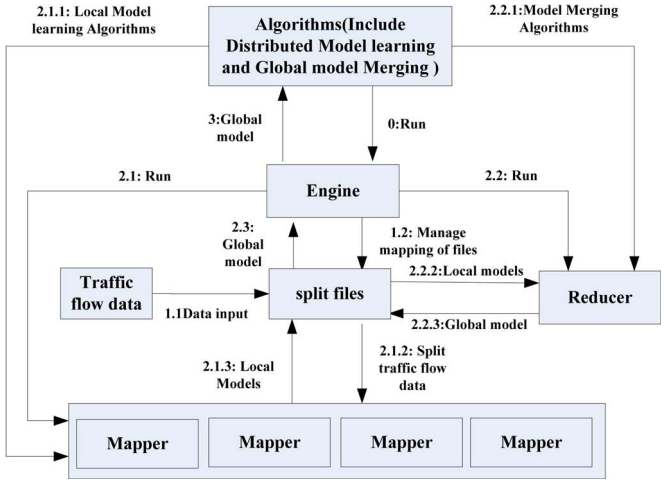


Fig. 2. General MapReduce framework of modeling in traffic flow forecasting system.

also be received by mappers (see Steps 2.1.1 and 2.1.2). The intermediate outcomes of mappers will be stored in a local file system for reducers (see Step 2.1.3). The reducers will then be invoked by the internal engine to process these local models for generating a global model (see Step 2.2). In this phase, reducers use the model fusion algorithm and local models to produce a global model of traffic flow data (see Steps 2.2.1, 2.2.2, and 2.2.3). Finally, the global model is formed and will be returned for forecasting traffic flow (see Steps 2.3 and 3).

C. Architecture of a Traffic Flow Forecasting System

The whole process of modeling in our traffic flow forecasting system is divided into two phases: 1) the data-storage phase and 2) the data-processing phase, as shown in Fig. 3. The architecture of our traffic flow forecasting system contains two kinds of computer node. One is responsible for taking charge of the whole system, such as task assignment and scheduling, whereas the other is for completion of one specific task. In the phase of data storage, the two kinds of nodes are Namenode and Datanode. Here, we use the Hadoop Distributed File System (HDFS) to realize this part of the system. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. In addition, it is highly fault tolerant and is designed to be deployed on low-cost hardware [14]. For a large-scale road map, HDFS can handle the requests of short-term high-speed data writing and long-term large data storage. Using a master/slave architecture as MapReduce data processing framework, the Namenode in HDFS is responsible for the management of the file system Namespace and regulates access to files for clients. User data never flow through the Namenode. It is the responsibility of Datanodes to serve the read/write requests from the clients of the file system. In the phase of data processing, the goal of the system is to acquire useful information from raw data. The time required for processing in a distributed system is made up of two parts: 1) computation time and 2) data transmission time. With large distributed data and limited network bandwidth, to decrease the rate of data transmission time in the total processing time, moving computation codes is less expensive than transferring raw data.

Here, the two kinds of computer nodes often used are Masternode and Slavenode. Masternode is responsible for the distribution of algorithms, management of task progress, etc. Slavenode takes charge of running specific algorithms on one data partition. In addition, the data-processing component follows the MapReduce framework to dispersedly process distributed traffic flow data. The Map phase takes the EM algorithm (one kind of model learning algorithm) to handle the task of local model learning. The input of EM is distributed traffic data sets. The output is the parameters of local models. In this phase, for the characteristics of the EM algorithm, the actual processing is iterative, containing hundreds of MapReduce jobs. The number of iterations is controlled by Masternode. The Reduce phase takes the global model merging algorithm to handle the task of merging local models. For hundreds of local models, the global model merging algorithm can return a single reasonable global model based on the similarity of these local models. When the global model of historical flow data is found, it can then be used for traffic flow forecasting with real-time traffic flow data.

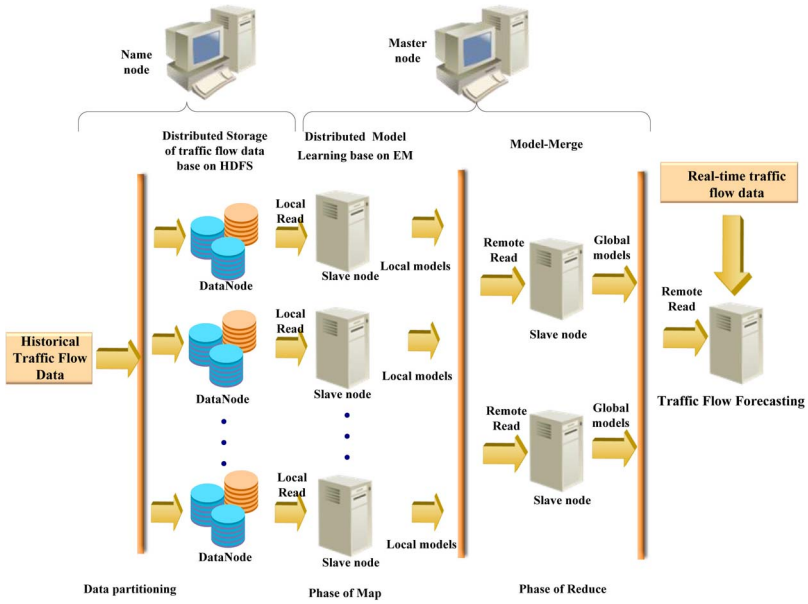


Fig. 3. Architecture of our traffic flow forecasting system.

III. Distributed Local Model Learning

In this section, we will show the process of data partitioning and local modeling of distributed data (see Fig. 3). This process corresponds to the stages from Step 2.1.1 to Step 2.1.3 described in the flowchart in Fig. 2. There are two phases in the process of local modeling, i.e., model selection and parameter optimization.

A. Model Selection

1) GMM for Traffic Flow Data and X-Means for Estimating About the Number of Clusters: GMM is a common probabilistic model for representing the probability distribution of data. We adopt it in our framework for its power in approximating arbitrary probability distributions. In GMM, the total model for the distribution of traffic flow data can be represented as follows:

$$p(X|\Theta) = \sum_{i=1}^{M} w_i p_i(X|\theta_l) \tag{1}$$

where X is a training data set, M is the number of mixture Gaussian models, and Θ and θ are the parameters of the total model and mixture model, respectively.

