Online Credit Card Fraud Detection: A Hybrid Framework with Big Data Technologies

You Dai*, Jin Yan*, Xiaoxin Tang*, Han Zhao† and Minyi Guo*
*Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
†School of Computer Science & Technology, Huazhong University of Science and Technology, China
Email: *{dywywh, yanjin, tang.xiaoxin, myguo}@sjtu.edu.cn, †zhaohan.miven@gmail.com

***Abstract*—In this paper, we focus on designing an online credit card fraud detection framework with big data technologies, by which we want to achieve three major goals: 1) the ability to fuse multiple detection models to improve accuracy; 2) the ability to process large amount of data and 3) the ability to do the detection in real time. To accomplish that, we propose a general workflow, which satisfies most design ideas of current credit card fraud detection systems. We further implement the workflow with a new framework which consists of four layers: distributed storage layer, batch training layer, key-value sharing layer and streaming detection layer. With the four layers, we are able to support massive trading data storage, fast detection model training, quick model data sharing and real-time online fraud detection, respectively. We implement it with latest big data technologies like Hadoop, Spark, Storm, HBase, etc. A prototype is implemented and tested with a synthetic dataset, which shows great potentials of achieving the above goals.**

***Index Terms*—Online Credit Card Fraud Detection Framework, Big Data, Model Fusion, Hadoop, Spark, Storm, HBase**

I. INTRODUCTION

With the rapid development of Internet and E-commerce, online payment has become one of the most important ways for trading. Credit card, for its convenient usage online, has gotten an explosive growth in recent years. According to the 2013 Federal Reserve Payments Study, the total number of credit card transactions in the U.S. was 26.2 billion in 2012, up from 21 billion in 2009. Seven of the largest card issuers – American Express, JP Morgan, Capital One, Bank of America, Citigroup, Discover and U.S. Bancorp – reported more than $490 billion in total credit card payments made in the fourth quarter of 2014 alone. In Amazon, net sales increased 20% to $107.0 billion, compared with $89.0 billion in 2014.

However, the increasing amount of online trading could also attract criminal activities. In online trading, the credit cards are usually used as virtual card[1]. An attacker only needs to obtain few important information of the card (e.g., card ID, secure code) to make a fraudulent transaction on the Internet while the genuine cardholder often does not notice that his card information has been leaked, which may cause a significant financial loss both to the cardholder and credit card company. In the past decades, financial companies and researchers have developed many Credit Card Fraud Detection Systems (CCFDS).

Although, the main challenge for most CCFDS is how to improve detection accuracy, the computational capacity of CCFDS have become more and more important with the explosive growth of trading data. The growing number of users and payment transactions has brought heavy workloads to these systems. The speed of new transactions coming into the system can reach millions per second while the size of stored historical transactions can reach several PBs or even EBs. In this case, processing detection tasks and model training on so many incoming transactions with a low delay is very hard for most traditional systems.

According to the recent trends, Big Data technology seems to be the key of solving the challenge of computational capacity. For example, before using Big Data technology, the Visa company could only analyze 2% of its historical transactions, and make one update of its detection model every 2 or 3 days. However, with the help of Hadoop, their new CCFDS can now analyze the whole historical transactions in time while support 16 models doing detection concurrently and make updates of these models every 1 to 2 hours.

However, the above case only considers the traditional scenarios in which physical cards are used more frequently than virtual cards. In E-commerce, virtual cards are used more frequently than physical cards, which makes the payment much easier and creates many small but frequent transactions. Thus, simply applying single Big Data tool can not solve the real challenge. In this paper, we try to address this challenge through a hybrid solution and have made the following three major contributions:

- We propose a credit card fraud detection workflow, which can fuse different detection models to improve accuracy. It contains most of the common design ideas in latest C-CFDSs, which make it much easier to integrate detection algorithms into the workflow;
- We design a four-layer framework which includes distributed storage layer, batch training layer, key-value sharing layer and streaming detection layer, to support massive trading data storage, fast detection model training, quick model data sharing and real-time online fraud detection, respectively;
- We implement the framework with the latest big data technologies like Hadoop, Spark, Storm, HBase, etc. With these technologies, we are able to handle the burst amount of data and build a scalable and reliable system. Experimental results show that this system has the potential to achieve a sustainable performance.

2324-9013/16 $31.00 © 2016 IEEE
DOI 10.1109/TrustCom.2016.251

IEEE computer society

This paper is organized as follows: Section II introduces the related work of this paper; Section III gives the design of our proposed fraud detection workflow; Section IV describes our framework for real-time online fraud detection; Section V shows how we implement the framework with the latest Big Data technologies; Section VI gives the experimental results of our framework; Finally, Section VII concludes this paper and sheds light on our future work.

