An improved TFIDF algorithm based on dual parallel adaptive computing model

Yuwan Gu, Yaru Wang, Juan Huan, Yuqiang Sun and Shoukun Xu*

School of Information Science and Engineering,
Changzhou University,
Changzhou, China
Email: gywss0724@126.com
Email: wyr627290@163.com
Email: huanjuan@cczu.edu.cn
Email: shisungu@126.com
Email: shoukxu@126.com
*Corresponding author

Abstract: The double parallel cloud computing framework based on graphics processing unit (GPU) and MapReduce is proposed. The method aims at the low efficiency for the large data sets on the stand-alone by text categorisation algorithm, constructs the adaptive computation process of double parallel computing and combines the advantage of improved term frequency-inverse document frequency (TFIDF) algorithm, and improves TFIDF text categorisation algorithm with double parallel adaptive computing. In different operating environments, the efficiency of improved TFIDF algorithm will be compared with different computing nodes. The result shows that the improved TFIDF based on dual parallel adaptation has an increase of 6.48% on Macro_F1 compared to the TFIDF based on CPU, and the operating efficiency has increased by nearly seven times. With the number of nodes increasing, the algorithm execution efficiency with double parallel adaptive computing is getting more and more effective.

Keywords: improved TFIDF algorithm; MapReduce; graphics processing unit; GPU; parallel computation.

Reference to this paper should be made as follows: Gu, Y., Wang, Y., Huan, J., Sun, Y. and Xu, S. (2020) 'An improved TFIDF algorithm based on dual parallel adaptive computing model', *Int. J. Embedded Systems*, Vol. 13, No. 1, pp.18–27.

Biographical notes: Yuwan Gu received her BS and MS degrees in Computer Science from Changzhou University, Changzhou, China in 2006 and 2010, respectively, and PhD degree in Agricultural Electrification and Automation from Jiangsu University, Zhenjiang, China in 2016. Currently, she is a Lecturer of Computer Science with School of Information Science and Engineering, Changzhou University. She is a member of China Computer Federation (CCF). Her research interests are parallel algorithm, distributed computing and parallel parsing.

Yaru Wang is a graduate student at the School of Information, Changzhou University, her research interest is image processing. She received her Bachelor's degree in Management from Anhui Polytechnic University in 2016.

Juan Huan received her Bachelor and the Master's degree in Computer Applications Technology from Jiangsu University, Jiangsu, China in 2004 and 2007, respectively. Currently, she is pursuing her PhD degree at the School of Electrical and Information Engineering, Jiangsu University. She is also an Associate Professor with the School of Information Science and Engineering, Changzhou University, Jiangsu, China. From 2014 to 2015, she was a Visiting Scholar supported by Jiangsu government scholarship for overseas studies with University of Texas at Dallas, Dallas, TX, USA. She is a member of China Computer Federation (CCF). Her research interests include intelligent prediction algorithm and wireless networks.

Yuqiang Sun is a Professor of Changzhou University. He received his PhD from College of Science of Xidian University in 2008. He took part in the revision of the *Book of the Design and Analysis of Parallel Algorithm* in 2009. His current research interests in parallel algorithm, distributed computing and parallel parsing.

Shoukun Xu is a Professor of Changzhou University. He received his PhD from China University of Mining and Technology in 2001. His main research interests are artificial intelligence and ubiquitous computing.

This paper is a revised and expanded version of a paper entitled 'An improved TFIDF algorithm based on dual parallel adaptive computing model' presented at CPSCom2018, Halifax, Canada, 30 July–3 August 2018.

1 Introduction

Text categorisation means the texts can be divided into predefined categories by auto text categorisation algorithm (Xu et al., 2008) based on text content to improve the efficiency of text retrieval and text storage. The rapid development of information technology has provided text categorisation with new application platforms. In programming model, Google has put forward the cloud computing, which takes MapReduce parallel computing programming model as the core and is currently widely applied to high performance computing for massive data processing. In hardware, coordinate operation of CPU and graphics processing unit (GPU) has been the mainstream, and with the development of electronic technology, the parallel computing power of GPU has exceeded CPU. Therefore, it is of high practical value and has broad application prospects that massive text data classification helps users access the information quickly by using information technology.