GMM has been widely used to fit a probability distribution of traffic flow. In standalone mode, a full range of historical data can be made available for the entire learning process. However, the actual traffic flow data can be roughly categorized into one of the three states, i.e., light, nominal, or congested traffic conditions. Therefore, three Gaussian models can be used to characterize the entire data set. Distributed traffic flow model learning for each slave server in the cluster can only see a portion of the historical data (e.g., only traffic flow data in the state of light traffic condition). The use of three Gaussian models for fitting the data may lead to large errors in the phase of model fusion. This will be explained further with a detailed example. A different choice of the number of submodels in the mixture model can lead to a different distribution of flow data. In Fig. 4, we show with a rather extreme case how this choice can influence the resulting estimates. The raw data in this figure are produced by the mixture model with three Gaussian distributions, as shown by three data blocks in Fig. 4.1. Each block contains only the data produced by one Gaussian distribution of the mixture model. Estimates are obtained with two different methods, i.e., K-means and X-means. For the K-means method, each split data block is approximated by K (a predetermined number) submodels ($K = 3$ in this case) (see Fig. 4.3.1–4.3.3). Those models for different blocks (see Fig. 4.5) are further merged, resulting in a single model for each block (see Fig. 4.6). The merge process is based on the distance function that will be discussed in a later section. The resulting clustering appears to deviate from the original data blocks since, with a fixed number of submodels, it is more likely to over- or underrepresent some data blocks. For this particular case, the data in the top block are underrepresented, whereas the data in the mid block are overrepresented.

Better estimates can be obtained by using the X-means method [16], which does not require a predetermined number for submodels. The number of submodels is determined in the clustering process based on Bayesian Information Criterion (BIC) or Akaike Information Criterion (AIC) [17]. X-means is an algorithm that searches both the space of cluster locations and the number of clusters. After using different models to approximate the initial distribution, X-means will score each model based on some criterion and choose the best one as the outcome. The mixture of smaller models will lead to better approximates. However, too many small models may degrade the generalization ability and increase the burden in the phase of model fusion. Since BIC tends to favor smaller models than

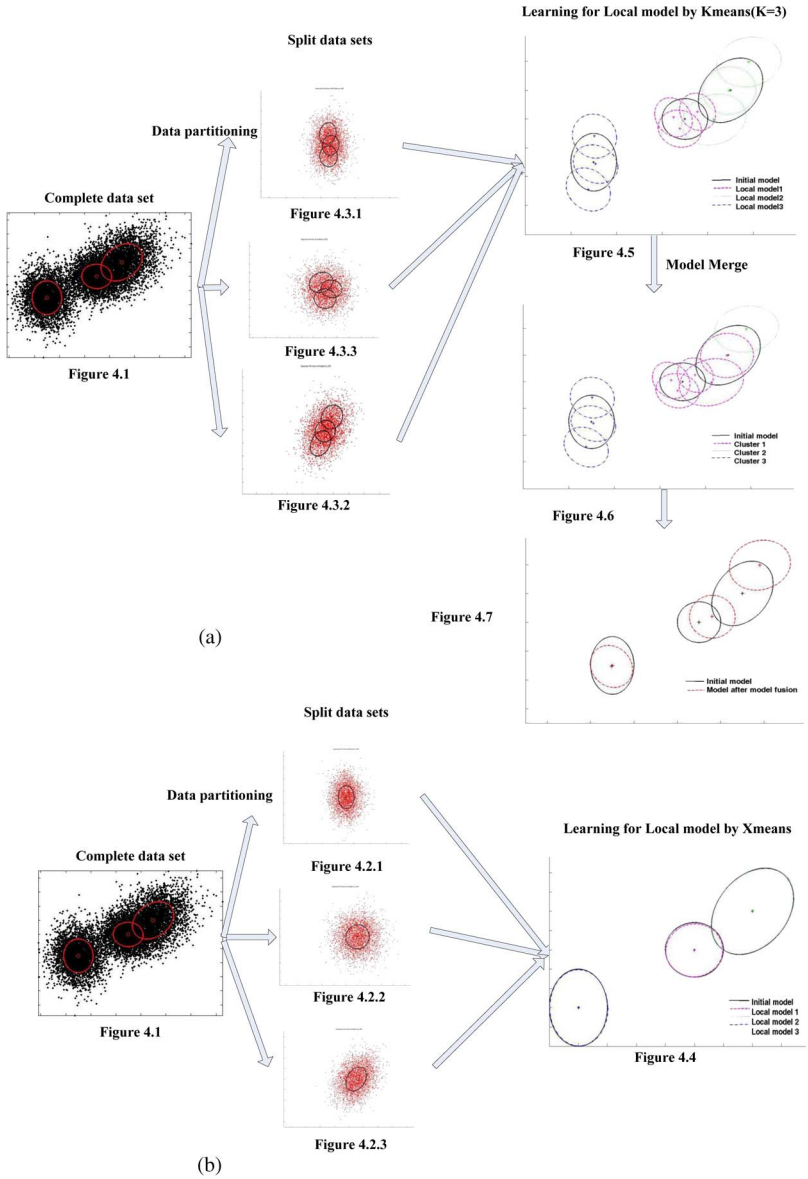


Fig. 4. Comparison between X-means and K-means.

AIC, we choose AIC as the criterion of our system, which is defined as follows:

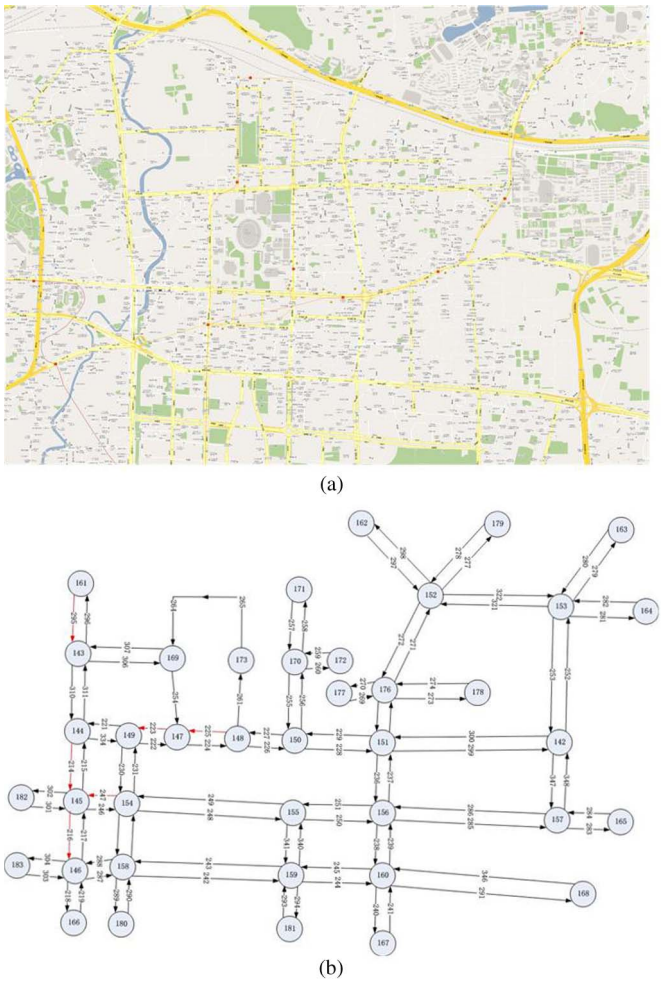
$$AIC = -2*Ln(\theta_k|X) + 2K. \tag{2}$$

Here, $L(\theta_k|X)$ is the likelihood of the model with the data set, and K is the number of parameters. In Fig. 4(b), it is shown that X-means obtained with AIC performs well. Its resulting clustering exactly captures the original three data blocks.

2. *Bayesian Network for Forecasting and One Directed Network in Guangzhou*

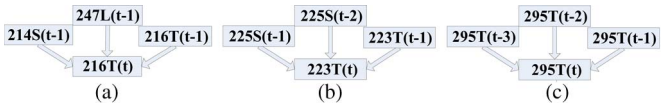
The traffic flow data used in this paper are the vehicle flow rate (in units of vehicles per hour) collected from the Tianhe area of Guangzhou. The data were collected every 5 min. The actual roadway network of this area is represented as a directed graph shown in Fig. 5. We then use the method given in [18] to model the relationship of traffic flows from adjacent intersections.

In the graph, links are classified into three types. The Bayesian networks for traffic flow forecasting about the three types of links are given in Fig. 6. As shown in Fig. 6, type-1 links have upstream links, like link 216. The Bayesian network of these links is shown in Fig. 6(a). The traffic flow at time t depends on both the traffic flow of upstream links and itself at time $(t-1)$. Fig. 6(b) shows a Bayesian network that includes the second type of links, i.e., a T-junction, like link 223. The third type is the link on the edge of the directed graph, such as link 295. The Bayesian network of these links is shown in Fig. 6(c). We can only forecast the traffic flow on this type of links at time based on the flow on those links from the previous time interval.



(a) (b)

Fig. 5. (a) Actual road network of the Tianhe area in Guangzhou. (b) Schematic of (a) with a directed graph.



(a) (b) (c)

Fig. 6. Bayesian networks of the links in graph. (S means the flow of through direction; L means the flow of left direction; T means the total flow).

C. Parameter Learning

The EM algorithm has been widely used to learn the parameters of the models. The EM algorithm has two phases, i.e., E-step and M-step. E-step is used to learn the degree to which a model fits data based on the old parameters of the model, whereas M-step is used to update the parameters of a model based on the adaptiveness of old parameters. Suppose that we have M models about the data set $X = \{x_1, \ldots, x_N\}$ and that the parameters of models are $\Theta_l = \{\mu_l, \sigma_l\}, l = 1, \ldots, M$. The detailed iterative formula is shown as follows:

Expectation step

$$P(l|x_i, \Theta) = \frac{P\left(x_i|\Theta_l^{old}\right) * \omega_l^{old}}{\sum_{j=1}^{M} P\left(x_i|\Theta_l^{old}\right) * \omega_l^{old}}. \tag{3}$$

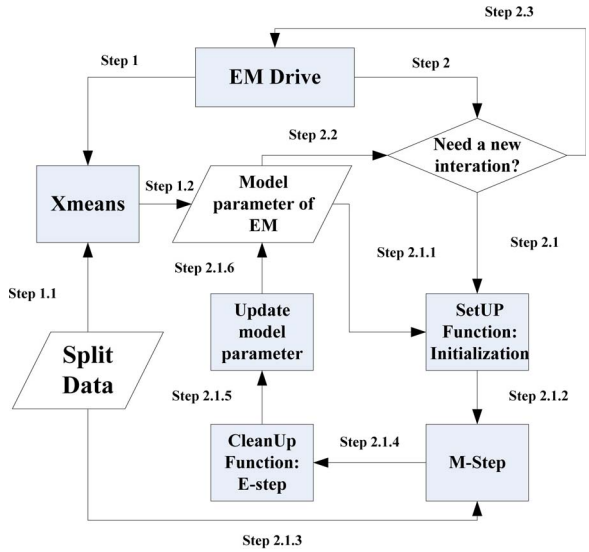


Fig. 7. Processing of EM based on MapReduce.

Maximization step

$$\omega_l^{new} = \frac{1}{N}\sum_{i=1}^{N} P(l|x_i, \Theta)$$

$$\mu_l^{new} = \frac{\sum_{i=1}^{N} x_i * P(l|x_i, \Theta)}{\sum_{i=1}^{N} P(l|x_i, \Theta)}$$

$$\sigma_l^{new} = \frac{\sum_{i=1}^{N} P(l|x_i, \Theta) * (x_i - \mu_l^{new}) * (x_i - \mu_l^{new})^T}{\sum_{i=1}^{N} P(l|x_i, \Theta)}. \tag{4}$$

Since model learning based on EM is an iterative process, with the successive update about the parameters of a model, users should give parameter δ to control the number of iterations. When the degree of improvement of degree falls below δ, the model is accepted, and the learning process will be terminated.