II. RELATED WORK

A. Fraud Detection Algorithms

Fraud detection has drawn a lot of researchers' interest in the past few years and many new algorithms have been developed[2]. Most algorithms can be divided into two categories: supervised algorithms which use labeled training data, and unsupervised algorithms, using unlabeled training data[3].

Popular supervised algorithms include neural network, logistic regression models, etc. Ghosh and Reigh[4] use a three-layer neural network to detect fraud transactions. Syeda[5] proposes a parallel granular neural network to speed up process. Logistic regression models are often used as components in CCFDS[6], [7].

Compared with supervised algorithms discussed above, unsupervised approaches are also widely used in CCFDS. Bolton and Hand [8] use peer group analysis to monitor behavioral fraud. They also recommend break point analysis to identify changes in spending behavior. Abhinav et al[1] use hidden Markov model(HMM) to detect fraud transactions. If the incoming transaction does not have a rather higher transferring probability, it will be regarded as fraud.

Instead of developing new algorithms to get a higher accuracy, we try to achieve that goal by fusing existing algorithms. Each algorithm may have their own advantages in detecting certain kinds of frauds. If they can be integrated into our system, we should have a better chance to detect most frauds.

B. Model Fusion Methods

The idea of fusing different algorithms is not new. Many researchers use meta-learning to create hybrid CCFDS. Peter et al[9] use an evolution-fuzzy detection system to classify suspicious and non-suspicious credit card transactions. This system is composed by a genetic programming (GP) search algorithm and a fuzzy expert system. It first clusters data into three groups, namely 'high', 'medium' and 'low'. GP part evolves a series of variable-length rules, which characterize the differences of data. They conclude that the use of evolution with fuzzy logic has higher accuracy.

Panigrahi et al[10] propose a fusion system using Dempster-Shafer theory and Bayesian learning. Their system consists of four components: rule-based filter, Dempster-Shafer adder, transaction history database and Bayesian learner. Rule-based filters first generates a suspicious score. If it can't determine whether an incoming transaction is fraud or not, Bayesian model will further detect the transaction, and give the final result.

Krishna and Lata[11] combine BLAST and SSAHA algorithms to do credit card fraud detection. These two algorithms are efficient sequence alignment approaches. In this two-stage system, a profile analyzer (PA) first determines the similarity of an incoming transaction with the past genuine spending sequence. Then the fraud transactions are passed to a subsequent deviation analyzer (DA) for possible alignment.

Our work concentrates on solving performance challenges of CCFDS with Big Data technologies. Due to a limited computational resources, the past systems can only fuse limited algorithms. But Big Data technologies provide an easy and scalable way to fuse more detection algorithms.

C. Big Data Technologies

One of the biggest challenge for fraud detection systems is the tremendous growing amount of transactions. Current fraud detection systems need to be more effective and scalable in order to handle such large amount of incoming data. Hence, using Big Data technology is the best solution for this problem.

Many Big Data platforms are released to store and process data in recent years. The MapReduce framework was proposed in 2004[12]. Apache Hadoop[1] is presented as the most popular open-source implementation of MapReduce and DFS for large-scale data processing and storing. However, Hadoop has a poor performance on iterative and online computing. Apache Spark[2] allows users to persist the data in memory and is the most popular batch processing platform for iterative computing. Storm[3] is the most widely used real-time streaming processing system. Storm's applications are submitted as topologies. These topologies usually contain two components, which are called spout and bolt. Spout is the source of streams in topology. It reads tuples from external source and sends them into the topology. Bolt processes the data once a tuple. HBase[4] is an open source distributed key-value store developed on top of the distributed storage system HDFS.

Many researchers have used these Big Data technologies to improve the performance of applications. Zhao and Shang [13] implement user-based collaborative-filtering recommendation on MapReduce to solve the scalability problem. Wu et al[14] considers the problem of doing data mining with Big Data technology. Jacinto Arias et al[15] apply MapReduce to implement a scalable learning of K-dependence Bayesian Classifiers. Sergio Ramrez-Gallego et al[16] develop a distributed entropy minimization discretizer for Big Data Analysis with Apache Spark.

In this paper, we propose a hybrid framework with Big Data technologies to solve performance challenges faced by online CCFDS. As a real time system, we not only need to consider the performance issues during data storing, model training, data sharing and fraud detection, but also take care of the integration problems of them since any slow component could become a bottleneck of the whole system.

[1] http://hadoop.apache.org/
[2] https://spark.apache.org/
[3] http://storm.apache.org/
[4] http://hbase.apache.org/

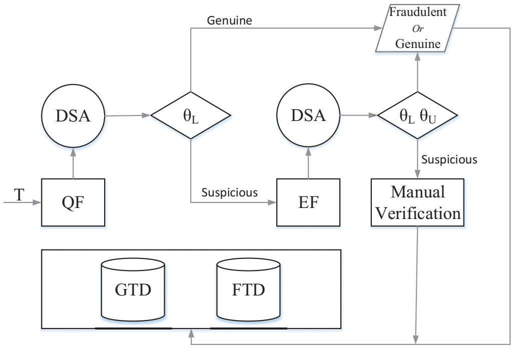

Fig. 1: Credit card fraud detection workflow

III. Fraud Detection Workflow

In this section, we illustrate our proposed credit card fraud detection workflow. The goal is to fuse different detection models and algorithms together. Thus it should assimilate main design ideas of CCFDS in recent years.

