Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

The Evolvement of Big Data Systems: From the Perspective of an Information Security Application[☆]

Gang Chen[a,*], Sai Wu[a], Yuan Wang[b]

[a] College of Computer Science, Zhejiang University, Hangzhou, 310027, PR China
[b] Netease (Hangzhou) Inc, Wangshang Road, Binjiang District, Hangzhou, 310052, PR China

A R T I C L E I N F O

Article history:
Received 12 September 2014
Accepted 4 January 2015
Available online xxxx

Keywords:
MapReduce
Pregel
Spark
Real-time analysis
Information security

A B S T R A C T

Recently, Google revealed that it has replaced the 10-year old MapReduce with its new systems (e.g., DataFlow) which can provide better performances and support more sophisticated applications. Simultaneously, other new systems, such as Spark, Impala and epiC, are also being developed to handle new requirements for big data processing. The fact shows that since their emergence, big data techniques are changing very fast. In this paper, we use our experience in developing and maintaining the information security system for Netease as an example to illustrate how those big data systems evolve. In particular, our first version is a Hadoop-based offline detection system, which is soon replaced by a more flexible online streaming system. Our ongoing work is to build a generic real-time analytic system for Netease to handle various jobs such as email spam detection, user pattern mining, game log analysis, etc. The example shows how the requirements of users (e.g., Netease and its clients) affect the design of big data system and drive the advance of technologies. Based on our experience, we also propose some key design factors and challenges for future big data systems.

1. Introduction

In a recent announcement from Google,[1] MapReduce [11] is abandoned as it is unable to handle the amounts of data Google wants to analyze these days. A new hyper-scale system, DataFlow, is considered as its successor which supports both batch and streaming data processing. Programmers can create complex processing pipelines using DataFlow to manipulate a huge size of data. In fact, besides DataFlow, Google already develops a series of big data systems, such as Dremel [22], Spanner [8] and Pregel [21], to replace the original two, MapReduce and BigTable [7]. Those systems show how big data technology evolves inside Google.

The same technology revolution also happens in the academic community. As the most popular open-source implementation of MapReduce, Hadoop attracts many research efforts [17] and has been widely applied to various real-world applications. However, Hadoop is also found inefficient in processing iterative jobs [4,23]. So GraphLab [20,13] which adopts a vertex-centric processing model similar as Pregel is being developed to support large-scale data mining algorithms. Most existing systems such as MapReduce (Hadoop) and Pregel (GraphLab) provide a fixed programming model for users, while the flexibility of the model determines the difficulty of writing programs on top of it. To address the problem, epiC [5,14] allows users to create their own programming model, simplifying the development of big data application and improving the performance significantly. The design of big data system is also affected by the advancement in computer hardware. As memory becomes cheaper and cheaper, we can equip a compute node with a very large memory, so that data can be fulled maintained in the distributed memory of a cluster. This observation motivates the development of in-memory based processing system, such as Spark,[2] which can run a job 100 times faster than Hadoop.

In summary, the design of big data systems keeps evolving when we need to handle larger-scale of data and more challenging user demands. This is also verified by our experience in developing and maintaining an information security system for Netease[3] which is collaborative project between Netease and Zhejiang University. In this paper, the information security system, especially its email spam module, is used as an example to illustrate how we

☆ This article belongs to Visions on Big Data.
* Corresponding author.
E-mail addresses: cg@zju.edu.cn (G. Chen), wusai@zju.edu.cn (S. Wu), wangyuan@corp.netease.com (Y. Wang).
[1] http://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system/.
[2] http://spark.apache.org/.
[3] http://www.163.com.

http://dx.doi.org/10.1016/j.bdr.2015.01.002
2214-5796/© 2015 Elsevier Inc. All rights reserved.

apply big data techniques in real-world application and how we change our design for new performance requirements. The remaining of the paper is organized as follows. First, we introduce the background of Netease information security system and its challenges in detecting the malicious information. Then, we present the design of our first version detection system which is based on Hadoop and HBase.[4] The Hadoop version is able to find more than 80% spams and works quite well until the number of emails, blog comments and microblogs per second increases to a new extent. Because Hadoop only supports batch processing, detection of new spams suffers a significant delay which may cause terrible user experience when new spams are flooding over Netease services. Hence, we replace the Hadoop system with a new streaming system based on Apache S4.[5] The new system applies human feedbacks to dynamically update its classification model, so that new spams can be detected more efficiently. However, we find that it is not easy to implement a complex classification model in the new system. We need the support for both batch and stream processing and a more flexible interface that can be used to integrate with various Netease services. So in our ongoing work, we are trying to build a new generic analytic system for big data. Its goal is:

- To provide a real-time or near real-time analytic service for extremely large data.
- To support high throughput updates from applications without violating the data consistence and affecting the performance of analysis.
- To enable third-party programmers to implement new applications on top of the exposed interfaces.

In this paper, we will present our initial design of the system and discuss some key challenges in implementing such a system. We hope that our experience can help developers design their big data systems by selecting the proper techniques.

2. Information security system

Netease is one of the largest email service providers in China. It provides both the free email service for individuals and the advanced email service for corporations. Due to its popularity in China, Netease email system becomes the target of many malicious advertisements. Besides, Netease is also a service provider of news, microblogging and e-commerce. All those applications can be considered as the sources of malicious information. Attackers send forged messages to users to advertise their products, obtain users' personal information or even distribute virus via attachments. Therefore, spam detection is a key feature in the system to improve user experience. Netease collaborates with Zhejiang University to develop its information security system, which can filter out the malicious information before delivering them to users. We integrate some state-of-the-art technologies from the research community to the real system to improve the performance.

The very first version of information security system developed by Netease is, in fact, a rule-based system. It maintains a database for malicious information senders, sensitive keywords and some other features. If an email or microblog shows one of the features, it will be marked as spam. Surprisingly, such a simple approach provides a very good performance at its early stage. For example, more than 60% spams can be identified by sender IDs. However, when senders of spams become more strategic, the rule-based approach is unable to identify most spams.