D. EM Based on MapReduce

The process of applying EM based on MapReduce is shown in Fig. 7. After the data partition phase, each Slavenode will have its split data, and the process of EM can start to work. In Step 1, for the high speed of the K-means algorithm, the EM driver will simultaneously call the K-means driver in the mahout toolkit several times with different parameters to realize the function of X-means [21]. This is the preprocessing of data. In addition, the outcome of X-means will be stored in the file of the model parameter of EM as the initialization for the whole process of iteration (see Step 1.2). Then, the EM driver calls the process of EM (see Step 2). Step 2.1, which has two phases, shows the process of an iteration of EM in the MapReduce framework. One phase is the map phase (see Step 2.1.1 to Step 2.1.4). The other is the reduce phase (see Steps 2.1.5 and 2.1.6). At the end of all iterations, which is determined by the number of iteration and the improvement degree of

model, the driver determines if an additional iteration is needed (see Step 2.2). If it is determined that the process needs to terminate, the return signal will return to the EM driver (see Step 2.3). The EM driver will call an extra job for global modeling.

In the following, we focus on the detail of the data partition phase, including the map phase and reduce phase:

Data Partition Phase: The task in the data partition phase has two parts: One is to extract reasonable training samples from raw data of the intersections. The other is to store these samples into different files in HDFS. The former is performed based on the relationship of the intersections in the Bayesian network, whereas the latter is performed based on three principles.

1) *One file only containing appropriate training samples for one direction at one intersection.* If data for different intersections are mixed in the same file, then the local model of this file will be meaningless for the learning process. The historical traffic flow data are thus stored by intersections. If, however, there is a logical relationship between the data in the Bayesian network, then the data will be stored in the same file. Moreover, the size of one file cannot be bigger than one block in HDFS, or it will be further divided into several chunks; if possible, each chunk will reside on a different Datanode. The default size of a block is 64 MB in hadoop (adjustable from 16 to 64 MB).
2) *The similarity about the size of files.* The whole data are divided into several subdata sets, and slaver nodes will learn local models from different subdata sets. Suppose that the computing capacity of computing nodes is similar, if the sizes of subdata sets are obviously different, the completion time of subtasks will substantially differ. In addition, the completion time of total task depends on the last completed subtasks. Therefore, the size of files should be similar. When the scale of computer clusters is big, the numbers of subtasks prefer to be the multiple of the number of idle computing nodes.
3) *Maximizing the ratio of computing time to total running time.* Different from single-point computing, the total running time of MapReduce computing has two parts: 1) preparation time and 2) computing time. In preparation time, the master will assign subtasks to idle nodes, and the algorithm will be copied to these nodes. In computing time, every computing node will perform a local model learning task. If the task is divided into too many small subtasks, the ratio of computing time to the total running time will decrease. Hence, the size of files cannot be very small.

According to these principles, we find that the size of traffic flow data files can neither be too large or too small. The size of files is closely related to the total amount of data and scale of computer clusters.

Map Phase: The map phase of one iteration has three subphases: 1) Setup phase; 2) M-step phase; and 3) Cleanup phase. In the Setup phase, the Mapper will read the parameters of EM for initialization. The parameters of EM contain the dimension

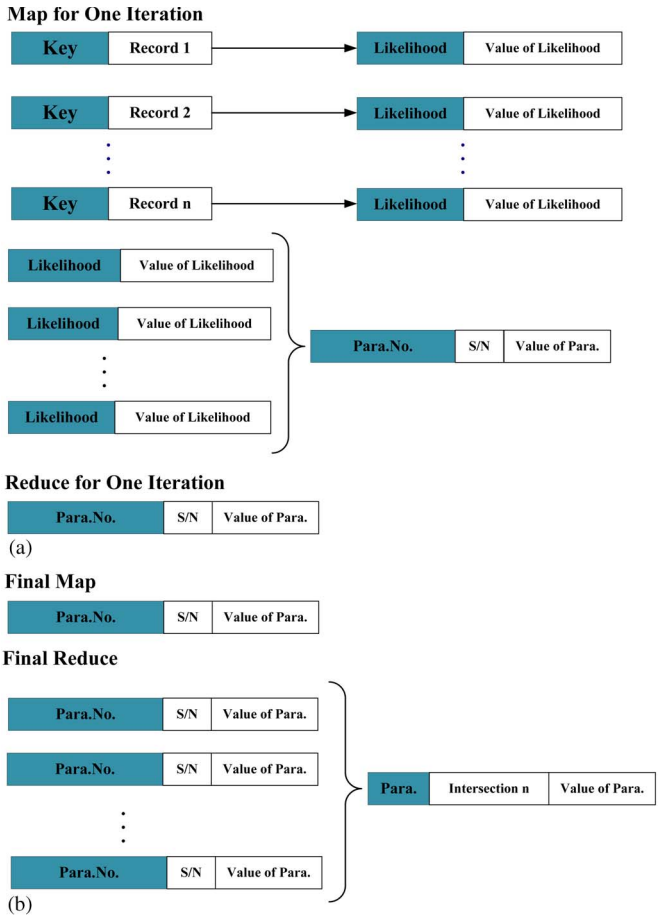


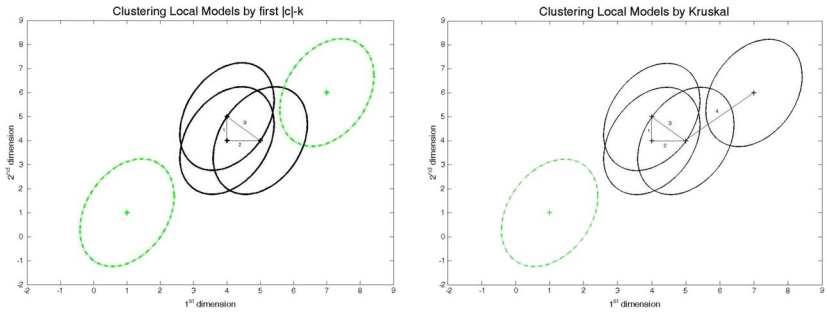
Fig. 8. Map phases and reduce phases of two types of MapReduce jobs. (a) Job of one Iteration. (b) Final job for global modeling.

of data and the old parameters of local models. In the M-step phase, as shown in Fig. 8(a), Mapper will read the training data line by line. Each record is transformed into a key value pair, whose value part is the context of the record. Reading data and processing data are alternately done. In addition, the training data and the outcome of M-step will be stored in memory, to be used as the input of the Cleanup phase. The outcome of the M-step phase is the likelihood of each record based on old parameters. In the Cleanup phase, based on the input, the parameters of local models will be updated by the operation in E-step. In addition, the parameters of local model will be stored in the form of key value pairs on local disk and wait for the collection of reducer.