A. Workflow Components

Table I shows the meaning of all variables used in this section. Figure 1 shows the proposed workflow. Given an incoming transaction T, it will be processed through the following components:

1) Quick Filter (QF): QF consists of many unsupervised algorithms for quick fraud detection. Each algorithm trains models for every cardholder to capture their consumption patterns by using their own historical transactions. A fraud score $f_i^Q(t)$, which represents the probability of whether this transaction is fraudulent or not, can be calculated based on the models. Typical algorithms include DBSCAN[10], HMM[1], Self-Organizing Map(SOM)[17];
2) Dempster-Shafer Adder (DSA): DSA combines different fraud scores, then generates a merged result. For example, if there are two fraud scores $f_1(t)$ and $f_2(t)$, DSA will generate a merged score $f(t)$ as follows:

$$f(t) = f_1(t) \oplus f_2(t) = \frac{\sum_{x \bigcap y=t} m_1(x) * m_2(y)}{1 - \sum_{x \bigcap y=\phi} m_1(x) * m_2(y)}$$

3) θ_L and θ_U: they are two threshold values where θ_L is the upper bound of genuine and θ_U is the lower bound of fraudulent. When $f(t) < \theta_L$, it will be regarded as genuine transaction. When $f(t) > \theta_U$, it is fraud. When $\theta_L < f(t) < \theta_U$, the transaction is suspicious (non-determined);
4) Explicit Filter (EF): EF contains several supervised algorithms to further decide whether a transaction is fraudulent or not. Each algorithm learns the fraud pattern by training with all historical transactions. Then the fraud score $f_i^E(t)$ can be calculated for an incoming transaction based on the fraud pattern. Typical algorithms include Logistic Regression[7], Decision Tree[18], Naive Bayes[7] and Neural Networks[7], [5];

TABLE I: Variable list

Variable	meaning
T	incoming transaction
θ_L	upper bound of genuine
θ_U	lower bound of fraudulent
$f_i^Q(t)$	fraud score of t in ith quick filter
$f_i^E(t)$	fraud score of t in ith explicit filter

5) Manual verification: in case of needs for more accurate results, manual verification could be added into the system. Those suspicious transaction can be sent here to be further checked;
6) Historical transaction database: there are two types of database to store historical transactions. One is Genuine Transactions Data (GTD), which stores all genuine historical transactions. The other one is Fraudulent Transactions Data (FTD), which stores all fraudulent historical transactions.

B. Design Ideas

This workflow contains many design ideas of the latest CCFDSs:

1) Fuse different algorithms for higher accuracy[9], [10], [11]: Apply Dempster-Shafer Theory to aggregate fraud scores can provide higher accuracy[10]. In the proposed workflow, both QF and EF consist of several different algorithms and we use two DSAs to merge the fraud scores. In this way, any other efficient detection algorithm could also be fused into our workflow;
2) Combine both supervised and unsupervised approaches for a better cover of fraud types: In unsupervised fraud detection, if a new transaction does not fit the past behavior model, it is considered as potentially fraudulent. Thus, unsupervised methods have the ability to detect undiscovered types of fraud but they may also have many false alarms[19]. On the other hand, supervised approaches can only be used to detect frauds of a type that have previously occurred but its accuracy could be higher[20]. Thus, a combination of them could lead to a better cover of most types of fraud;
3) Combine quick filter and explicit filter for better performance : since QF detection only involves the behavior model of each cardholder, its computational performance is much higher than that of EF detection which involves the whole model built with all historical data. By using a faster filter first to find fewer suspicious transactions and then using a accurate filter to double confirm the fraud, the system can achieve a better performance without hurting the accuracy.

However, the current workflow does the model training in offline. A better design should be able to train models used in detection with the latest data and then update them frequently. Thus, we need to consider batch training and model sharing during implementation.