The explosive expanding of data reduces the execution efficiency of text categorisation under traditional platform, while the high performance parallel computing becomes an effective way for text categorisation. Currently some researchers try to realise MapReduce computing framework. For example, R.M.Yoo puts forward Phoenix and Wenbin Fang presents Mars (Fang et al., 2011). On the basis of GPU, in this paper, double parallel computing platform based on MapReduce is set up and adopted by improved term frequency-inverse document frequency (TFIDF) classification algorithm, which is of higher capacity of category division. In this research, improved TFIDF algorithm has been designed and realised to improve the running efficiency on the basis of dual parallel adaptive computing.

2 Text categorisation process

Text categorisation technology is an important part of text mining. It based on text content. Through a guiding learning process, it finds the connection between text features and categories. It classifies the text into predefined categories and realises automatic classification of text. From the view of data, text categorisation is the process of mapping, which is to classify the text into one or more known categories according to the function (Tang et al., 2016). The basic flow of text categorisation shows in Figure 1.

The process of text categorisation is as follows:

1 text preprocessing: classify text in the form of words, words, etc., and delete deactivated words, low-frequency words, etc., and express the text in the form of vectors

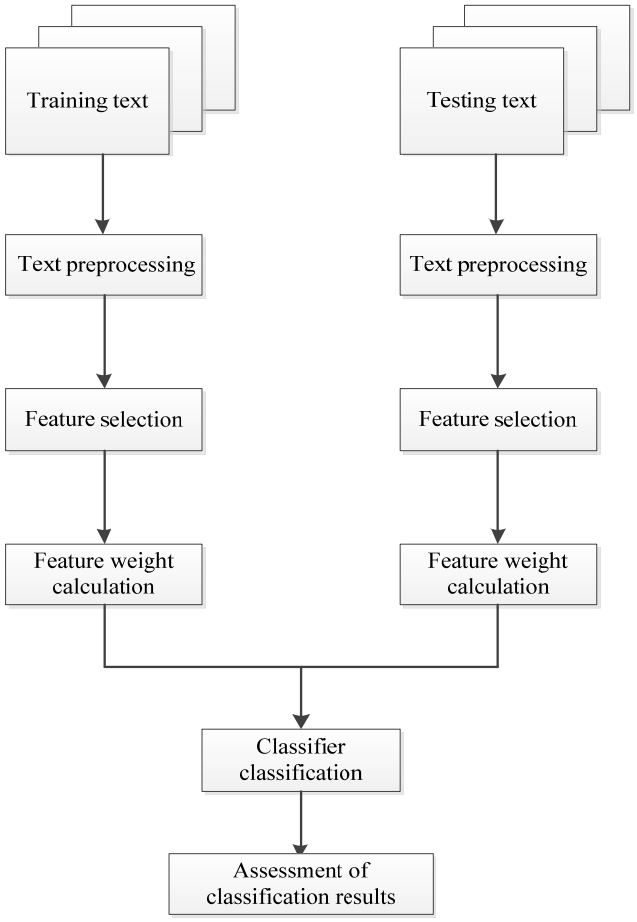
2 feature selection: that is, feature extraction, design feature selection function removes feature words that have little effect on classification, reduces the amount of operation, and reduces the scale of calculation

3 feature weighting: after the feature selection, the feature items are weighted, to give a larger weight to the feature items with strong ability to distinguish

4 classification: classification algorithm is used to classify text sets

5 performance evaluation: it is possible to evaluate the result of text categorisation.

Figure 1 The process of text categorisation



3 Research background

3.1 MapReduce model

MapReduce programming model is a framework model put forward by Google, which can process mass data by parallel processing in the large computer cluster. As a simplified

model of distributed programming, it can divide application programs into many small work units by MapReduce and are concurrently executed when they are sent to large cluster nodes (He et al., 2013). Then the results can be obtained by integrating the intermediate results that each sub node works out. The data processing is accomplished by map and reduce (Wang et al., 2016; Bechini et al., 2016).