When the parameter learning process of the local model is over, the driver will add an extra job as the final job. In the map phase, the task is only loading the parameters of a local model in the form of key value pairs to prepare for reducer.

Reduce Phase: During one iteration of the EM algorithm based on MapReduce, all the Mappers store updated parameters in different local disks. The tasks of Reducer are to collect them and update the file that has parameters of local models.

As shown in Fig. 8(b), using the model fusion technology, which will be discussed in the next section, the reduce phase of the additional job will merge the local model into a global model and store the parameters of the global model into a file for traffic flow forecasting.



(a) (b)

Fig. 9. Clustering outcome of (left) marking the first $|C|-K$ pairs and (right) marking the first $|C|-K+n$ pairs. (One class of local models are represented by a solid line; the other ones are represented by a dashed line.)

IV. Distributed Model Fusion

Based on the massive daily traffic flow data, we use EM to produce many local models about traffic flows at each intersection. A distributed model fusion approach is utilized to find out a reasonable global model for each intersection. This is the key point in the phase of reduce in Fig. 3. This process corresponds to the stage from Step 2.2.1 to Step 2.1.3 in Fig. 2. We first calculate the similarity of different local models and identify those that best describe the same traffic flow data. Finally, we merge local models according to their weights to get the global model.

A. Computation of Similarity of Local Models

The concept of similarity considered here is adopted from [18], which is defined as follows:

Definition 1 (Mutual Support): Let C_1 and C_2 be two local Gaussian models that are determined by a mean vector μ_i and a covariance matrix Σ_i. The mutual support of C_1 and C_2 is given as follows:

$$MS(C_1,C_2)=\int_{-\infty}^{+\infty} N_{\mu_1,\Sigma_1}(\overrightarrow{X})*N_{\mu_2,\Sigma_2}(\overrightarrow{X})\,d\overrightarrow{X}. \quad (5)$$

If two local Gaussian models overlap each other more, the mutual support of these will be larger. Then, we take another definition to reduce computational complexity, i.e.,

$$MS(C_1,C_2)=\sum_{i=1}^{d}\int_{-\infty}^{+\infty} N_{\mu_1^i,\Sigma_1^i}(\overrightarrow{X})*N_{\mu_2^i,\Sigma_2^i}(\overrightarrow{X})\,d\overrightarrow{X}. \quad (6)$$

This would enable us to use 1-D operation d times, instead of one d-dimensional operation. The computational complexity is reduced from $O(d^2)$ to $O(d)$. The detailed expression is given as follows:

$$\begin{aligned} &MS(C_1,C_2)\\ &=\sum_{i=1}^{d}\int_{-\infty}^{+\infty}\frac{1}{2*\pi*\sqrt{\left[\left(\sigma_1^i\right)^2+\left(\sigma_2^i\right)^2\right]*\left(\sigma^i\right)^2}}\\ &\quad\times\exp\left(-\frac{\left(\mu_1^i-\mu_2^i\right)^2}{2*\left[\left(\sigma_1^I\right)^2+\left(\sigma_2^i\right)^2\right]}+\frac{\left(x^i-\mu^i\right)^2}{2*\left(\sigma^i\right)^2}\right)d\overrightarrow{X^i}\\ &=\sum_{i=1}^{d}\frac{1}{\sqrt{2*\pi*\left[\left(\sigma_1^i\right)^2+\left(\sigma_2^i\right)^2\right]}}\\ &\quad\times\exp\left(-\frac{\left(\mu_1^i-\mu_2^i\right)^2}{2*\left[\left(\sigma_1^i\right)^2+\left(\sigma_2^i\right)^2\right]}\right) \end{aligned} \quad (7)$$

where

$$\mu^i=\frac{\mu_1^i*\left(\sigma_1^i\right)^2+\mu_2^i*\left(\sigma_2^i\right)^2}{\left(\sigma_1^i\right)^2+\left(\sigma_2^i\right)^2};\sigma^i=\sqrt{\frac{\left(\sigma_1^i\right)^2*\left(\sigma_2^i\right)^2}{\left(\sigma_1^i\right)^2+\left(\sigma_2^i\right)^2}}.$$

B. Cluster Local Models

After the mutual supports between local models are calculated, the task of clustering is performed. Here, we need to divide the local models into K clusters. In [20], the mutual supports are sorted by descending order, and the first $|C|-K$ pairs will be marked. ($|C|$ means the number of local models.) Then, the related local model will be marked in the same cluster. However, this way has two obvious deficiencies.

1) The first $|C|-K$ pairs should contain no cycles. If n cycles exist in these pairs, then the clustering of n local models will become undetermined. This can be illustrated with the example given in Fig. 9. In the figure, we need to group five Gaussian distributions into two clusters for $K=2$. A cycle is identified after the three middle pairs with the highest MC are clustered, i.e., $n=1$. The left one shows the outcome from the method given in [20], in which the first three $(|C|-K)$ local models are marked. The remaining two local models are grouped into one cluster. For the approach we propose, we look into the first four pairs $(|C|-K+n)$, and the outcome is shown on the right side in Fig. 9. Clearly, our approach is more reasonable since the distance between the local model on the top is closer to the cluster of the middle three submodels than to the local model at the bottom.
2) The computational complexity of ordering is high. If we do this ordering by bubble sort, the computational complexity is $O(|e|^2)$. $|e|$ is the number of pairs.

Step 1: Building empty edge set E_{max} whose capacity is K, Initial $I_{ClusterTag}$ =1 and Setting all the cluster tags of local models to $I_{ClusterTag}$.
Step 2: According to Dijkstra algorithm, we find out minimum spanning tree of these local models. During the search process, we will record the first K-longest edges.
Step 3: If E_{max} is not empty, set $I_{ClusterTag}$ = $I_{ClusterTag}$ +1 and go to Step 4; If E_{max} is empty, go to Step 7;
Step 4: Taking one edge away from E_{max} and Initial an empty node set N_{NewTag}. Setting the cluster tag of one point of this edge to $I_{ClusterTag}$. If this node is not a leaf node, then add it to N_{NewTag}.
Step 5: If N_{NewTag} is not empty, go to Step 6, else go to Step 3;
Step 6: If any and one point of edge e exists in N_{NewTag}, we should set the cluster tag of the other point of this edge to $I_{ClusterTag}$. After this operation, we judge whether it is a leaf node. If it is, we add it to N_{NewTag}. Then go to Step 5;
Step 7: According to the cluster tags of local models, we can divide them into K clusters.