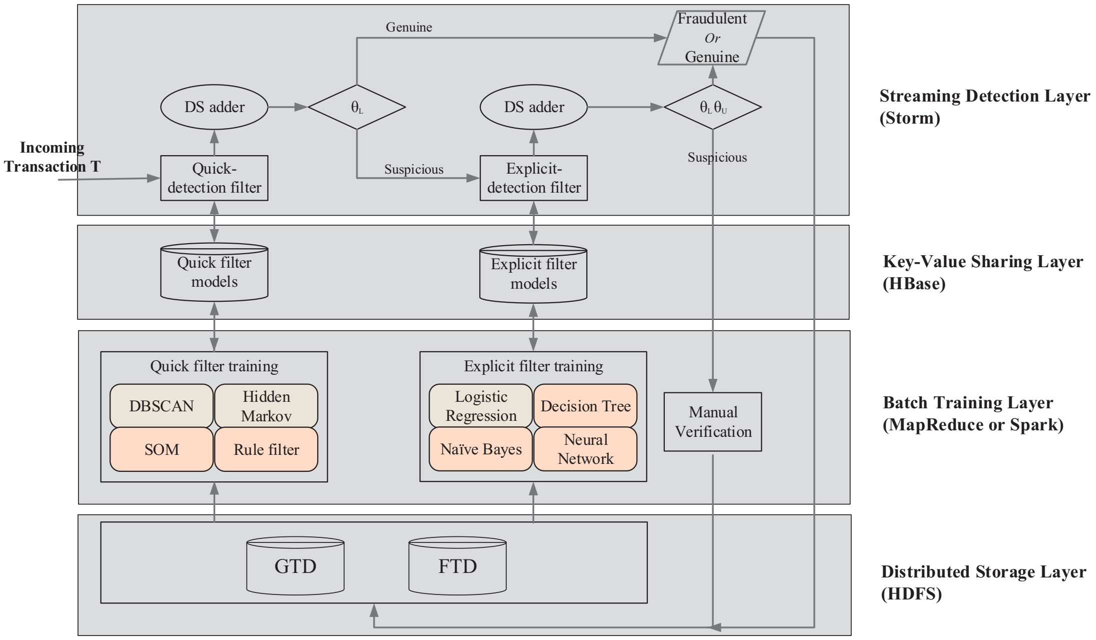

Fig. 2: Hybrid framework for CCFDS workflow.

IV. Hybrid Framework

The explosion of users and payment transactions brings several challenges to online CCFDS: performance, fault tolerance, system scalability, etc. Each of them would be far beyond the research scopes of fraud detection. Luckily, with the support of Big Data technologies, we can achieve the above goals with much less effort. But a proper design is certainly the key to success. In this case, we propose a hybrid framework to represent our previous workflow properly, which is shown in Figure 2. The whole framework consists of four layers. We will analyze the performance challenges in each layer and then choose appropriate Big Data tools to solve them.

A. Distributed Storage Layer

This layer is responsible for storing both Genuine Transaction Data (GTD) and Fraudulent Transaction Data (FTD). Since the transaction data is increasing very fast, this layer has to offer enough storage capacity to store them. Meanwhile, fault-tolerance is important since credit card transaction data is very valuable. Once the data is stored, it will be seldom changed but might be visited a lot. Thus the IO access pattern of this layer is a write-once, read-many-times pattern. We find that HDFS is very suitable for this layer since it is fault tolerant, scalable, and has a high read-write throughput.

B. Batch Training Layer

This layer is responsible for fast training of detection models. As described in previous section, this layer needs to train two types of models which are contained in QF and EF respectively. Models in QF are prepared for each cardholder. Thus, it should have the ability of training models separately. On the other hand, models in EF involves all historical data. Thus, it needs to iteratively train the models. Based on these requirements, we find that Hadoop is best suited for QF model training as we can assign each map and reduce task to process one cardholder's transaction data. Spark should be a good candidate for EF model training since it has very high performance in dealing with iterative tasks.

C. Key-value Sharing Layer

This layer is responsible for quick sharing of all model data. As in the training stage, both Hadoop and Spark need to frequently do sequential write and update operations to the models. In the detection stage, these models also need to be visited frequently. Thus, it is very important to have a data sharing layer for both model training and fraud detection. We find that HBase is a proper candidate for this task since its key-value storage supports very fast and scalable data sharing.

D. Streaming Detection Layer

This layer is responsible for processing the detection in real-time. Since the payment transactions are coming continuously and rapidly, the system has to process them on time. Otherwise there would be more and more un-processed data in the system. Thus there are two constraints here: scalability and low delay. Spark Streaming and Storm are two popular scalable streaming processing engines. Spark Streaming processes the incoming data in mini-batch mode and the delay can be reduced to seconds while Storm processes a tuple once a time and can reduce delay to milliseconds. Thus, we choose Storm to implement this layer.

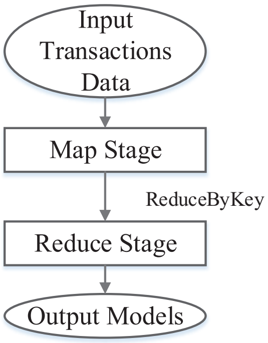

Fig. 3: MapReduce workflow for QF's training phase

V. Implementation

In this section, we give details on how we implement batch model training and online fraud detection with Big Data technologies, especially with Hadoop, Spark and Storm.