In map sections, the data will be input into map function in the form of <key, value>. The function aggregates all values with the same key and generates a set of intermediate data in the form of <key1, value1>.

In reduce sections, <key1, value1> will be received firstly and reduce function disposes intermediate data lists. Then the data with the same key1 are combined in a list. The final result <key2, value2> will be generated eventually.

3.2 GPU

GPU, the microprocessor that processes images, is widely used in the field of graphic processing and 3D graph drawing (Smirnov, 2016). Compared with CPU, GPU has ultrafast computing power in graphical matrix operations and non-graphical parallel numerical value operations. In a complete computer system, CPU and GPU are distributed to different tasks. CPU focuses on digital logical operation and more transistors in GPU are used to process data instead of data cashing and control flow. Therefore, in terms of floating-point arithmetic and parallel computation, the property of GPU is ten times stronger than CPU. In the data parallel computing, GPU is more suitable for high density calculating. In addition, GPU could be easier to reach the higher memory bandwidth than CPU (Benalia et al., 2018), for it adopts simpler memory model.

3.3 Improved TFIDF algorithm

TFIDF algorithm, widely used in the field of information retrieval, is usually applied to feature weighting of data mining, such as text categorisation (Liu et al., 2009). However, traditional TFIDF algorithm cannot correctly calculate the weight of the word appearing frequently in a class. In other words, the traditional algorithm has worse capacity of category division.

In traditional algorithm, a word is suitable for classifying when it appears frequently in a particular article and seldom appears in other texts. The weight of this word in text set will be figured out in accordance with the term frequency (TF) and inverse document frequency (IDF) of the word. The weight ω_k shows as follows:

$$\omega_k = \frac{TF_k \times IDF_k}{\sqrt{\sum_{k=1}^{n} (TF_k \times IDF_k)^2}} \tag{1}$$

TF_k is the frequency of the word k appearing in the text, IDF_k is IDF of k.

$$TF_k = \frac{t_k}{\sum t_i} \tag{2}$$

t_k is the times of the feature word k appearing in the text, $\sum t_i$ stands for the sum of other words in the text, IDF_k is IDF of k.

$$IDF_k = \log \frac{N}{n_k} \tag{3}$$

N is the sum of texts, and n_k is the number of texts which contain k.

From formula (2), we can see that the value of IDF is inversely proportional to the number of texts, which contain k in traditional TFIDF algorithm. It thus shows that the weight is greater when less text contains k. In the above case, k will have a good capacity of text division. However, traditional TFIDF does not consider distribution information of the feature word with different conditions: distributed among different classifications, distributed inside the classification, incompleteness in classification. According to formula (2), the word has a worse capacity of category division when it appears frequently in a text and seldom appears in other texts. However, it is just opposite (Zhang and Zhang, 2011).

Supposing that word k_1 merely appears in texts of classification C_1, k_2 merely in texts of classification C_2, and 1,500 texts in C_1 and 500 texts in C_2, according to calculated result of formula (2), the weight of k_1 is greater than k_2, while the two variables have the same capacity of classification division in accordance with domain knowledge.

$$P(C_1 \mid t_1) = P(C_2 \mid t_2) = 1 \tag{4}$$

Therefore, the paper will adopt the improved TFIDF algorithm, which is improved as follows: if the number of texts n_1 stands for the number of texts containing word k in class C_1, and n_2 is the number of texts containing word k in other classes.

$$n_k = n_1 + n_2 \tag{5}$$

The formula (2) can be converted to:

$$IDF_k = \log \frac{n_1}{n_1 + n_2} N = \log \frac{1}{1 + \frac{n_2}{n_1}} N \tag{6}$$

According to formula (5), we can see that the word k is representative when k appears frequently in the class C_1 and seldom appears in other classes.

4 Improved TFIDF algorithm of dual parallel adaptive computing model

4.1 Dual parallel computing model

Hadoop is adopted as parallel computing frame in this research, which is a distributed parallel computing frame

and mainly composed of Hadoop Distributed File System (HDFS) and MapReduce programming model. Hadoop can be divided into three layers logically, including the client layer, the management node layer and computing node layer. The JobTracker of management node layer starts and dispatches the computing tasks supervised at the same time. The TaskTracker of computing node layer sends task requests to JobTracker and then executes the tasks.