Fig. 10. Global clustering algorithm based on Dijkstra [19].

For these two deficiencies, we use Dijkstra [17] to cluster local models. We first introduce a definition for characterizing the similarity between two local models.

Definition 2 (Distance Between Local Models): Let C_1 and C_2 be two local models; then, the distance between C_1 and C_2 is described as

$$DLM(C_1, C_2) = \frac{1}{MS(C_1, C_2)}. \tag{8}$$

If two local models are closer, the degree of similarity about two local models is higher. The problem of clustering local models is then converted to a minimum spanning tree problem. Accordingly, we use the Kruskal algorithm to find a minimum spanning tree, and the computational complexity is reduced to $O(|e| \log |e|)$. Then, we just cut off the first K-longest edges to group local models into K clusters. The detailed algorithm is given in Fig. 10.

C. Model Fusion

After dividing local models into K clusters, we need to get K global models C_i based on the weights, mean vectors, and covariance matrixes of local models C_i. The representations of weights, mean vectors, and covariance matrixes of local models are $\omega_{\mathrm{LM}} = \{\omega_1, \ldots, \omega_n\}$, $\mu_{\mathrm{LM}} = \{\mu_1, \ldots, \mu_n\}$, and $\sigma_{\mathrm{LM}} = \{\sigma_1, \ldots, \sigma_n\}$, respectively. We define a function $J(C_i, k)$ as follows to judge which cluster model C_i belongs to

$$J(C_i, k) = \begin{cases} 1, C_i \in C_k \\ 0, C_i \notin C_k \end{cases} \tag{9}$$

In addition, the number of samples in the file that the local model C_i is learning from is defined as $\rho(C_i)$. When we update the online global model based on newly obtained traffic flow data, we take the global model as a local model to merge with new local models. Therefore, when the sizes of samples used in the learning process for local models are different, the weights, mean vectors, and covariance matrixes of the global model are defined as follows:

$$\begin{aligned} \omega_k &= \frac{\sum_{i=1}^{n} \omega_i \cdot \rho(C_i) \cdot J(C_i, k)}{\sum_{i=1}^{n} \omega_i \cdot \rho(C_i)} \\ \mu_k &= \frac{\sum_{i=1}^{n} \omega_i \cdot \rho(C_i) \cdot \mu_k \cdot J(C_i, k)}{\sum_{i=1}^{n} \omega_i \cdot \rho(C_i) \cdot J(C_i, k)} \\ \sigma_k &= \frac{\sum_{i=1}^{n} \omega_i \cdot \rho(C_i) \cdot \sigma_k \cdot J(C_i, k)}{\sum_{i=1}^{n} \omega_i \cdot \rho(C_i) \cdot J(C_i, k)}. \end{aligned} \tag{10}$$

If the sizes of samples used in the learning process for local models are similar, the weights, mean vectors, and covariance matrices of the global model are simplified as follows:

$$\begin{aligned} \omega_k &= \frac{\sum_{i=1}^{n} \omega_i \cdot J(C_i, k)}{\sum_{i=1}^{n} \omega_i} \\ \mu_k &= \frac{\sum_{i=1}^{n} \omega_i \cdot \mu_i \cdot J(C_i, k)}{\sum_{i=1}^{n} \omega_i \cdot J(C_i, k)} \\ \sigma_k &= \frac{\sum_{i=1}^{n} \omega_i \cdot \sigma_i \cdot J(C_i, k)}{\sum_{i=1}^{n} \omega_i \cdot J(C_i, k)}. \end{aligned} \tag{11}$$

The comparison between distributed model fusion and the modeling in standalone mode is shown in Fig. 11. Fig. 11.1 shows the origin mixture model and the data set produced by it. In addition, the distribution of the origin mixture model is represented by the solid line in Fig. 11.4 and 11.5. For distributed learning, the data set that contains 1000 data points is split into three parts, as shown in Fig. 11.2.1–11.2.3. Then, three local models are learning from these three split data sets in Fig. 11.4. Finally, the three local models are merged to obtain a single global model, whose distribution is shown in Fig. 11.5. For comparison, the complete data set is also learning by EM in standalone mode to get the model whose distribution is also shown in Fig. 11.5. We can see that the global model is slightly worse than the model based on the complete data set. However, it indeed has fixed the error of some individual local models, like Local model 1. Hence, this approach is considered as acceptable.

V. Experiments

In this section, we present the test results of an experimental analysis conducted with our traffic flow forecasting system. The largest real-world application of hadoop uses a total of 4500 computers in one computing cluster (Yahoo's cloud computing platform) [14]. Our experiment is aimed at finding the best load of one slave server to handle the learning process of traffic flow data in the computing cluster. It is important to determine the number of required computers in the computing cluster in an ideal situation since it is directly connected with Step 1.2 in Fig. 2 and the data partition phase in Fig. 3. Random allocation of workload would lead to a heavy processing burden in some slave nodes. The load imbalance can prolong the entire computation time by some individual slave nodes. In addition, this analysis helps us to optimize the cost effectiveness of the cloud computing platform. Our computing cluster consists of

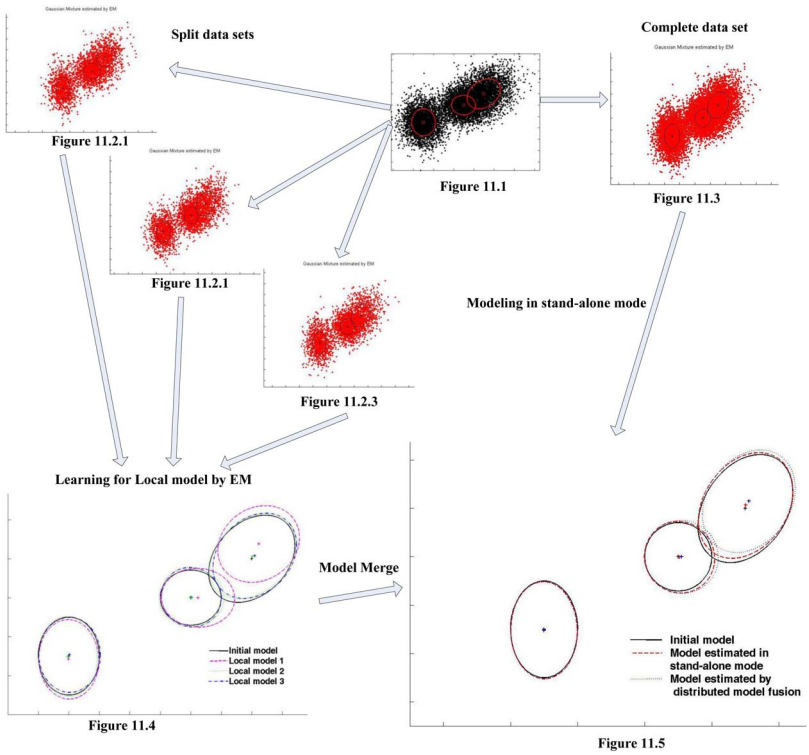


Fig. 11. Comparison between distributed model fusion and modeling in standalone mode.