A. Separate Model Training with Hadoop

Figure 3 shows the workflow of QF training, which consists of two stages:

- Map Stage: for a given transaction record from the distributed storage layer, it is formated as a key/value pair, in which the key stands for the card ID and the value consists of payment attributes. Then the key/value pairs are shuffled to Stage 2 based on their key;
- Reduce Stage: pairs with the same key are aggregated together since they correspond to the same cardholder's historical transactions. Then they are used to train the same model. After the training, the model will be stored in the key-value sharing layer, in which the key still represents card ID while the value contains the model.

The implementation of the two stages on Hadoop is very straightforward, as shown in Algorithm 1 and 2.

Algorithm 1 Mapper function for key aggregation on Hadoop.

```
Input: $t_i^j$ //the jth transaction of Card i;
Output: $< i, t_i^j >$ //card ID and its transaction;
  $i = splitCardId(t_i^j)$
  $< i, t_i^j >= generateKVPair(i, t_i^j)$
  Emit $< i, t_i^j >$
```

Algorithm 2 Reducer function for model training on Hadoop.

```
Input: $list(< i, t_i^j >)$ //list of all transactions of Card i;
Output: $< i, model_i >$ //trained model of Card i;
  for $< i, t_i^j >$ in $list(< i, t_i^j >)$ do
    $list_i.add(t_i^j)$
  end for
  $model_i = train(list_i)$
  $< i, model_i >= generateKVPair(i, model_i)$
  Emit $< i, model_i >$
```

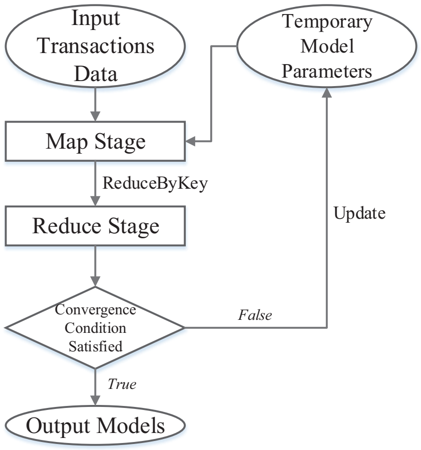

Fig. 4: Iterative workflow for EF's training phase.

B. Iterative Model Training with Spark

Figure 4 shows the workflow of EF training, which also consists of two stages:

- Map Stage: for all input transactions data from the distributed storage layer, they will be partitioned and each Map Stage task will handle a part of the data. Then, each Map Stage task will also update a part of model parameters in this stage and they will be shuffled to Reduce Stage;
- Reduce Stage: this stage will aggregate all parts of model parameters and merge them into one. Then, it will check whether the convergence condition is satisfied. If the condition is not satisfied, the temporary model parameters will updated and Map Stage will be started again. But when the condition is satisfied, the aggregated parameters will be stored in the key-value sharing layer.

For QF models, the training is mainly done in the Reduce Stage since models are trained for each cardholder. However, for EF models the training is done in Map Stage, since the big training job needs to be partitioned. By using Spark, the input transactions data and the temporary model parameters can be cached in memory to accelerate iterative processing. The implementation of the two stages on Spark is shown in Algorithm 3.

Algorithm 3 RDD function for model training on Spark

```
Input: $T$ //transaction data
Output: $TMP$ //Trained Model's Parameters
  $inputRDD$ = $read(T).cache()$
  initialize model parameters in array $TMP$
  while convergence condition is not satisfied do
    $tmpParamsRDD$ = $inputRDD.map(TMP)$
    $TMP$ = $tmpParamsRDD.reduce()$
  end while
  save $params$
```

C. *Streaming Fraud Detection with Storm*

Figure 5 shows the streaming workflow for fraud detection, which contains several stages called bolts:

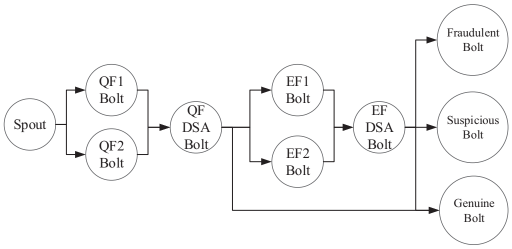

Fig. 5: Storm topology

- QF Bolt contains different QF detection algorithms. They calculate fraud scores independently based on their own models;
- QF DSA Bolt merges the fraud scores from different QF Bolts and then decide whether a transaction is suspicious or not. Those suspicious transactions will be sent to EF Bolts for further detection;
- EF Bolt contains different EF detection algorithms. They also calculate fraud scores based on their own models;
- EF DSA Bolt merges the fraud scores from different EF Bolts and then decides whether a transaction is fraud or not.
- Fraudulent Bolt is used to handle fraudulent transactions properly;
- Genuine Bolt is used to store genuine transactions in distributed storage layer;
- Suspicious Bolt is responsible for accepting suspicious transactions from EF DSA Bolt and handle them properly(e.g., manual verification).