Different from the rule-based approach, our goal is to develop a model-based information filter which, given a document (e.g., emails or microblogs) set $\mathcal{D}$, classifies the documents into the ham set $\mathcal{H}$ and spam set $\mathcal{S}$. Users can selectively view or delete the documents based on the ham or spam tags. The core part of the system is a classifier which can be trained either offline or online using a well-tagged document dataset. Most current learning models, such as Naive Bayes [24] or SVM [9], can be adopted to provide a satisfied precision and recall ratio. In the following discussion, we show the three versions of our information security systems and the key factors that lead to those specific designs. To simplify the presentation, we use email spam detection module as the example in the following discussion, because other malicious information modules actually follow the same design.

3. Hadoop-based system

The first rule of designing a spam detection system is that it cannot delay the delivery of an email. In other words, it should be non-intrusive for the email system. So we build a Hadoop-based system which trains the classification model in an offline manner while classifies the incoming emails in real-time. The design is also based on the requirement of Netease which is listed as below:

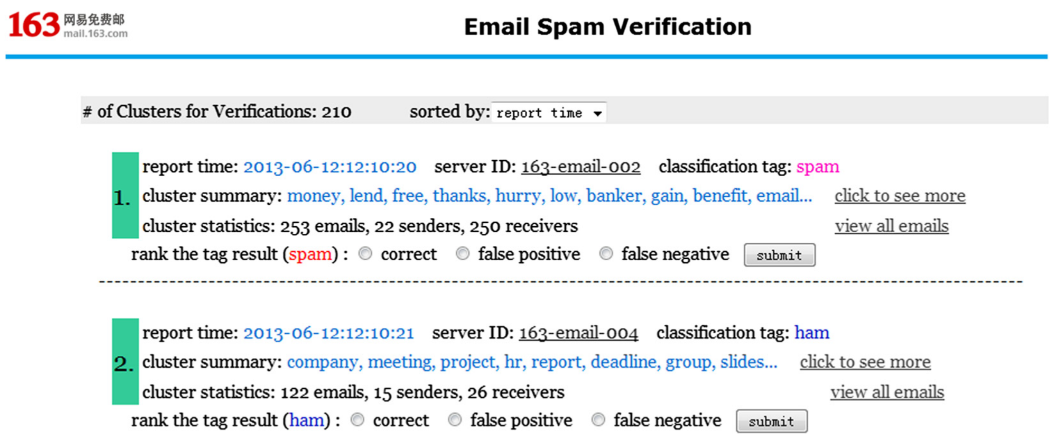
Scalability At that time, Netease email system needs to process thousands of emails per second averagely. For the Hadoop-based approach, the online classifier is very efficient because it only computes the distances between the new email and the ham/spam cluster. A single server can handle more than ten thousands emails per second. Hence, there is no performance bottleneck here. For the offline part, we collect more than one hundred million emails per day, which are approximately a few hundred gigabytes of semi-structured data. Tagging the emails and training a Naive Bayes model take about two hours in a 10-node cluster which is not very good but acceptable.

Update The slow training process is acceptable, because Netease does not pose requirement for real-time updates. We log the emails received so far and update the classification model using the new emails at every 3:00 am when the system is least busy. As there are too many emails, we first invoke a Hadoop-based KMeans algorithm to cluster the emails into small groups. Each group will be summarized and forwarded to the email administrators to tag it as hams or spams (in Netease, email administrators are responsible for monitoring the email system and the spam system. They help tune the systems to achieve their best performances). Fig. 1 shows the interface for the email administrators. The clustering and tagging process may last for one to one and half hour depending on the number of groups. Then, we need another half hour to do the model updating.

Model complexity Since MapReduce is a very flexible programming framework, we can implement Naive Bayes, SVM [6], Decision Tree, Neural Network [19] or other popular training models. Some complex models such as Neural Network may take a longer time for training than the Naive Bayes. So it is a tradeoff between the complexity of the model and the accuracy of the spam detection.

Fig. 2 shows the workflow of Hadoop-based system which consists of an online classification process and an offline training process. In the online process, the email streams are logged to the distributed file system or a key-value store such as HBase before forwarded to the classification model. Then the model tags emails

[4] http://hbase.apache.org/.
[5] http://incubator.apache.org/s4/.


163 网易免费邮 mail.163.com

Email Spam Verification

# of Clusters for Verifications: 210 sorted by: report time

1. report time: 2013-06-12:12:10:20 server ID: 163-email-002 classification tag: spam
cluster summary: money, lend, free, thanks, hurry, low, banker, gain, benefit, email... click to see more
cluster statistics: 253 emails, 22 senders, 250 receivers view all emails
rank the tag result (spam) : ○ correct ○ false positive ○ false negative submit

2. report time: 2013-06-12:12:10:21 server ID: 163-email-004 classification tag: ham
cluster summary: company, meeting, project, hr, report, deadline, group, slides... click to see more
cluster statistics: 122 emails, 15 senders, 26 receivers view all emails
rank the tag result (ham) : ○ correct ○ false positive ○ false negative submit

Fig. 1. Result verification UI.

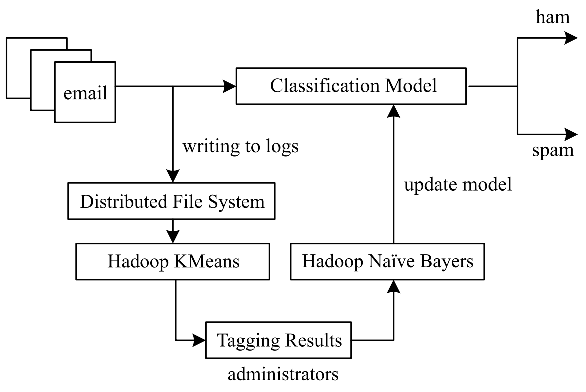


Fig. 2. Hadoop-based system.

as hams or spams and delivers those emails to users with their tags.