TABLE I
CONFIGURATION OF FIVE NODES IN THE COMPUTING CLUSTER

IP of node	Role in cluster	CPU	Memory
192.168.0.238	Master and Namenode	INTEL Quad-core 2.66GHZ	2Gbyte
192.168.0.239	Slave and Datanode	INTEL Quad-core 2.66GHZ	2Gbyte
192.168.0.239	Slave and Datanode	INTEL Quad-core 2.66GHZ	2Gbyte
192.168.0.240	Slave and Datanode	INTEL Dual-core 2.0GHZ	1Gbyte
192.168.0.168	Slave and Datanode	INTEL Quad-core 2.66GHZ	2Gbyte

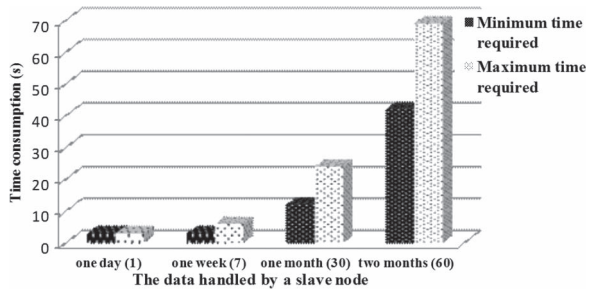


Fig. 12. Time consumption for different sizes of data sets.

five computers or five nodes. The configuration of the five nodes is given in Table I.

Since the number of iterations of EM depends on the initial parameters and the degree of improvement in each iteration, we only analyze the time needed for a single iteration. To start with, the whole data set is partitioned into files for the system. The split files for every experiment are different and are then aligned to the slavers by the master. The time required for handling different sizes of data sets is shown in Fig. 12. As the minimum time required is the processing time needed by the fastest computer in the cluster to finish one subtask, the maximum time required reflects the processing speed of the slowest one in the cluster. As mentioned in Section II, the time required for processing data is made up of two parts: 1) computation time and 2) transmission time. The former is essentially used for data processing, whereas the latter is mainly consumed by transmission of codes and results. Therefore, when a split data set is small, the portion of computing time in the total running time is small. As a result, the time required to complete a subtask does not depend on the processing speed of nodes. Therefore, the minimum and maximum time required to process a one-day traffic flow data set is similar. However, once the data set contains more than one-week data, the computation time takes up a large portion of the total running time. The granularity of subtasks should thus be the multiple of one-week data set. However, for the convenience of load balancing for this system, the multiple should not be too large.

We consider here the processing time used per kilobyte data in our computing cluster for different sizes of split data sets. The outcome is shown in Fig. 13. With the increase in the size of the split data set, the processing time per kilobyte data gradually drops in the beginning and then increases again. This can be explained by looking into the transmission and computation time required for different sizes of the split data set. Initially, the size of the split data set handled by slave nodes is small in scale. The transmission time is dominant over other factors in the total processing time. The rate of its increase, however, is lower than the increase in the data size. Hence, the processing time per kilobyte data on slave nodes gradually drops. As the size of

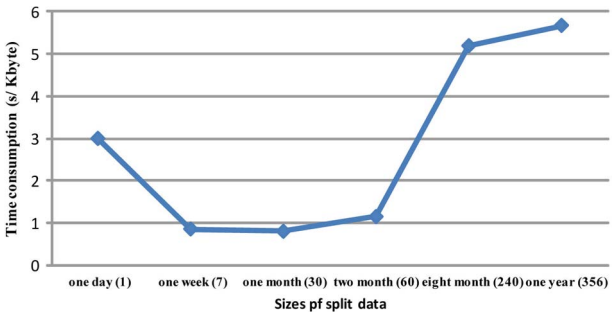


Fig. 13. Time consumption per kilobyte data on different sizes of split data sets in computing cluster.

the split data set continues to increase, the computation time becomes dominant in the total processing time. The step for computing the expectation in EM will substantially increase the memory usage since it is required to store the likelihood of each data set into memory. This will increase the computation time, which, in turn, will increase the processing time per kilobyte data on slave nodes in every cluster. From Fig. 13, we can see that, in terms of the processing time per kilobyte data, the use of one-month data set is the best choice of the split data set size for our computing cluster.

For different sizes of subtasks, the computation time for each subtask in a cluster can be obtained. Suppose that the maximum number of iterations in the total iterative process of EM is 200 and that we require that our computing cluster is large enough to guarantee that one node only needs to handle one subtask in each step. The computation time required is no more than 70 min for any size of traffic flow data set.

VI. Conclusion and Further Work

In this paper, we have discussed how the MapReduce approach can be employed to handle distributed modeling in a traffic flow forecasting system. A MapReduce framework has been designed to provide an effective way to process large-scale data sets in a data-driven traffic flow forecasting system, which can be viewed as a subsystem in agent-based intelligent traffic clouds [8]. The proposed MapReduce framework deals with the key issues in distributed processing, distributed local model learning, and model fusion. We have also identified the best granularity of data split for such a system.

Further improvements over some of the details of our system can be made. For example, the performance of the EM algorithm based on MapReduce can be enhanced by reducing memory consumption and using the multicore program. Work is ongoing that will further optimize our system for various system components.

Acknowledgment

The authors would like to thank Prof. F.-Y. Wang for his guidance and encouragement and the other students who contributed to this work at the State Key Laboratory of Management and Control for Complex Systems, Chinese Academy of Sciences.

