A Cloud Service Resource Classification Strategy Based on Feature Similarity

Qingtao Wu, Min Cui, Mingchuan Zhang, Ruijuan Zheng, and Ying Lou
Information Engineering College, Henan University of Science and Technology, Luoyang, 471023 P. R. China
Email: wqt8921@haust.edu.cn

Abstract—There are now a vast array of heterogeneous cloud service resources, which makes it difficult to identify suitable services for the various types of cloud users. A classification of cloud service resources would help users find suitable cloud services more easily. We propose such a classification strategy, which has two parts. First, we improve the original naive Bayesian classification algorithm, designing a Bayesian classification algorithm based on feature similarity. Second, to improve the efficiency of the classification algorithm, we design a parallel programming model using the Hadoop platform. Simulation results show that the proposed classification strategy is feasible and effective, improving not only the resource classification accuracy but also greatly enhancing the processing efficiency for large-scale cloud service resources.

Index Terms—Cloud Service; Resource Classification; Bayesian; Feature Similarity; MapReduce Programming Model

I. INTRODUCTION

The rapid development of cloud computing technology means that cloud services [1] have become the main service pattern on the Internet. The aim of cloud services is to provide services that suit all of a user’s demands. However, because there are such a vast array of cloud service resources, it is difficult to identify suitable services for the various types of users. A classification of cloud service resources would be a good solution to this problem.

For any type of existing service or resource in the cloud, we can utilize cloud service resources classification to classify it and make it belong to a particular category according to its attributes and characteristics. Cloud services can be seen as a huge pool of virtual service resources [2, 3]. They integrate all types of resources, such as computing resources, storage resources, software resources, and platform resources, in a network which form a distributed-service cluster system [4]. However, the classification of cloud service resources is facing big challenges caused by the features of large quantities and variety. Classification will not only require precision, validity, and reliability, but will also make great demands on the system's ability to deal with large-scale data.

Recently, researchers have classified service resources in a network using intelligent learning methods [5] such as clustering algorithms, decision-tree algorithms, and naive Bayes algorithms. Studies [6-8] have shown that these existing classification methods for service resources tend to focus on the classification of the small datasets for common network resources, paying no attention to the relationships among the various features of service resources, which will greatly influence the classification accuracy. As a result, new classification methods about cloud service resources will have to be introduced, in order to make the resource classification more flexible and adaptable to changing service requirements within the cloud network.

In this work, we propose a classification strategy for the study of service resources in the cloud. It can support large-scale dataset categorization, taking into account the dynamics and heterogeneity of user requests as well as resource classification performance. The contributions of this paper are twofold. First, we present an improved classification algorithm according to naive Bayes characteristics, which offers maximum accuracy by using similarity calculations for the various features of the resource dataset to evaluate the role each feature plays in the classification. Second, we implement parallelization of the improved classification algorithm combining with the MapReduce parallel programming model [9] on the Hadoop platform [10]. This streamlines the operation of the distributed computing tasks and enhances significantly the classification performance of the algorithm when dealing with large-scale resource datasets in the cloud. Extensive experiments show that our strategy is feasible and effective in a dynamic cloud environment.

The remainder of the paper is organized as follows. In Section II, we provide an overview of work related to resource classification methods. Section III presents our improved classification algorithm. Section IV describes the parallelization process for the proposed classification algorithm and Section V presents our experimental results. Finally, in Section VI, we conclude the paper.

II. RELATED WORK

A substantial amount of research has been devoted to the problem of service resource classification in a network, where the objective is to find appropriate classification solutions that simultaneously minimize costs such as time taken or memory overhead [11], increase the categorized result’s availability and reliability, and raise the success rate of execution, thereby
meeting the service demands of users during the process of service delivery [12].