In the offline process, the training job is scheduled at 3:00 am every day. The first series of Hadoop jobs run the conventional KMeans algorithm which groups the similar emails into a cluster. In the clustering algorithm, we use the bag-of-word model, where for a new email $E_i \in \mathcal{D}$, we transform it into a set of words $E_i = \{w_1, \dots, w_k\}$.[6] The distance between two emails is computed as the cosine distance of their word vectors. In particular, *mappers* compute the distances between an email E_i and all cluster centers $\mathcal{C}$. Let C_j be the cluster center with the minimal distance. We generate a key value pair (C_j, E_i) and shuffle it to the corresponding *reducer*. So the emails of the same cluster are sent to the same *reducer* where a new center is computed. In next job, *mappers* will compute the distances between emails and new centers. The process continues until no email changes its cluster center in the *map* phase. Finally, the cluster result is stored as an inverted list. The key of the list is the cluster center ID, while the value is a list of emails that belong to the cluster. We also generate a summary for the cluster by extracting the highest weighted keywords.

The cluster results are presented to the email administrators who will identify a whole cluster as hams or spams. The tagging results along with the emails are then used in updating the existing classification model. If we use Naive Bayes model, only one MapReduce job is required. Specifically, two types of *mappers* are created. One loads the old model (namely, the probabilities of a word in the ham and spam email respectively) and one estimates the probabilities for words in new mails. In *reducers*, we merge the probabilities of the same word and use it in the new classification model. If we use Neural Network, multiple jobs are processed consequentially. Due to its complexity, we discard the details.

[6] Segmentation tools are used to transform Chinese sentences into words.

4. Streaming system

The Hadoop-based system worked well until Netease received a burst of complaints from its clients in some specific periods of time. Every summary, there is a popular show *The Voice of China (TVC)* attracting more than 50 million viewers. The malicious advertiser forges emails like "you are invited by TVC for interviews", "you are given two free tickets for the live show", etc. Those emails also contain links to their advertisement URLs or even some cheating messages such as "transferring money to a specific account". As a new type of spams, our Hadoop-based system fails to identify them in the first few days. The reason is twofold. First, our classification model is updated in a batch manner. There is a delay between the emergence of new spams and update of the model. Second, we incrementally update our model using new emails. So the importance of new spams will only be recognized when we get enough samples. A new spam detection system is required to address the problem which can update its classification model in real-time. We list our challenges as below:

Scalability The email system needs to handle an increasing number of emails. When we design the new spam system, the number of emails per second doubles and keeps increasing. Moreover, different from the Hadoop system, if we want to update the model in real-time, we need to perform the training process in an online manner. The training process should be efficient and light weighted to avoid delaying the delivery of emails.

Update To improve the user experience, the classification model continuously changes to catch the trends of new spams. One metric is to measure the delay between the first emergence of a spam type and it being tagged as spams in the system.

Model complexity At our design stage, to reduce the processing overhead, we prefer the simple model like Naive Bayes, so that we can provide a real-time response and update our model efficiently.

The second version of spam detection system is designed as a streaming system on top of Apache S4. The architecture of our spam detection system is shown in Fig. 3. Based on the functionalities, we have three types of streaming nodes, *probability node*, *classification node* and *clustering node*.

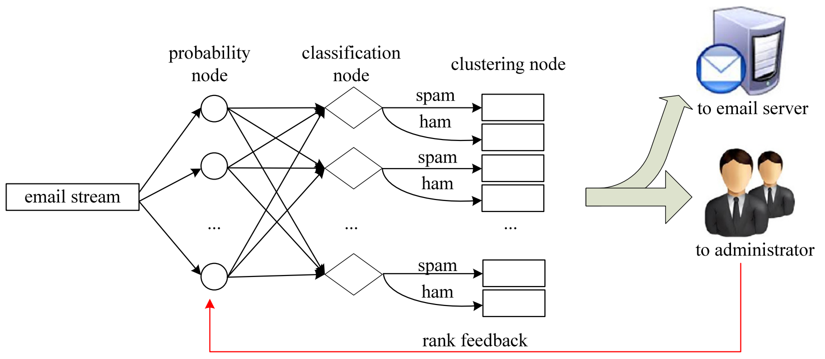


Fig. 3. Architecture of streaming system.

4.1. Probability node

Before our system starts, we learn a probability model offline using some collected training data. The model is loaded by the probability node during the initialization. In particular, we assign an ID I_i to each probability node n_i. I_i ranges from 0 to $N-1$, where N is the number of nodes. For each word w_j, we say node n_i is its host node of w_j, if $hash(w_j)\%N = I_i$. We group the words by their host nodes and generate N input files, one for each probability node. The file maintains the probability of each word and the numbers of ham and spam emails that contain the word. When the system starts, the probability node loads its corresponding file into memory.

Similar to the Hadoop system, we transform a new email $E_i \in \mathcal{D}$ into a set of words $E_i = \{w_1, \dots, w_k\}$. We generate key-value pairs (w_j, E_i) for email E_i, which are sent to the probability node for processing. To reduce the network overhead, in fact, we group all key-value pairs to one probability node into a batch. Specifically, the batch B_x for node n_x is

$$B_x = (\{(w_j, E_i) | hash(w_j)\%N = I_i\}, k)$$

The number of words in the email (k) is appended to the batch, so the classification node knows that it has received all results for an email.

Given a batch B_x, the probability node retrieves the probability P_j for each w_j in B_x and computes $\epsilon_j = ln(1-P_j) - lnP_j$. If we cannot find P_j for word w_j (namely, the word does not appear in the training dataset), we set $\epsilon_j = 0$. Then, a pre-aggregation is performed:

$$f(E_i, B_x) = \sum_{\forall w_j \in B_x} \epsilon_j$$

A new key-value pair $(E_i, (f(E_i, B_x), |B_x|, k))$ is generated and forwarded to a specific classification node based on the hash value of the key. The value is a triple, consisting of the aggregation value, the number of the words in the batch and total number of words in the email.