References

[1] F.-Y. Wang, "Toward a revolution in transportation operations: AI for complex systems," *IEEE Intell. Syst.*, vol. 23, no. 6, pp. 8–13, Nov./Dec. 2008.
[2] F.-Y. Wang and S. Tang, "Artificial societies for integrated and sustainable development of metropolitan systems," *IEEE Intell. Syst.*, vol. 19, no. 4, pp. 82–87, Nov./Dec. 2004.
[3] B. M. William, "Modeling and forecasting vehicular traffic flow as a seasonal stochastic time series process," Ph.D. dissertation, Dept. Civil Eng., Univ.Virginia, Charlottesville, VA, 1999.
[4] I. Okutani and Y. J. Stephanedes, "Dynamic prediction of traffic volume through Kalman filter theory," *Transp. Res., Part B, Methodol.*, vol. 18B, no. 1, pp. 1–11, Feb. 1984.
[5] J. Hall and P. Mars, "The limitations of artificial neural networks for traffic prediction," in *Proc. 3rd IEEE Symp. Comput. Commun.*, Athens, Greece, 1998, pp. 8–12.
[6] H. B. Yin, S. C. Wong, J. M. Xu, and C. K. Wong, "Urban traffic flow prediction using a fuzzy-neural approach," *Transp. Res., Part C, Emerging Technol.*, vol. 10, no. 2, pp. 85–98, Apr. 2002.
[7] L. Li, W.-H. Lin, and H. Liu, "Type-2 fuzzy logic approach for short-term traffic forecasting," *Proc. Inst. Elect. Eng.—Intell. Transp. Syst.*, vol. 153, no. 1, pp. 33–40, Mar. 2006.
[8] Z. Li, C. Chen, and K. Wang, "Cloud computing for agent-based urban transportation systems," *IEEE Intell. Syst.*, vol. 26, no. 1, pp. 73–79, Jan./Feb. 2011.
[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. OSDI*, San Francisco, CA, Dec. 2004, pp. 137–149.
[10] M. Gaggero, S. Leo, S. Manca, F. Santoni, O. Schiaratura, and G. Zanetti, "Parallelizing bioinformatics applications with MapReduce," in *Proc. CCA*, 2008, pp. 1–6.
[11] J. Lin and C. Dyer, *Data-Intensive Text Processing With MapReduce*. San Rafael, CA: Morgan & Claypool, Apr. 30, 2010.
[12] J. Cohen, "Graph twiddling in a MapReduce world," *Comput. Sci. Eng.*, vol. 11, no. 4, pp. 29–41, Jul. 2009.
[13] F.-Y. Wang, "Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 3, pp. 630–638, Sep. 2010.
[14] [Online]. Available: http://hadoop.apache.org/
[15] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *Proc. NIPS*, 2006, pp. 281–288.
[16] D. Pelleg and A. Moore, "X-means: Extending K-means with efficient estimation of the number of clusters," in *Proc. 17th Int. Conf. Mach. Learn.*, 2000, pp. 727–734.
[17] D. Pelleg and A. Moore, "A new look at the statistical model identification," *IEEE Trans. Autom. Control*, vol. AC-19, no. 6, pp. 716–723, Dec. 1974.
[18] S. Sun, C. Zhang, and G. Yu, "A Bayesian network approach to traffic flow forecasting," *IEEE Trans. Intell. Transp. Syst.*, vol. 7, no. 1, pp. 124–132, Mar. 2006.
[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[20] H.-P. Kriegel, P. Kröger, A. Pryakhin, and M. Schubert, "Effective and efficient distributed model-based clustering," in *Proc. IEEE Int. Conf. Data Mining*, 2005, pp. 1–8.
[21] [Online]. Available: http://mahout.apache.org/
[22] X. Min, J. Hu, Q. Chen, T. Zhang, and Y. Zhang, "Short-term traffic flow forecasting of urban network based on dynamic STARIMA model," in *Proc. Int. IEEE Conf. Intell. Transp. Syst.*, 2009, pp. 1–6.

Cheng Chen received the B.Eng. degree in measuring and control technology and instrumentation from HuaZhong University of Science and Technology, Wuhan, China, in 2008. He is currently working toward the Ph.D. degree in control theory and control engineering with the State Key Laboratory of Management and Control for Complex Systems, Chinese Academy of Sciences, Beijing, China.

His research interests include multiagent systems and distributed artificial intelligence and its applications.

Zhong Liu received the Ph.D. degree from the National University of Defense Technology.

He is currently with the National University of Defense Technology, Changsha, China, as a Professor, Deputy Director of the Science and Technology on Information Systems Engineering Laboratory, and Senior Advisor for the Research Center for Computational Experiments and Parallel Systems. His research interests include planning systems, computational organization, and intelligent systems.

Wei-Hua Lin received the Ph.D. degree from the University of California, Berkeley.

He was a Postdoctoral Researcher with the California Partners for Advanced Transit and Highways program, University of California, Berkeley, for two years. He is currently an Associate Professor with the Department of Systems and Industrial Engineering, University of Arizona, Tucson. Prior to that, he was with the Virginia Polytechnic Institute and State University, Blacksburg. His research interests are optimization in intelligent transportation systems, logistics and distribution systems, and transportation network analysis.

Dr. Lin is a member of the Intelligent Transportation Systems Committee of the Transportation Research Board, National Research Council of the United States.

Shuangshuang Li received the B.Eng. degree in automation from Hangzhou Dianzi University, Hangzhou, China, and the M.Eng. degree in control theory and control engineering from Zhejiang University, Hangzhou, in 2007 and 2010, respectively. He is currently working toward the Ph.D. degree in technology of computer applications with the State Key Laboratory of Management and Control for Complex Systems, Chinese Academy of Sciences, Beijing, China.

His research interests include recommendation of multiagent control systems, web search, and data mining.

Kai Wang received the B.Eng. degree from the National University of Defense Technology, Changsha, China, in 2007. He is currently working toward the Ph.D. degree in technology of computer applications with the Center for Military Computational Experiments and Parallel Systems Technology, College of Mechatronics Engineering and Automation, National University of Defense Technology, and the State Key Laboratory of Intelligent Control and Management of Complex Systems, Chinese Academy of Sciences, Beijing, China.

His research interests include the control and management of complex systems, as well as supercomputing.