CPU operates and computes the tasks in traditional Hadoop (Aslam et al., 2017; Digalwar et al., 2017) while the tasks are running. In order to improve the operation efficiency of tasks, this research utilises the high performance parallel computing capacity of GPU to structure the double parallel computing model based on GPU and MapReduce. It is mainly divided into calling module of CPU and computing module of GPU. CPU executes commands, including accessing the hard disk, obtaining the physical address, reading and writing files. While massive parallel computing tasks of map and reduce are accomplished by GPU, corresponding to parallel computing of map, sorting operation and parallel computing of reduce, combining operation. The double parallel computing framework (Zhang et al., 2017; Li and Dong, 2012; Wang et al., 2018) is based on GPU and MapReduce, as depicted in Figure 2.

Figure 2 Dual parallel computing model based on GPU and MapReduce

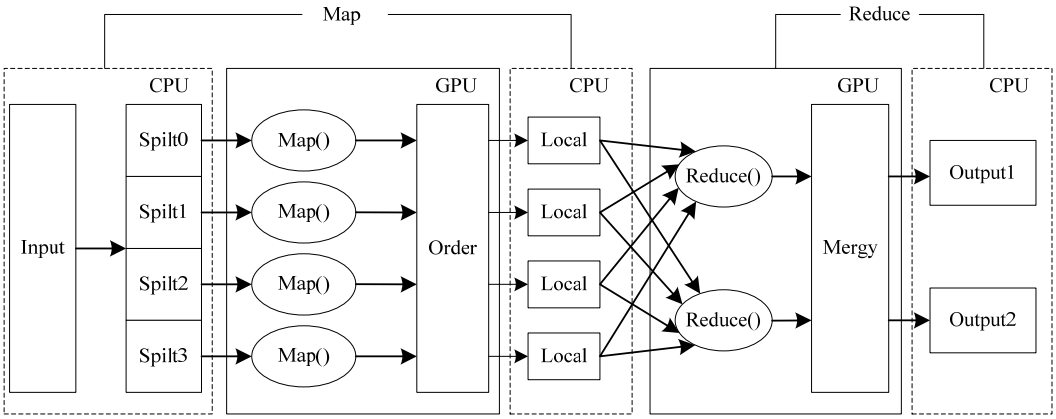
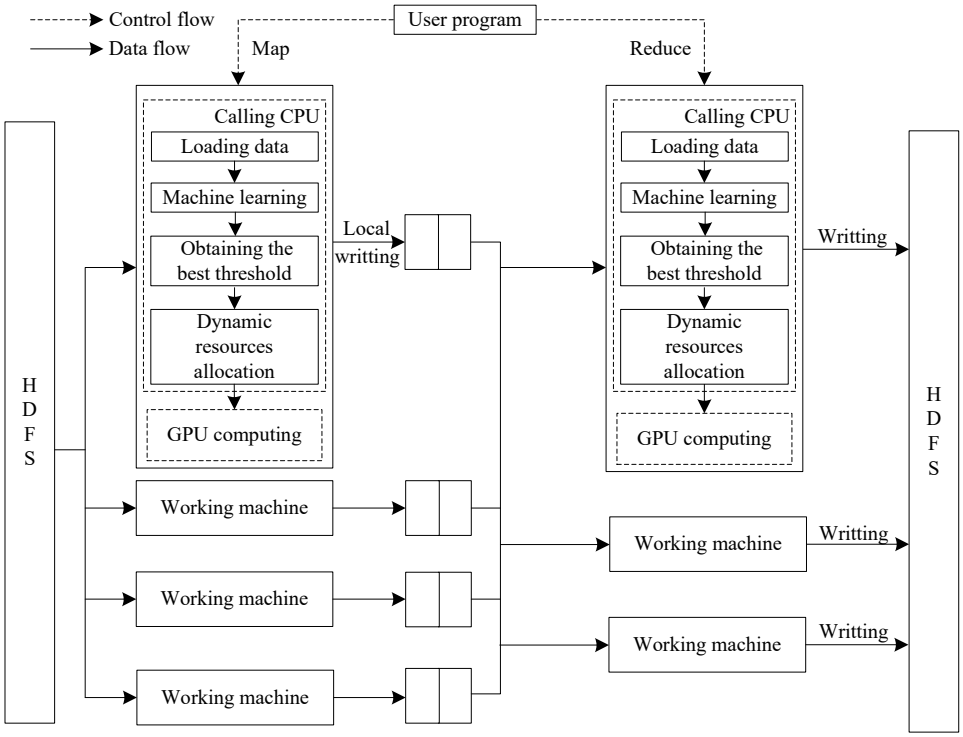