4.2. Classification node

The classification node collects the aggregation results of an email from different probability nodes. For email E_i, the classification node updates its η value continuously. When a new aggregation result of batch B_x is received, we update the η value as $\eta = \eta + f(E_i, B_x)$. We also maintain a count c_i for E_i. Each time, c_i is updated as $c_i + |B_x|$.

If $\frac{1}{1+e^{\eta}} > \mathcal{T}$, we tag E_i as a spam and stop the classification process for E_i. In other words, we do not need to wait for all aggregation results to make a decision. On the contrary, we can only tag an email as a ham, when all its results are received ($c_i = k$) and $\frac{1}{1+e^{\eta}} < \mathcal{T}$.

Based on the email's tag (spam or ham), we shuffle it to different set of clustering nodes. To guarantee that the similar emails are sent to the same clustering node, we adopt the LSH (Locality Sensitive Hashing) approach. In particular, the LSH based on p-stable distribution [10] is applied. We first transform the email E_i into a vector $V(E_i) = (v_1, \dots, v_k)$, where v_i denotes the TF (term-frequency) value of word w_i. To reduce the dimensionality of $V(E_i)$, we map words into m ($m << k$) buckets and rewrite $V(E_i)$ as $(v_1, \dots, v_m)$. Each v_i denotes the accumulative TFs of all words in the ith bucket. The idea of LSH is to define a hash function $h: \mathbb{R}^m \to \mathbb{U}$, such that for any two emails E_i and E_j:

- If $||V(E_i) - V(E_j)|| \le r$, then $P(h(V(E_i)) = h(V(E_j))) \ge p_1$.
- If $||V(E_i) - V(E_j)|| > cr$, then $P(h(V(E_i)) = h(V(E_j))) \le p_2$.

h hashes the vector to an integer. r is the distance between two vectors. $P(h(V(E_i)) = h(V(E_j)))$ denotes the probability that the two vectors share the same hash value. For a good LSH, $p_1 >> p_2$.

In p-stable LSH, we define h as:

$$h(V(E_i)) = \lfloor \frac{V(E_i) \times X + b}{r} \rfloor$$

where X is an m-dimensional vector and its element is independently generated based on the p-stable distribution (currently, we use Gaussian distribution). b is an integer, uniformly selected from the range $[0, r]$. Normally, we generate c hash functions $h_1, \dots, h_c$, so that the similar emails may share the same hash value with a higher probability. Then, we use the c hash values as the keys to shuffle the emails to different clustering nodes. If the clustering node receives duplicate emails, it will just keep one copy.

4.3. Clustering node

The clustering node maintains a buffer to cache the received spams/hams. Note that a clustering node only handles either the hams or the spams, not the both. When the buffer is full, it runs a local clustering algorithm to group the emails based on their content similarity. During the clustering process, a new buffer is established to collect the received emails. So the clustering process is a non-blocking process. In this paper, we apply the *correlation clustering* technique, which has the advantage that it provides a clustering method to partition the emails without requiring a specified cluster number (e.g., the parameter K in KMeans) in advance.

Correlation clustering operates on a signed graph where the edges are labeled as positive (+) or negative (−), indicating that the two ending nodes are similar or dissimilar. The goal is to cluster the nodes so that the number of disagreements (in terms of the number negative edges inside clusters plus the number of positive edges between clusters) are minimized. The dual problem that maximizes the agreements can be defined similarly. Formally, given a signed graph $G = (V, E)$, let E^+ be the set of positive edges, and E^- be the set of negative edges ($E = E^+ \cup E^-$). Correlation clustering computes a clustering such that the following cost function is minimized:

$$cost = \sum_{e \in E^+} x_e + \sum_{e \in E^-} (1 - x_e) \tag{1}$$

where $x_e = 0$ if the two ending nodes of e are assigned to the same cluster, $x_e = 1$ otherwise.

Correlation clustering is an NP-complete problem. However, one can rewrite it into an integer program and solve it exactly using IP solvers for small graphs. For large graphs, polynomial time approximation algorithms [2] can be applied.

When the buffer is full, we output the emails in the buffer and construct the graph G. We let each email denote a node in V and compute the similarity between each pair of nodes. The similarity of two emails is estimated using the cosine function:

$$sim(E_i, E_j) = \frac{V(E_i) \times V(E_j)}{||V(E_i)|| \times ||V(E_j)||}$$

A threshold θ is pre-defined. If $sim(E_i, E_j) \geq \theta$, we create an edge between them. θ is a tunable parameter, but we found that the results of correlation clustering is less affected by θ. So in our implementations, we randomly test a few θs in the training process and select the best one, which will be fixed for the system.

4.4. Model update

After clustering, the cluster nodes push the results to the spam management system which notifies the administrators to verify the precision of the results. The results of verification are also added into the training dataset to improve the performance of the model. For a cluster C, we use $|C|$ to denote the number of emails in C. Let $\mathcal{W}$ denote all words in C. We define $\theta(w_j)$ to return the number of emails in C that contains w_j. Three types of feedbacks can be collected:

1. If C's tag is spam/ham and the rank is "correct", the number of spam/ham emails that contain w_j increases by $\theta(w_j)$, while the total number of spam/ham emails increases by $|C|$.
2. If C's tag is spam and the rank is "false positive", the number of hams that contain w_j increases by $\theta(w_j)$, while the total number of ham emails increases by $|C|$.
3. If C's tag is ham and the rank is "false negative", the number of spams that contain w_j increases by $\theta(w_j)$, while the total number of spam emails increases by $|C|$.

The feedbacks are used to update the information maintained by the probability nodes. To reduce the overhead, the feedback is aggregated into a tuple (w_j, h_j, H, s_j, S), where w_j is a specific word, h_j and s_j are the hams/spams that contain w_j and H and S represent the total numbers of hams and spams respectively. The tuple is sent to the probability node using w_j as the hash key. Once receiving the feedback, the probability node starts its update process.