Recently, tractable classification solutions have been proposed in the form of sorting algorithms based on clustering and decision-tree algorithms, which have obtained good results to some extent. Wang et al. [13] proposed an efficient hierarchical categorization approach based on the aggregation of the software’s online attributes and the design of a hierarchical categorization framework. It used the weighted aggregation of software’s descriptions and tags across multiple repositories to categorize the massive software hierarchically. Taking the integrity description of service-requirement semantics as a fulcrum, Wen et al. [14] studied three technologies, namely requirements-semantics acquisition techniques, requirements-semantics-driven aggregated methods for service software, and service customization based on requirement semantics. As a hot research orientation for datastream mining, the clustering algorithm is a powerful tool for analyzing multiple datastreams [15-17], where issues were in-depth discussions involving processing cost and classification accuracy.

Considering the stability and operability of classification, the decision-tree algorithm is intuitively appealing. An empirical hyper-heuristic evolutionary algorithm [18] is capable of automatically designing top-down decision-tree induction algorithms. These are of great importance, considering their ability to provide intuitive and accurate knowledge representations for classification problems. Compared with traditional decision-tree algorithms, the proposed hyper-heuristic evolutionary algorithm was found to be more efficient in testing large numbers of gene-expression datasets. Dash et al. [19] proposed a novel scheme for the measurement, identification, and classification of various types of power quality (PQ) disturbances. The proposed method employed a fast variant of the S-Transform algorithm for the extraction of relevant features, which were then used to distinguish among different PQ events by a fuzzy decision tree (FDT)-based classifier. Afsari et al. [20] developed a novel fuzzy modeling scheme based on the FDT classifier by using subtractive clustering and a multi-objective evolutionary algorithm. This was used to construct an accurate and interpretable system defined as an interpretability-based fuzzy decision tree classifier.

The solutions described above have drawbacks. For example, being subjected to the “dimension effect”, the clustering method cannot directly handle high-dimensional data for effective classification, while the decision-tree algorithm has poor scalability, with the running time being greatly increased as the quantity of the data increases. Therefore, these methods can prove very inefficient in highly dynamic scenarios, as is the case with today’s heterogeneous cloud environment.

In contrast, the naive Bayes algorithm [21] based on a probability density function is characterized by a lower error rate and higher accuracy. It describes the mapping relationship between condition attributes and classification properties in a classification system, but its conditional-independence assumption is often violated in the real world. However, it enables researchers to improve the accuracy easily by adding external parameters, thereby laying the basis for a new classification algorithm.

In terms of data processing efficiency, initial studies have shown three main parallel programming models for data-parallel computation, namely the multiple-threaded model, the messaging model, and the MapReduce model [22]. A multiple-threaded model such as the OpenMp model [23], which implements information interaction via shared memory and can only run on machines with a shared-storage structure, cannot be used for clustering and does not apply to the calculation of large quantities of data. A messaging model such as MPI [24], which adopts interprocess communication to coordinate parallel computation, has a series of problems such as a large memory overhead and the programming complications. Compared to these models, the MapReduce programming model can effectively avoid the problems faced by the first two models, thereby implying its suitability for a complex and dynamic networking environment.

III. EFFICIENT CLASSIFICATION METHOD: THE IMPROVED NAIVE BAYES ALGORITHM

A. Naive Bayes Classification

By calculating the posterior probability, the naive Bayes classification (NBC) method determines the category to which a data sample belongs. The basic idea is to use the joint probability of properties and categories to estimate the category of the new sample, using the Bayes formula and simplifying assumptions from probability theory. We can formulate Bayes’ theorem as follows.

Assume that \( A_1, A_2, \ldots, A_n \) is a group of pairwise incompatible events, while event \( B \) can occur simultaneously with only one of the events. Formula (1) is then established.

\[
p(A_i | B) = \frac{p(B | A_i) / p(A_i)}{\sum_{j=1}^{p} p(B | A_j) p(A_j)}
\]

(1)

Each data sample is represented by an n-dimensional feature vector \( X = \{x_1, x_2, \ldots, x_n\} \), which describes separately the feature values from \( A_i \) to \( A_n \). \( C = \{C_1, C_2, \ldots, C_i\} \) is the predefined class set, with \( C_i \) representing the ith category. According to this method, for each sample \( X \) of unknown category, we can first calculate the probabilities \( p(X | C_i) p(C_i) \) that \( X \) belongs to category \( C_i \), then select the category whose probability is maximum as its category. \( C(X) \) is the class tag of the final classification results, thereby obtaining the NBC model:

\[
C(X) = \arg\max_{C_i \in C} p(C_i) \prod_{j=1}^{n} p(x_j | C_i)
\]