Since each bolt is running different tasks, the system needs to dynamically allocate computational resources for them. Meanwhile, since the connections between bolts are very complicated, the system also needs to have the ability of dynamic scheduling. We have implemented this streaming detection layer with Storm. Due to space limitation, we would not give the pseudo-codes here.

VI. Experiment

A. *Test Environment*

We evaluate all experiments in a cluster with 11 nodes. Each node holds the following hardware configurations: 2 Intel(R) Xeon(R) E5645 CPU with 6 cores per processor running at 2.40GHz; Intel(R) PRO/1000 Network Connection (1Gbps); 3 x 1 TB SATA disk; 64 GB RAM. The software configurations for each node are as the follows: Apache Hadoop 2.5.0, Apache Spark 1.5.1, Apache HBase 1.1.4, Apache Storm 0.10.0, Apache ZooKeeper 2.4.6.

B. *Transaction Dataset*

Due to a limited access to credit card transaction data, many researchers use synthetic data to simulate real workload on CCFDS. In this paper, we also use Markov Modulated Poisson Process Model (MMPPM) [10] to generate a simulated transaction dataset. In MMPPM, there are two states: genuine state G and fraudulent state F. q_{gf} and q_{fg} are two transform possibility between these two states. If MMPPM is at good state, it will generate a genuine transaction. Otherwise, a fraudulent transaction is generated.

In our simulation, each transaction consists of 20 payment attributes. We apply one attribute called ”transaction interval” (the interval between current and previous transaction of a credit card) to represent transaction frequency. The attribute ”transaction interval’ follows Poissonian distribution. The other attributes are generated randomly. We generate 5 transaction dataset which contains 10,000, 30,000, 100,000, 300,000 and 1,000,000 credit cards’ transactions within one year.

C. *Detection Algorithms*

In this experiment, we choose two unsupervised algorithms (DBSCAN[10], HMM[1]) in QF and one supervised algorithm (Logistic Regression[7]) in EF, which are all widely used in credit card fraud detection. It should be noted that the goal of this experiment is to evaluate our hybrid framework to see whether it has the ability to handle a bursting amount of transactions in real-time. Therefore, the chosen algorithms here are only used to simulate the computation workload and we do not give further discussion about accuracy issues.

D. *Distributed Storage Throughput*

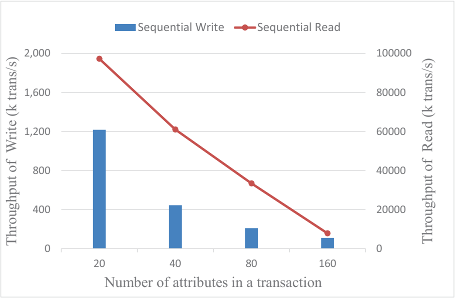

Fig. 6: Throughput of sequential write and read on HDFS

We evaluate the throughput of Distributed Storage Layer when the number of a transaction’s attributes vary from 20 to 160 to simulate possible transactions with different length. We use 10 clients to write transactions into HDFS and use 70 clients to read data from HDFS in parallel. The HDFS cluster consists of 11 nodes. Figure 6 shows the overall throughput of both sequential read and write on Hadoop. The x axis represents the length of each transaction and the y axis represents the throughput which shows the number of transactions done within one second. From the figure we can find that the throughput will decrease when transaction length is increasing. However, the system is able to write up to a million of transactions per second and read up to 100 millions transactions per second.

TABLE II: Training Time on Hadoop

Number of credit card	HMM / s	DBScan / s	LR / s
10,000	63	36	323
30,000	109	43	574
100,000	145	79	1253
300,000	347	240	3489
1,000,000	992	575	13030

TABLE III: Training Time on Spark

Number of credit card	HMM / s	DBScan / s	LR / s
10,000	72	66	18
30,000	138	120	54
100,000	444	192	78
300,000	600	378	204
1,000,000	1125	780	300

E. Batch Training Performance

In the evaluation of Batch Training Layer, we use a simulated one year transaction data to train models in QF and EF on our Hadoop and Spark cluster whose number of credit cards varies from 10,000 to 1,000,000. Both the Hadoop and Spark cluster consist of 11 nodes. During training, the number of iterations of each algorithm is fixed to make sure that the computation workload are the same for both platforms. The Batch Training Layer is an off-line module. There is no need to train and update models in realtime manner.

Table II and Table III give the training time of different algorithms on Hadoop and Spark Clusters. For HMM and DBScan, each cardholder owns a model. In LR, there is only one model, which is trained by all the transaction data. When the number of credit cards reaches 1,000,000, the throughput of the training phase in QF is about 1,000 models per second, and the throughput in EF is 0.003 models per second.