4.5. Performance test

To evaluate the performance of new spam detection system, we deploy it on a 16-node cluster of Netease. We compute the precision of our spam detection system by comparing the newly generated tags to the ground truth. Figs. 4 and 5 show the results.

In our system, we use the feedbacks from the humans to adjust our classification model. The feedbacks are achieved by forwarding

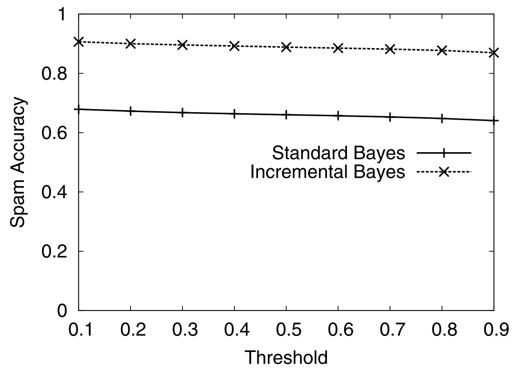


Fig. 4. Precision of spam detection.

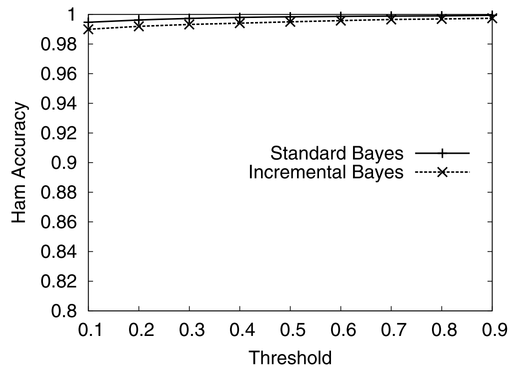


Fig. 5. Precision of ham detection.

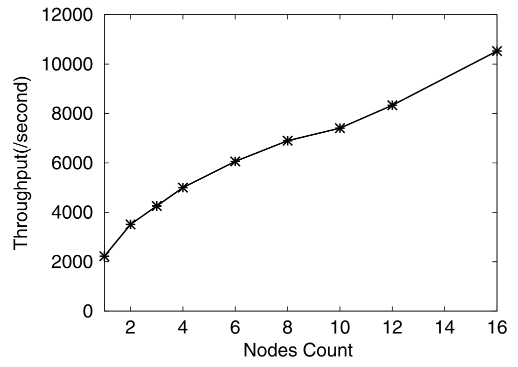


Fig. 6. Performance of scalability.

the clustering reports of spams or hams to the email administrators of Netease, who are well trained to identify the spams from hams efficiently. If we neglect the feedback process, our spam detection system works as a standard Bayes model, where no new spam words are inserted into the model. As we can see from the figures, our incremental Bayes model significantly improves the precision of spam detection by 20%. It is because the model evolves based on the feedbacks to handle the new emerging spams. On the other hand, the precision of ham detection is slightly worse than the standard Bayes, as our system adopts a more aggressive model. We do not show the recall in the figures, because the email is either spam or ham and the recall can be computed from the precisions of spam and ham.

Another reason of redesigning the spam detection system is that the number of emails increases by orders of magnitude and the old system can no longer provide a satisfied performance. Therefore, in the new system, we exploit the parallelism of nodes to speed up the processing. In the experiments, we vary the number of cluster nodes in Fig. 6 and observe that our system shows

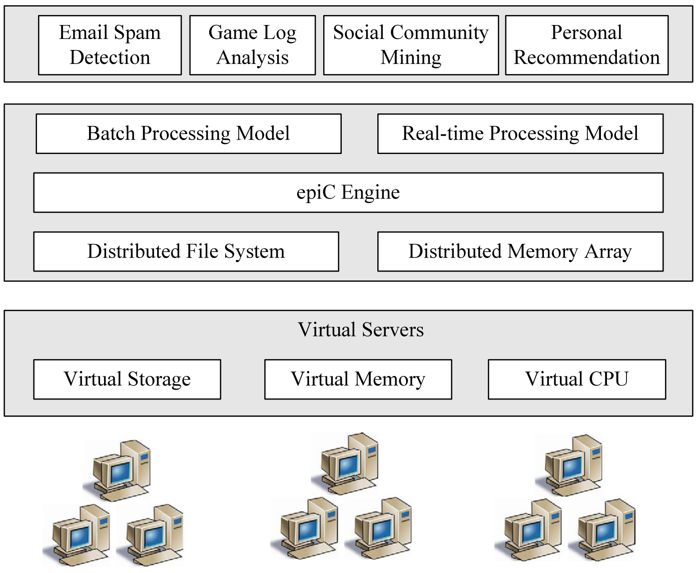


Fig. 7. Architecture of real-time analytic system.

a linear scalability, which indicates that to improve the throughput (number of emails processed per second), we can add more compute nodes into the cluster.

5. A generic real-time analytic system

One major problem of our streaming system is lack of flexibility. The architecture of *probability node*, *classification node* and *clustering node* is tailored for the Naive Bayes model. If we want to adopt a more comprehensive model to improve the accuracy, we need to completely change the design. Moreover, it is impossible to extend the system to support other analytic jobs such as game log analysis and social community detection. In summary, the architecture is limited to the spam detection system using Naive Bayes model.

Another issue is the scalability and load imbalance problem. Clustering nodes are the bottleneck of the system which may slow down the whole system, while probability nodes and classification nodes only perform simple computations. Because we use the streaming system, the configuration of window size affects the performance. For probability nodes and classification nodes, the window size is 1. Namely, an email will be processed immediately at those nodes. On the other hand, the window size of clustering nodes equals to the predefined buffer size. When buffer is full, we invoke the clustering algorithm. Larger buffer incurs high memory overhead and delays the update of model. On the contrary, if we use a small buffer, we can update the model more frequently. However, the clustering results are not good, as some clusters only contain few members. This may make the administrators hard to decide whether the emails are spams or not and further affect the accuracy of the model.