(2)
B. An Improved NBC Algorithm Based on Feature-Similarity Weighting

NBC is a simple and effective classification model. However, the performance of this model may be poor because of the assumption about conditional independence, which makes the importance of all condition features be consistent with that of decision features for the classification. Moreover, some factors, such as similarity of features, will have more influence than the others on the classification results. Based on these observations, we will introduce the concept of similarity in combination with the weighted Bayesian formula already in use [25-26]. Different features will be given different weights by calculating feature similarities. The weighted Bayesian formula is:

\[ C(X) = \arg \max_{C_i \in C} p(C_i) \prod_{j=1}^{m} p(x_j | C_i) \]  

(3)

In (3), \( w_j \) represents the weight assigned to the \( k \)th feature. Larger values for \( w_j \) imply a greater impact on the classification results. However, in any real application, there will be correlation and similarity among the features of the dataset used to some extent. The roles that the various features play in the process of classification will also vary. This paper proposes a classification algorithm based on feature similarity that applies feature-similarity computing to formula (3). This method is used to determine the weight \( w_j \) assigned to the feature \( k \). The calculation is described below.

Definition 1. Given two object spaces \( O_i \) and \( O_j \),
\[ d(X_{ik}, X_{jk}) = \begin{cases} 0 & X_{ik} \subseteq X_{jk} \text{ or } X_{jk} \supseteq X_{ik} \\ \frac{E_{X_{ik}} - E_{X_{jk}}}{3(\sigma_{X_{ik}} + \sigma_{X_{jk}})} & X_{ik} \nsubseteq X_{jk} \text{ or } X_{jk} \nsubseteq X_{ik} \end{cases} \]  

(4)

In the above formula (4), \( E_{X_{ik}} \) and \( E_{X_{jk}} \) are the central values of the feature set \( X_{ik} \) and \( X_{jk} \) respectively, with \( 3\sigma_{X_{ik}} \) and \( 3\sigma_{X_{jk}} \) being half of the coverage area of the two feature sets separately, namely:
\[ \sigma_{X_{ik}} = \frac{E_{X_{ik}} - X_{\text{min}_{ik}}}{3} \text{ or } \sigma_{X_{jk}} = \frac{X_{\text{max}_{ik}} - E_{X_{jk}}}{3} \]  

(5)

In (5), \( X_{\text{min}_{ik}} \) and \( X_{\text{max}_{ik}} \) are the minimum and maximum of the feature set \( X_{ik} \), respectively.

The training sample set for category \( C_i \) is \( X_i = \{x_1, x_2, \cdots, x_n\} \), \( n_i \) is the number of class \( i \) samples, the \( k \)th feature set is expressed by \( X_{ik} \), the expectation of \( X_{ik} \) is \( E_{X_{ik}} \), the minimum is \( X_{\text{min}_{ik}} \), and the maximum is \( X_{\text{max}_{ik}} \) (\( i = 1, 2, \cdots, n; k = 1, 2, \cdots, n \)).

Sorting \( E_{X_{ik}} \) and \( \sigma_{X_{ik}} \) of feature \( k \) in ascending order, \( l \) is category id after sorting and \( c \) is the total of all categories. The distance of feature \( k \) between categories is \( D(k) \), which is represented as shown below.

\[ D(k) = \min_{l=1,2,\cdots,c} \left( \frac{E_{X_{ik}} - E_{X_{jk}}}{3(\sigma_{X_{ik}} + \sigma_{X_{jk}})} \right) \]  

(6)

The normalized feature similarity of feature \( k \) is \( S(k) \).
\[ S(k) = \begin{cases} 0 & D(k) \geq 1 \\ 1 - D(k) & \text{else} \end{cases} \]  

(7)