Figure 3 Dual parallel adaptive computing model based on GPU and MapReduce



4.2 Adaptive computing

The number of GPU calculating threads will become too large or too small while CPU allocates the data blocks to GPU (Li et al., 2017; Rathore et al., 2018). That may result in the decline of GPU parallel computing efficiency. We can obtain the GPU computing resources, which are fitted with data blocks to improve the efficiency of GPU parallel computing by designing dynamic data processor.

The design of dynamic data processor means that according to the amount of the tasks allocated several times, the existing computing resources and the present GPU performance, dynamic data processor can find out optimal threshold that conforms to the current computing scale by learning process of machine. The number of threads higher or lower than the adaptive threshold size in calculating threads will be automatically assigned by dynamic processor to structure the best allocation scheme for optimising GPU thread resources (Wang et al., 2015).

The threads of GPU are allocated by the traditional GPU with static mode during resources allocation, while processing the data blocks are transported by CPU (Deshmukh, 2017; Liu et al., 2017). In order to be made full use of, GPU computing resources that each data block needs are calculated by dynamic data processor and the threads of GPU are allocated dynamically. In other words, the GPU computing resources are divided dynamically to improve the efficiency of GPU parallel computing (Trejo-Sánchez et al., 2018).

4.3 Dual parallel adaptive computing model

In this research, we combine GPU with MapReduce and add dynamic data processor (Chen et al., 2018a). The implementation starts from CPU, the calling part of which reads data from HDFS and sends to dynamic data processor after receiving the request of management node. The calculating threads will be allocated dynamically. Then CPU splits GPU computing resources fitted with data block to guarantee GPU parallel computing to operate at full load (Chen et al., 2018b; Liao et al., 2018). GPU generates multiple intermediate key-values by a number of threads execute multiple map operation simultaneously (Rathore et al., 2017). GPU will order the key-values in the light of keyword, and then write to local files. Device memory resources will be released while map tasks are accomplished. In reduce sections, CPU reads the data map submitted and sends to dynamic data processor. The GPU threads are distributed dynamically (Neshatpour et al., 2018). GPU achieves output values of reduce tasks after parallel computing has been accomplished. The results will be obtained by merging operation. Then GPU sends results to HDFS through CPU calling module. Figure 3 is the dual parallel adaptive computing model based on GPU and MapReduce.

4.4 The improved TFIDF processing flow of parallel model

The fundamental principle of Hadoop cloud computing platform is task splitting, parallel computing and that the improved TFIDF algorithm is suited to distributed computation in accordance with weight computing formula. The TF is merely relevant to how much and how often the words appear in the text. The IDF depends on the number of texts. We define t_{kj} as number of times when the certain word k appear in text j. d_j as the number of the words that text j contains. N as the sum of the texts, n_1 as the number of texts, which contain the certain word k in current class and n_2 as the number of texts, which contain the certain word k in other classes. In this paper, four MapReduce processes has been designed to calculate t_{kj}, d_j, n_1, n_2 in TFIDF. Finally, an extra MapReduce process is taken to obtain the computing result of TFIDF. The improved TFIDF processing procedure of parallel model is depicted as Figure 4. It mainly contains the following five steps.