So our ongoing work is to design a generic real-time analytic system based on the following considerations:

Scalability The system should be able to support different analytic jobs of Netease in the next two years. The size of data will continuously increase (e.g., the number of email per second is expected to increase to more than ten thousands). Given limited computation resource, the system should provide scalable and efficient services. Moreover, because Netease has several data centers located in Hangzhou, Beijing and Guangzhou, the analytic system should support the deployment on multiple data centers.

Update The analytic system is expected to handle both batch processing and real-time processing. For email spam detection and malicious payment detection, the system should return real-time results, while for the game log analysis and social community mining, it may process a huge size of data in an offline manner.

Model complexity The system needs to provide a flexible programming interface, so that different applications and models can be efficiently developed on top of the system. The interface is required to be compatible with Hadoop to reduce the efforts of migration, as most existing analytic jobs of Netease are processed in Hadoop.

Fig. 7 shows the architecture of our new analytic system. In the bottom layer, machines of multiple data centers are organized as virtual compute resource which are dynamically allocated for different applications. The core layer of the system is based on a modified version of epiC [14], manipulating data from both the distributed file system (DFS) and the distributed memory array. We also create two computation models, one for batch processing and one for the real-time streaming processing. The batch processing model is implemented in epiC as a bulk synchronization model where a job is processed in multiple steps and we perform a global synchronization for compute nodes at the end of each step. The real-time streaming model is implemented as an asynchronous model where the compute node will immediately start the computation of next step without communicating with others. In the following discussion, we briefly introduce our design and implementation details.

5.1. epiC engine

epiC is based on the actor model where each compute node is considered as an individual actor, communicating with others via messages. Each actor performs its own processing specified by a user-defined function which accepts messages as its parameters. In epiC, actors do not transfer data between each other. Instead, they only use messages to indicate the location of a specific partition in the DFS. All actors will directly fetch their data from the DFS for processing. This strategy allows users to link actors dynamically as a DAG (Directed Acyclic Graph). Note that different from previous systems, such as S4 and Dryad [27], epiC does not maintain the DAG explicitly which is dynamically set up based on the message flow. The structure of the DAG can even change during the processing. This is one of the most important features of epiC, the flexibility.

epiC's flexibility is also reflected on its programming model. It provides a simple *unit* interface, where users can write their customized functions. We can implement MapReduce or Pregel model on top of epiC. So previous programs written for Hadoop can be run on epiC with a few modifications. We can also develop a streaming system on top of epiC by using the asynchronous model. Currently, the streaming based spam detection system is re-implemented using epiC. Besides Naive Bayes model, it now supports other complex classification models.

epiC also supports customized optimizations, so that we can extend it to support in-memory processing to further improve the performance.

5.2. *In-memory processing*

Users are no longer satisfied with the performance of offline analysis. For example, Netease game designers want to have a real-time interactive tool to analyze the game log. They may start from a very general query, such as retrieving all gamers who have not logged in for three days. After examining the list, they can submit a more specific query, e.g., grouping gamers who have not logged in for three days by their characters' levels. They expect every query to be processed in real-time, so they can adjust their queries adaptively.

To build a system that can provide near real-time analytic services, we have to apply the in-memory techniques due to slow I/O performances of existing hardware. In particular, we are focusing on the following points:

- We are implementing a distributed memory array which follows the same design philosophy of RDD [28] to support scalable and fault tolerant data storage in memory. The memory array provides an interface for epiC's units to access its data. In this way, we extend epiC to an in-memory processing engine.
- Simply storing the data in memory cannot fully exploit the benefit of new architecture. We need some specific optimizations. For example, new index structures should be designed to maximize the hit ratio of L2 cache, instead of reducing I/O costs. Radix sort works better than quick sort and merge sort. Column-oriented storage model can adopt the late materialization approach more aggressively.
- Even the price of DRAM continuously drops, it is still too expensive to create a pure memory cluster to support Petabyte scale data. Therefore, a large portion of data are still maintained in hard disks. The data partitioning strategy (which part of data are maintained in memory) is crucial for the performance. Our initial solution is to design a fast loading approach which can efficiently parse disk data into memory. The dataset for analyzing will be loaded into memory on the fly. However, this approach incurs high overheads if data are frequently swapped in or out from memory.

5.3. *Processing updates*

As a real-time system, we cannot apply the batch update scheme which is widely adopted in the MapReduce systems. Updates and queries must be processed concurrently. To guarantee the consistence of analytic results, two typical strategies are employed, locking and multi-versions. In Google's Spanner [8], the conventional two-phase locking is used. Although locking affects the system's throughput, Google argues that the programmers and users should be responsible for that. If they want a better performance, they should avoid using the locks. In our system, we run analytic queries and updates together. Because some queries need to scan a large portion of data, they may lock the entire database blocking all the updates. So in our solution, we adopt the multi-version approach as in Hyper [15]. We still apply the two-phase locking to resolve the conflictions between updates. For each analytic query, we create a specific version V of data for it. Originally, V equals to current dataset. After a few updates, current dataset is newer than V. To handle such cases, we replicate a tuple t before applying any update to it. Let T be the tuples that have not been updated during the query processing. We use T' to denote the set of replicated tuples. V is then materialized as $T \cup T'$. In other words, by using replication, we avoid locking tables for the analytic query. Note that if there are multiple analytic queries, we may create multiple versions of data. The storage overhead, however, is low, because we will discard the old versions of data after those queries have been processed.

5.4. *Deployment on multiple data centers*

Designing a system that be deployed on multiple data centers is much more challenging. There are a few issues affecting the performance or even correctness of the data processing.

First, the network latency between data centers is much higher and unstable. When partitioning data, we should always store the data that are frequently accessed together in the same data center. An efficient analytic algorithm is partition-aware which intentionally avoids shuffling data between nodes in different data centers. This violates our design rule, namely, providing a transparent programming model for up-layer users. Hence, we are trying to developing a module for epiC that can automatically optimize the performance by considering the network partitions.