We then calculate the weight of each feature from the following formula.
\[ w(k) = \frac{1-S(k)}{\sum_{j=1}^{n}[1-S(j)]} \]  

(8)

This calculates each feature weight \( w(k) \) based on feature similarity according to the training set data. We substitute \( w(k) \) into formula (3) to obtain the Bayesian classification algorithm based on feature similarity. We can then apply this algorithm to cloud service resources classification for an unknown resource data sample \( X \) by calculating the probability of its belonging to each category and choosing the category with the highest probability as the category for \( X \). This is our proposed cloud service resource classification method based on feature similarity (SFCRC). The algorithm classification process is depicted below.

![Algorithm classification process](image)

It can be seen from Fig. 1 that the algorithm comprises three stages.

1) Preparation stage: The main work is to determine features of training samples and then form the feature vector set.
2) Classifier training stage: The task of this phase is to generate a classifier by calculating related parameters and recording the results.
3) Application stage: The task of this stage is to classify those await categorized items using the classifier.

IV. PARALLELIZATION OF CLASSIFICATION ALGORITHM

As described in Section III, we improved the naive Bayes algorithm to include feature similarity. In this Section, we describe a parallelization process for the classification algorithm in dynamic environments. Combined with the characteristics of the MapReduce parallel programming model, the improved Bayes algorithm can classify cloud service resources effectively.

A. The Extraction of the Feature Vector Collection

There may be hundreds of features in the feature-vector collection for a data file about cloud service resources. The expressive quality of the feature-vector collection will directly affect the efficiency of classification. After comprehensive consideration, we choose the following eigenvector of the resources data for analysis.

a) Resource filename: It can be classified by analyzing keywords in the filename.

b) Resource filename extension: The filename extension can be filtered to classify the resource file.

c) The description content of the resources: If the filename extension belongs to the type of text file, the resource file can be classified by the text content.

d) The size of the resource file: After getting the size of the file, we can give different weights to different sizes for classifying the resource file.

e) Source of resources: It can be classified according to keywords in the field which the resource comes from.

B. Mapreduce Parallel Implementation of the Classification Algorithm

We now consider the basic ideas for parallelizing the NBC algorithm based on feature similarity using MapReduce. Initially, training sets and testing sets are downloaded to the local node from the data file system by a work node. Through a series of preprocessing operations on the datasets such as data segmentation and feature extraction, the datasets are organized into the input data format for the Bayes model. A Map calculation for each test sample is then started, completing the calculation of the matching degree between test samples and training samples. Finally, the intermediate computational results are sent to the Reduce node for normalization and to generate the final results. To meet the requirements for the MapReduce calculation model, the data to be processed should be stored in the form of lines before invoking the Map operations. The specific operation is to split the data according to the line, which requires slices of data to be independent of each other. The default “split” operation is performed automatically by MapReduce, with no code needing to be written by the user. (There is an option for user-generated code.) The MapReduce parallelization method is shown in Fig. 2.

The design of the Map function is shown in Fig. 3. The detailed description is as follows.

1) First, it calls a built-in function (split) to read the dataset to be processed line by line, which is convenient for later operations. The category name and the feature set are then converted into the mapping <key, value>. From Line 4 to Line 10 in Fig. 3, each of the training samples and test samples are traversed to calculate the match distance between them. The results are stored in a predefined collection.

2) From Line 13 to Line 20 in Fig. 3, the SFCRC classification model is first loaded. Each feature item is then traversed and the relevant parameters are extracted from the classification model, after which the conditional probability and the weight based similarity are calculated. Finally, the posterior probability of the given sample for all categories is obtained.

3) From Line 21 to Line 28 in Fig. 3, all posterior probabilities are compared to determine the maximum. The category to which the maximum posterior probability corresponds is then taken as the input sample’s category. “1.0” represents the number of times that a feature occurs. For a test task, output the key-value pairs: < (the original class label before calculation, the category name that is selected), 1.0> . For a classification task,
output the key-value pairs: < (the original class label before calculation, the category name that is selected), the maximum posterior probability>. The results are finally stored in a collection.