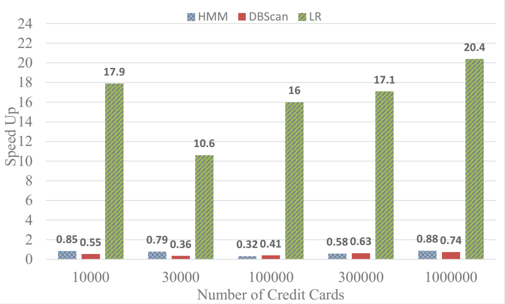

Fig. 7: Speed up of Spark over Hadoop for model training.

Figure 7 illustrates the speed-up of Spark over Hadoop to train models in Batch Training Layer. The training process of models in EF is completed by muliple iterative jobs. Transaction data and temporary model parameters are stored on disks in Hadoop, while they can be cached into memory in Spark. This is the reason why Spark outperform Hadoop on training models in EF. When training models in QF, each model is trained by only one cardholder's historical transactions, the training process can be completed within one job. The transferring time between map and reduce tasks are overlapped by the computation time of map tasks in Hadoop, while generated key/value pairs will not be transferred to the next stage until the current stage has completed in Spark. Thus Hadoop performs better when training models in QF. The results match our analysis that Spark suits iterative computation more than Hadoop while Hadoop gets less overhead when dealing with single-job computation.

F. Key-Value Sharing Throughput

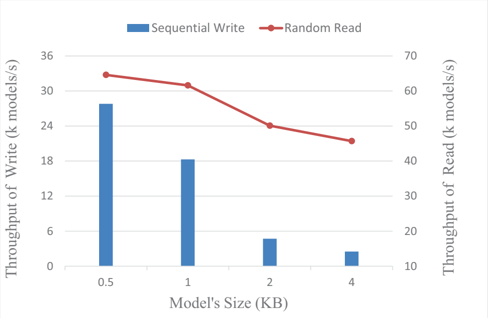

Fig. 8: Throughput of sequential write and random read on HBase.

Since detection models are sequentially updated by Batch Training Layer, and randomly read by Streaming Detection Layer frequently, we put the trained models in Key-value Sharing Layer using HBase. The HBase cluster consists of 11 nodes, of which one is HMaster and the others are HRegionServer. Figure 8 shows the throughput of sequential write and random read when the size of each model varies from 0.5 KB to 8 KB. We can find that the throughput of sequential write drops linearly when the model's size increases, while the throughput of random read only drops a little. This is because disk seek time is a main part of total time spent in random read, while data transferring time is key part of sequential write.

G. Streaming Detection Performance

The streaming detection layer is deployed on an 11-node Storm cluster. The submitted topology is configured as 10 workers and 240 executors. Figure 9 shows the delay of the streaming detection layer when the speed of incoming transactions varies from 2,500 trans/s to 40,000 trans/s. When the speed of incoming transactions is less than 10,000 trans/s, the delay is only one second. Assume that each cardholder has a credit card payment every 24 hours and the total number of cardholder is 100,000,000, the average speed of incoming transactions is 11,574 trans/s (100,000,000 trans/ 24 hours). It means our system can support near real-time detection for 100,000,000 cardholders. For an increasing incoming transactions, the delay will increase significantly, due to a limited amount of computing resources.

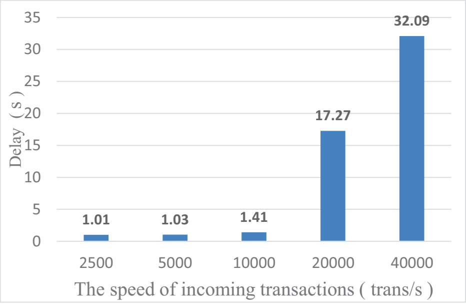

Fig. 9: Detection delay of Streaming Detection Layer on Storm

H. *Overall Performance Analysis*

Basically, the system can be divided into two stages: detection stage and training stage. In the detection stage, the Streaming Detection Layer can support a throughput of 10,000 trans/s with a very low delay. The throughput of random read in Key-Value Sharing Layer has a higher throughput of nearly 50,000 models/s. In the Distributed Storage Layer, the throughput of sequential write is much higher. Therefore, the Streaming Detection Layer is the performance bottleneck in the detection stage. In the training stage, the throughput of the Batch Training Layer is less than that of sequential read in Distributed Storage Layer and sequential write in Key-Value Sharing Layer. Thus, the Batch Training Layer is the bottleneck in the training stage. However, since the system is highly scalable, we should allocate more computing resources to Streaming Detection Layer and Batch Training Layer to improve the overall performance during real system deployment.