Second, the clocks of different data centers are not synchronized. This makes timestamp based approach does not work any more. Two consecutive messages may be handled in different orders at each data center. To address the problem, we select some nodes as time servers from each center and synchronize their clocks. Other nodes will ask the time servers to get the correct clock.

Finally, it is difficult to keep the CAP property. If one data center fails or disconnects from the network, all its data are not accessible. If we keep a replica for data in that center, we will have the consistent issue. It is too costly to keep the replica (normally in other data centers) consistent with the master copy. Besides, if we use the replica to process updates and failures, when the data center recovers, we need to start a synchronization process. To reduce the complexity of failure recovery, when a data center fails, updates to its data will be rejected. We only use the replica to answer the analytic queries. Updates to other data centers are still allowed. This simple solution works well when we only have a few data centers. A more sophisticated approach is being developed to support more complex multi-center architecture.

6. Conclusions and open problems

In this paper, we use the information security system in Netease as an example to illustrate how big data system evolves when users' requirements keep changing. We start from an offline Hadoop system to an online streaming system. Finally, we want to design a generic system that can provide near real-time analytic services for many Netease applications, such as spam detection, game log analysis and social community mining. Based on our experiences, no solution can address all big data problems, especially when 1) data size keeps increasing; 2) more complex user requirements need to be handled; 3) the emergence of new hardware violates the old design; and 4) the old system becomes too complicated for maintenance.

In our developing system, we face a series of technical challenges that have not been well addressed by both academic community and industry. Currently, we compromise our solution by adopting some conventional techniques which may not be the best choices for the new real-time analytic system. In summary, we list them as below:

1. **Performance problem:** As mentioned previously, we use in-memory techniques to speed up our processing. When most data can be maintained in memory, the system gives us a real-time performance. However, there are many cases that only a small portion of data can be buffered in memory. For example, a one-year game log can be a few hundreds terabytes, while our distributed memory is less than 100 TB. When memory is the bottleneck, the system degrades as a MapReduce-like system. Many data are scanned from disks, incurring high I/O overheads. The analytic jobs may take hours to complete. Such cases definitely cannot be called as real-time analysis. We still do not have a solution for this problem. We try to use compression to reduce the data size, but the compression ratio varies a lot and the decompression is also costly.
2. **Consistence problem:** To guarantee the data consistence when updates and queries are processed concurrently, we replicate data for each query. However, we find that when many queries are running together, the system's performance slows down significantly. This shows that our solution is not a scalable solution which cannot handle many concurrent users. If we do not maintain multiple replicas, we must use locks which are more expensive. One possible solution is to use the non-locking strategy which predefines some orders of the updates and queries, so that we do not need to maintain multiple replicas and locks are not required. However, it seems that existing non-locking approaches cannot be directly applied to our system. We are still searching for a proper one.
3. **Failure recovery problem:** The failure recovery becomes difficult for 1) updates are handled in real-time; and 2) data are split into many nodes which even belong to different data centers. Node failure happens almost every day. It is a non-trivial problem, as we need to search and update the data in the failed node. In current implementation, we adopt a simple strategy. When failure happens, we only allow the queries to run on the replicas, while the updates will be blocked. This strategy is sometimes complained by our users who must log their new updates and apply them when the failed node recovers. A more elegant approach must be designed to replace the existing one.

In our opinion, the big data technology will continuously evolve with a larger data size and more comprehensive user requirements. It is difficult to predict a clear future for big data systems. However, based on our experiences with clients and application developers, we find that the following technologies are crucial to improve the big data applications to next level.

First, a new processing model is required to inherit the advantages of both big data systems and state-of-the-art data management systems. When people find that MapReduce and Hadoop cannot provide a satisfied performance, they try to build indexes [12], data cubes [18] and query optimizer [25] on top of MapReduce, while those techniques are well adopted by modern database systems. On the other hand, to improve the scalability, parallel database also embeds MapReduce model into its processing engine.[7] The two systems tend to merge into one, and HadoopDB [1] is the first try. However, to fully exploit the benefit of scalability of big data systems and performance of database systems, a new parallel processing model should be designed from scratch, not based on the enhancement of old systems. The new parallel model is required to support many state-of-the-art technologies in data management systems transparently, such as indexing, query optimization, view maintenance and column-oriented storage. It should also allow the programmers to implement their customized functions using a flexible interface for both batch processing and pipeline processing.

Second, big data systems are supposed to handle various types of data from different sources. For instance, the information security system of Netease maintains data from the email system, news system, microblog system and online e-commerce transaction system. The smart city system processes data from sensors, cameras, mobile phones and many other devices. It is a challenging job to analyze those data with different formats together. One typical application of information security system is to link a user's records in the game log, microblog system and transaction data to detect possible game cheating. This requirement actually joins a relational table, a social graph and a text-format log file together. There is no such big data tools that can perform the analytic jobs for hybrid data formats. GraphX [26] is recently introduced to handle relational data and graph data. epiC [14] can also support analyzing multi-format data. Unfortunately, they are limited to specific types of data in an explicit way (users have to write the codes for handling different data formats by themselves). To facilitate the development of big data applications on hybrid data formats, we should formally define a set of operators to process the hybrid data and translate them into efficient jobs on top of existing processing engines such as MapReduce, Spark and epiC.

Third, to exploit the features of new hardware, many existing distributed algorithms should be redesigned. A future cluster node may be equipped with hybrid hardware consisting of CPU, GPU, DRAM, NVM (Non Volatile Memory), SSD and HDD. The big data algorithm must be optimized for specific hardware configuration. For example, in the in-memory system, the intuition is to maximize the utilization of L1 cache, while in the disk-based system, the main concern is the I/O overhead. So the algorithms of in-memory system should be designed as cache conscious. One possible solution is to employ the radix based processing technique. In particular, the index adopts the radix tree structure [16] which is an in-memory trie tree with adaptive fanouts. The sort algorithm is based on the radix sort, a non-comparative sorting algorithm which turns out to be a better sorting algorithm for modern hardware [3]. Although some algorithms have been proposed for new hardware and hybrid framework, they are mainly targeted at the single-node system, while to handle big data applications, we need the scalable distributed versions.