1. Input: <key, value>
2. Output: <key, vector> Context Context0 and Context1
3. Begin
4. // Serve all the training samples \( t \), take out class identity \( t \)
5. For \( i = 1 \) to \( t \) (training dataset) do
6. \( i = \) FindCatalog ( \( i \) );
7. // Traverse the test sample \( k \), calculate the matching distance with the training samples. key is the line number of test samples, the results are stored in Context0
8. For all \( k \in \) testfile do
9. MD = MatchDegree ( \( k \), \( i \) );
10. Context0. write (key, vector( \( t \), MD));
11. End For
12. End For
13. // load the SFCRC classification model, take out the value of \( P(C) \)
14. Class SFCRC ( );
15. new SFCRC ( ).get ( \( P(C) \) );
16. // Traverse each feature item \( t \), calculate conditional probability and the weight based similarity, obtain the posterior probability of unknown samples.
17. For \( t = 1 \) to \( n \) ( \( <key, value> \)) do
18. new SFCRC ( ).cond_Probability ( \( t \) );
19. new SFCRC ( ).weight ( \( w_t \), similarity ( \( t \) ));
20. new SFCRC ( ).posterior_Probability ( \( t \) );
21. // Comparing probability
22. compare ( \( C(X)_{new} \), new SFCRC ( ).posterior_Probability ( \( t \) ));
23. // The results of test or classification task are stored in Context1
24. If test_task ( \( t \) )
25. Context1. write (key, 1.0);
26. End If
27. If class_task ( \( t \) )
28. Context1. write (key, \( C(X)_{new} \) );
29. End If
30. End For
31. End

Figure 3. Map pseudocode

The Reduce function is mainly designed for normalized operation, which is relatively simple. The design of the Reduce function is shown in Fig. 4. The detailed description is as follows. First, take the output results of the Map function as the input to the Reduce function. Then, from Line 7 to Line 13 in Fig. 4, for a classification task, directly output the intermediate computational results. For a test task, sum the “value” with the same “key” to evaluate the classified effect of the output results.

V. SIMULATION EXPERIMENTS AND ANALYSIS OF RESULTS

A. The Experimental Environment

We first built a cloud-computing simulation platform. The configuration of the platform was as follows. The experimental cluster involved eight computers, with one chosen to service the master nodes NameNode, JobTracker and the remainder servicing slave nodes DataNode, TaskTracker. The hardware configuration for each node comprised a CPU (Intel Core i3-2130), RAM (4 GB), and hard drive (500 GB). The software configuration involved Hadoop (Version 1.0.4) and JDK (Version 1.7.0.25) [27, 28]. The experimental platform programs were implemented in the Eclipse integrated development environment [29]. The cluster configuration was built according to the recommended methods of the Hadoop official website.

1. Input: <key, value>
2. Output: <key, ID> Context Context
3. Begin
4. // Key-value pairs are added to ArrayList for executing operations, \( t \) is the class identity of samples, “value” is the output result of the Map function
5. For all key and value do
6. // output the results directly in the classification task, store results in Context
7. If class_task ( \( t \) )
8. ArrayList (context1 (key,value));
9. New ArrayList result;
10. // Add \( t \) to the result
11. result. Add (key, ArrayList.get( \( t \) ));
12. Context.write (key, result);
13. End If
14. // The test task sum the “value” with the same “key”
15. If test_task ( \( t \) )
16. ArrayList (key, sum (value));
17. End If
18. End For
19. End

Figure 4. Reduce pseudocode

B. Experiment and Results Analysis

Because it was difficult to obtain the required variety of service-resource data from cloud service providers, we adopted a synthetic data generator “DataFactory” for our experimental simulations. DataFactory is data simulation software developed by the Quest company, and is a test tool specializing in large amounts of data. We used an Oracle database [30] as the DataFactory’s interface, producing the experimental data via the association rule being applied to the field name of the database table. We finally obtained a cloud service resource data table, which was saved in the Oracle database in the form of a resource (number, resource class name, service description). The synthetic experimental data were divided into four categories, corresponding to “picture processing” (15,000 records), “store service” (18,000), “computing service” (20,000), and “Java application” (22,000) in the field “resource class name” in the table. From among the 75,000 records, we chose 10,000 for each class as training data (40,000 in total), while 8750 were chosen for each class as testing data (35,000 in total).