1 Counting the number of words k in the text of current class. The original data is transmitted to map function by dynamic data processor, the output result of Map1 is ((word, text, category), 1), which indicates that a certain word appears one time in the text of class category. The input value of Reduce1 is the output of Map1. After adding up the key-values with same value of (word, text, category). Reduce1 outputs the result.

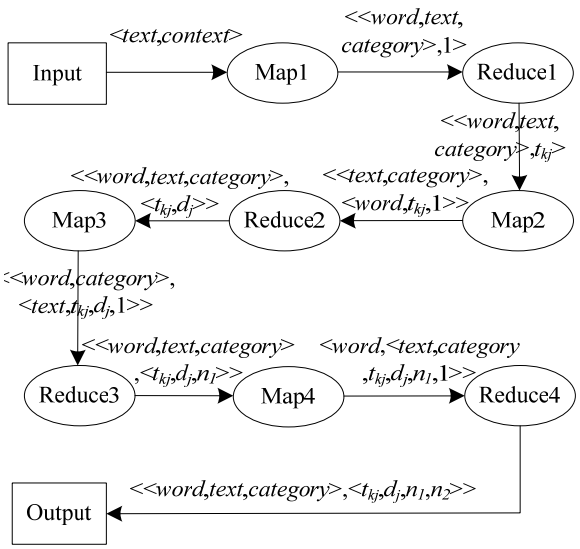
2 Counting the number of all the words in current class. The output result (word, text, category, t_{kj}) of Reduce1 is transformed to (text, category, word, t_{kj}, 1) by Map2. Reduce2 collects the key-values with the same (text, category). The sum of words d_j can be obtained by adding up different words.

3 Counting the number of texts with a certain word in current class. The output result of Reduce2 is transformed to ((word, category), (text, t_{kj}, d_j, 1)) which means that a certain word appears in the text of current class category by Map3. Reduce3 obtains the number n_1 of the text contain k in current class by summing key-values with same (word, category).

4 Calculating the number of texts that contain words in other classes. The output result of Reduce3 is transformed to (word, (text, category, t_{kj}, d_j, n_1, 1)) by Map4. It indicates that the key-values with the same word appear in a text of all classes. The reduce of four collects the key-values with the same word and n_k which is the number of texts contain k will be calculated. Thus, we can obtain n_2, which is the number of texts that contain k in other classes.

5 Computing the final result of improved TFIDF. Transform the three parameters t_{kj}, d_j, n_1 and n_2 to the last MapReduce. Then, the result of ω_k will be achieved by formula (1), (2), (6).

Figure 4 Improved TFIDF process procedure with parallel model



4.5 The improved TFIDF algorithm of dual parallel adaptive computing model

The first MapReduce process to improve TFIDF algorithm of dual parallel adaptive computing model based on GPU and MapReduce completes is as follows:

1 User program sends task requests to job-tracker of management node. The calling part of CPU reads files with text name and content from HDFS after receiving the requests from management node. Then these data will be sent to dynamic data processor. The processor can find the best threshold fitted to current computational scale by machine learning procedure to divide GPU computing resources dynamically. The free task-tracker of CPU formats the data in segment and decomposes them into text (contents).

2 The free task-tracker of GPU sends task requests to job-tracker of management node. According to the input data subset, the corresponding map tasks are set up after receiving the responses. Then a number of threads parallel execute these tasks. task-tracker takes text(contents) as input and scans them. A combiner will be realised by using compute unified device architecture (CUDA) of GPU. Thus intermediate key-values ((word, text, category),1) are output. The partition function hash (key) mod R partitions intermediate <key, value> generated by map function and orders them as (word, text, category). The same data will be aggregated into a new list. Then the list is written to local files. Memory resources of device are released after map tasks are accomplished.

3 The working machines which take charge of reduce tasks are processed by invoking available CPU. The task-tracker of CPU is started to read the data map function submits. Then the data are processed and the GPU thread resources are divided dynamically. The task-tracker of CPU sends each partition to available task-tracker of GPU. The relative processes are done by using the high performance of GPU parallel computing. The number of different words in each text will be figured out by merging the outputs of reduce tasks. Finally, the values of device memory will be copied to host memory.