Fourth, most end users are still unfamiliar with big data systems. Even we provide analytic tools for them, they are still unaware of what the systems can do for them. So user-friendly visualization techniques are necessary to narrow the gap between big data system and its users. The visualization techniques should display the analytic results in an intuitive way, so that users can identify the interesting results effectively. Besides, each interaction between users and visualized results will trigger a new query for the big data system. To enable a fast response, the back-end system is expected to provide a real-time performance and the visualization algorithms need to transform the users' event into a proper and optimized query. Due to the complexity and diversity

[7] https://blogs.oracle.com/datawarehousing/entry/mapreduce_oracle_tablefunction.

of analysis jobs, visualization algorithms should support as many operations as possible, e.g., typical data cube operations like slice, drill down/up and roll up.

Last but not the least, big data applications are still limited to specific domains, such as finance, e-commerce and biology. New applications will emerge when we combine big data technologies with other conventional industries, while in the combination process, those applications will pose new requirements for big data systems, pushing us to search and propose new solutions. We are optimistic about the adoption of big data technologies. More users will update their systems using big data technologies, when they see more successful stories.

Acknowledgements

The work of Gang Chen was supported by the National Science Foundation of China under Grant No. 61472348. The work of Sai Wu was supported by The National Key Technology R&D Program of the Ministry of Science and Technology of the People's Republic of China (Grant No. 2013BAG06B01).

References

[1] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, Avi Silberschatz, Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads, Proc. VLDB Endow. 2 (1) (2009) 922–933.

[2] Nir Ailon, Moses Charikar, Alantha Newman, Aggregating inconsistent information: ranking and clustering, J. ACM 55 (5) (2008).

[3] Cagri Balkesen, Gustavo Alonso, Jens Teubner, M. Tamer Özsu, Multi-core, main-memory joins: sort vs. hash revisited, Proc. VLDB Endow. 7 (1) (2013) 85–96.

[4] Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Ernst, Haloop: efficient iterative data processing on large clusters, VLDB J. 3 (1–2) (September 2010) 285–296.

[5] Yu Cao, Chun Chen, Fei Guo, Dawei Jiang, Yuting Lin, Beng Chin Ooi, Hoang Tam Vo, Sai Wu, Quanqing Xu, Es^2: a cloud data storage system for supporting both oltp and olap, in: ICDE, 2011, pp. 291–302.

[6] G. Caruana, Maozhen Li, Man Qi, A mapreduce based parallel svm for large scale spam filtering, in: Fuzzy Systems and Knowledge Discovery, FSKD, 2011, pp. 2659–2662.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, Bigtable: a distributed storage system for structured data, ACM Trans. Comput. Syst. 26 (2) (2008).

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J.J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford, Spanner: Google's globally distributed database, ACM Trans. Comput. Syst. 31 (3) (2013) 8.

[9] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines, Cambridge University Press, 2000.

[10] Mayur Datar, Nicole Immorlica, Piotr Indyk, Vahab S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG'04, 2004, pp. 253–262.

[11] Jeffrey Dean, Sanjay Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (January 2008) 107–113.

[12] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Efficient big data processing in hadoop mapreduce, Proc. VLDB Endow. 5 (12) (2012) 2014–2015.

[13] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, Carlos Guestrin, Powergraph: distributed graph-parallel computation on natural graphs, in: OSDI, 2012, pp. 17–30.

[14] Dawei Jiang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Sai Wu, epic: an extensible and scalable system for processing big data, Proc. VLDB Endow. 7 (7) (2014) 541–552.

[15] Alfons Kemper, Thomas Neumann, Hyper: a hybrid oltp&olap main memory database system based on virtual memory snapshots, in: ICDE, 2011, pp. 195–206.

[16] Viktor Leis, Alfons Kemper, Thomas Neumann, The adaptive radix tree: artful indexing for main-memory databases, in: ICDE, 2013, pp. 38–49.

[17] Feng Li, Beng Chin Ooi, M. Tamer Özsu, Sai Wu, Distributed data management using mapreduce, ACM Comput. Surv. 46 (3) (2014) 31.

[18] Li Feng, M. Tamer Özsu, Gang Chen, Beng Chin Ooi, R-store: a scalable distributed system for supporting real-time analytics, in: ICDE, 2014, pp. 40–51.

[19] Zhiqiang Liu, HongYan Li, Gaoshan Miao, Mapreduce-based backpropagation neural network over large scale mobile data, in: Natural Computation, ICNC, 2010, pp. 1726–1730.

[20] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, Joseph M. Hellerstein, Distributed graphlab: a framework for machine learning in the cloud, Proc. VLDB Endow. 5 (8) (2012) 716–727.

[21] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski, Pregel: a system for large-scale graph processing, in: SIGMOD Conference, 2010, pp. 135–146.

[22] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Dremel: interactive analysis of web-scale datasets, Proc. VLDB Endow. 3 (1) (2010) 330–339.

[23] Makoto Onizuka, Hiroyuki Kato, Soichiro Hidaka, Keisuke Nakano, Zhenjiang Hu, Optimization for iterative queries on mapreduce, Proc. VLDB Endow. 7 (4) (2013) 241–252.

[24] Vangelis Metsis Telecommunications, Vangelis Metsis, Spam filtering with naive Bayes – which naive Bayes? in: Third Conference on Email and Anti-Spam (CEAS), 2006.

[25] Sai Wu, Feng Li, Sharad Mehrotra, Beng Chin Ooi, Query optimization for massively parallel data processing, in: SoCC, 2011, p. 12.

[26] Reynold S. Xin, Daniel Crankshaw, Ankur Dave, Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica, Graphx: unifying data-parallel and graph-parallel analytics, CoRR (2014), arXiv:1402.2394.

[27] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, Jon Currey, Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language, in: OSDI, 2008, pp. 1–14.

[28] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, Ion Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: NSDI, 2012, pp. 15–28.