This experiment used a tokenizer called “PaoDingJieNiu”. The tool is in open-source code and can be used free of charge. It provides a Lucene interface, offers cross-platform and reliable performance, and is an optimal data segmentation tool. It selects the original feature set by using a statistical method.
The first group of experiments involved measuring the speedup ratio $S = T_r / T_m$, which is an important indicator of performance of a parallel algorithm. $T_r$ refers to the time spent in solving a problem on a single node and $T_m$ refers to the time that $m$ nodes take for solving the algorithm in parallel. In our experiment, we set the number of nodes participating in the MapReduce calculations to 1, 2, 3, 4, 5, 6, or 7. The classification was then tested on the dataset. The experimental results are shown in Fig. 5.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig5}
\caption{The speedup ratio experiment}
\end{figure}

It is evident from Fig. 5 that the speedup ratio is improved with an increase in the number of nodes. This increase can significantly improve the system-processing capacity for the same scale of data. This suggests that MapReduce has a good speedup ratio for processing the SFCRC classification algorithm. We can predict that this performance improvement would be even more obvious with large-scale data involving many more nodes.

In our second group of experiments, we compared classification by the traditional NBC and by the SFCRC, as shown in Table I.

\begin{table}[h]
\centering
\caption{The comparison of two classification algorithms}
\begin{tabular}{|c|c|c|c|}
\hline
Resource name & Test sample number & Percentage of correct classification & \\
& & NBC & SFCRC \\
\hline
Image processing & 206 & 99.03\% & 99.51\% \\
Store service & 185 & 98.38\% & 100\% \\
Computing service & 178 & 98.31\% & 99.44\% \\
Java application & 189 & 95.24\% & 98.94\% \\
\hline
\end{tabular}
\end{table}

From Table I, we find that the SFCRC classification usually achieves better classification accuracy than NBC classification. The correct rate is the percentage of the number of resources classified correctly as a proportion of the total number in the test dataset. The SFCRC correct rate in this experiment was 99.47\%, which was 1.73\% better than that for NBC. This occurs because SFCRC not only considers the close relationship between each feature item, but also takes into account important information about the distribution of feature items within the class. Classification performance for the heterogeneous service resources in the cloud will therefore be greatly improved, again making the SFCRC classification algorithm more reliable.

VI. CONCLUSIONS

In this paper, we present a cloud service resource classification strategy for service delivery in the cloud, taking into account the various needs of users and the complex networking environment. Within this strategy, we provide an improved classification algorithm based on feature similarity by calculating the weights that feature vectors account for in the classification. The probabilities with which a feature item belongs to each category are therefore obtained. The algorithm considers the multidimensional and multiattribute traits of cloud service resources and the correctness of resource classification. Then, using a MapReduce parallel programming model based on the Hadoop open-source cloud platform, we implement the parallelization of our improved classification algorithm. MapReduce simplifies the whole programming process through the separate and aggregated processes of a Map function and a Reduce function for distributed computing tasks. This greatly improves the system's ability to deal with large quantities of cloud service resources. Initial simulation results corroborate the potential of our classification strategy in classifying cloud service resources, which will improve significantly the classification performance on the mass of cloud service resources in a dynamic network. This work may therefore serve as a valuable tool for individuation services and information push in the cloud, thereby offering a high-quality service to cloud users.

ACKNOWLEDGMENT

The authors wish to thank the reviewers for their detailed reviews and constructive comments, which have helped improve the quality of this paper. This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61370221, No. U1204614. And in part by the Plan for Scientific Innovation Talent of Henan Province under Grant No. 124100510006, the Program for Innovative Research Team (in Science and Technology) in University of Henan Province under Grant No. 14IRTSTHN021.

REFERENCES