VII. Conclusion

Online fraud detection is challenging due to the burst amount of trading transactions that are happening everyday. In this paper, we design a hybrid framework to solve this problem. Our framework aims at fusing different detection algorithms to improve accuracy and using a four-layer design to handle data storage, model training, data sharing and online detection. We implement the framework with latest Big Data technologies, which help to build a scalable, fault-tolerant and high performance system. The hybrid framework can also be applied in other similar application fields, for example internet advertising fraud detection, telecom fraud detection and so on. However, this work still has a lot things to be done, e.g., better integration of more detection algorithms and other Big Data tools, test with real transaction data, systematic optimizations to all components in the framework, etc. We will try them in our future work.

VIII. Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments. This work is partially sponsored by the National Basic Research 973 Program of China (No. 2015CB352403), the National Natural Science Foundation of China (NSFC) (No. 61261160502, No. 61272099), the Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT), the Scientific Innovation Act of STCSM (No. 13511504200), and the EU FP7 CLIMBER project (No. PIRSES-GA-2012-318939).

References

[1] D. Iyer, A. Mohanpurkar, S. Janardhan, D. Rathod, and A. Sardeshmukh, “Credit card fraud detection using hidden markov model,” in *Information and Communication Technologies (WICT), 2011 World Congress on*, Dec 2011, pp. 1062–1066.

[2] E. Ngai, Y. Hu, Y. Wong, Y. Chen, and X. Sun, “The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature,” *Decision Support Systems*, vol. 50, no. 3, pp. 559 – 569, 2011.

[3] S. Wang, “A comprehensive survey of data mining-based accounting-fraud detection research,” in *Proceedings of the 2010 International Conference on Intelligent Computation Technology and Automation - Volume 01*, ser. ICICTA ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 50–53.

[4] S. Ghosh and D. L. Reilly, “Credit card fraud detection with a neural-network,” in *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, vol. 3, Jan 1994, pp. 621–630.

[5] M. Syeda, Y.-Q. Zhang, and Y. Pan, “Parallel granular neural networks for fast credit card fraud detection,” in *Fuzzy Systems, 2002. FUZZ-IEEE’02. Proceedings of the 2002 IEEE International Conference on*, vol. 1, 2002, pp. 572–577.

[6] D. P. F. R. A. Stine, “Variable selection in data mining:,” *Journal of the American Statistical Association*, vol. 99, no. 466, pp. 303–313, 2004.

[7] I.-C. Yeh and C. hui Lien, “The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients,” *Expert Systems with Applications*, vol. 36, no. 2, Part 1, pp. 2473 – 2480, 2009.

[8] R. J. Bolton, D. J. Hand *et al.*, “Unsupervised profiling methods for fraud detection,” *Credit Scoring and Credit Control VII*, pp. 235–255, 2001.

[9] P. J. Bentley, J. Kim, G.-H. Jung, and J.-U. Choi, “Fuzzy darwinian detection of credit card fraud,” in *the 14th Annual Fall Symposium of the Korean Information Processing Society, 14th October*, 2000.

[10] S. Panigrahi, A. Kundu, S. Sural, and A. Majumdar, “Credit card fraud detection: A fusion approach using dempstershafer theory and bayesian learning,” *Information Fusion*, vol. 10, no. 4, pp. 354 – 363, 2009.

[11] A. Kundu, S. Panigrahi, S. Sural, and A. K. Majumdar, “Blast-ssaha hybridization for credit card fraud detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 4, pp. 309–315, 2009.

[12] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[13] Z. D. Zhao and M. s. Shang, “User-based collaborative-filtering recommendation algorithms on hadoop,” in *Knowledge Discovery and Data Mining, 2010. WKDD ’10. Third International Conference on*, Jan 2010, pp. 478–481.

[14] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, “Data mining with big data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, Jan 2014.

[15] J. Arias, J. A. Gmez, and J. M. Puerta, “Scalable learning of k-dependence bayesian classifiers under mapreduce,” in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 2, Aug 2015, pp. 25–32.

[16] S. Ramrez-Gallego, S. Garca, H. Mourio-Taln, and D. Martnez-Rego, “Distributed entropy minimization discretizer for big data analysis under apache spark,” in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 2, Aug 2015, pp. 33–40.

[17] V. Zaslavsky and A. Strizhak, “Credit card fraud detection using self-organizing maps,” *Information and Security*, vol. 18, p. 48, 2006.

[18] M. A. Arbib, *The Handbook of Brain Theory and Neural Networks, Second Edition*. MIT Press, 2002.

[19] M. Krivko, “A hybrid model for plastic card fraud detection systems,” *Expert Systems with Applications*, vol. 37, no. 8, pp. 6070 – 6076, 2010.

[20] Y. Kltr and M. U. alayan, “A novel cardholder behavior model for detecting credit card fraud,” in *Application of Information and Communication Technologies (AICT), 2015 9th International Conference on*, Oct 2015, pp. 148–152.