4 Memory resources of device are released after reduce tasks are accomplished. The task-tracker of GPU receives the results ((word, text, category), t_{kj}) and sends them to CPU calling part. At this point, a MapReduce procedure based on GPU is completed.

5 Evaluation indicators for text categorisation

Text categorisation is a process of text mapping. In general, the performance of text classifier can be evaluated from the point of view of mapping speed and mapping accuracy. The speed of mapping is generally determined by the complexity of text categorisation algorithm, and the accuracy of mapping is compared with the results of text categorisation and manual classification. If the results are closer to the results of manual classification, then the classification has a higher accuracy. The goal of text categorisation system is to create a faster and more accurate text categorisation process.

After the text categorisation is finished, the performance of text categorisation results needs to be evaluated. Classification time can be used to measure the speed of text categorisation, and the accuracy of text categorisation can be reflected by evaluation indicators, including: precision (P), recall (R) and F1 value. Table 1 gives the discriminant table of categories.

Table 1 The discriminant table of classification results

	Number of texts that belong to the class	*Number of texts that do not belong to the class*
Identification of the number of texts that belong to the class	A	B
Identification of the number of texts that do not belong to the class	C	D

The precision (P) refers to the probability that a text is correctly classified by the classifier to a certain category and that the classification is correct. Its calculation formula is as follows:

$$P = \frac{A}{A+B} \tag{7}$$

The recall (R) refers to the probability that a text originally belonged to a category classifier in classifying it to that category. Its calculation formula is as follows:

$$R = \frac{A}{A+C} \tag{8}$$

$F1$ is a new evaluation indicator that can be used for comprehensive evaluation the precision (P) and recall (R). The precision (P) and recall (R) are mutually restricted. If only one of the indicators is increased, the value of the other indicator will be reduced accordingly. A good text categorisation system can comprehensively consider both, so $F1$ was proposed. Its calculation formula is as follows:

$$F1 = \frac{2 \times P \times R}{P + R} \tag{9}$$

Since P, R, and $F1$ determine the effect of text categorisation in a certain category, the results cannot fully reflect the classification situation. Assessing the classification results of data in all categories requires the use of micro-averages and macro-averages.

The process of calculating the micro-averages is as follows: calculate the sum of the number of correct and wrong text in each class, and facilitate the calculation of micro-average P, micro-average R and micro-average $F1$. The specific calculation formula is as follows:

$$Micro\_P = \frac{\sum_{i=1}^{m} a_i}{\sum_{i=1}^{m} a_i + \sum_{i=1}^{m} b_i} \tag{10}$$

$$Micro\_R = \frac{\sum_{i=1}^{m} a_i}{\sum_{i=1}^{m} a_i + \sum_{i=1}^{m} c_i} \tag{11}$$

$$Micro\_F1 = \frac{2Micro\_P \times Micro\_R}{Micro\_P + Micro\_R} \tag{12}$$

The process of calculating the macro-averages is as follows: calculate the P, R, and $F1$ of each category, and then get the average of P, R, and $F1$ of the entire category. The specific calculation formula is as follows:

$$Macro\_P = \frac{\sum_{i=1}^{m} P_i}{m} \tag{13}$$

$$Macro\_R = \frac{\sum_{i=1}^{m} R_i}{m} \tag{14}$$

$$Macro\_F1 = \frac{2Macro\_P \times Macro\_R}{Macro\_P + Macro\_R} \tag{15}$$

The difference between the micro-average and the macro-average is that the text in the text set has the same degree of importance in the micro-average, which aims to reflect the impact of the sub-categories on the whole, while the macro-average holds that each category in all texts is important. Emphasis is placed on explaining the impact of the categories on the whole.

6 Experiments and the result analysis

6.1 The experiments about TFIDF and improved TFIDF

The TFIDF algorithm and the improved TFIDF algorithm are respectively used to calculate the weight of the feature words, and then the KNN classifier is used to classify the test set text. The data were selected from the female, news, business, sports, real estate, financial, entertainment, and IT in SogouCS, 1,200 texts from them are as the training texts for the experiment and 1,500 texts from them are as the test texts. Tables 2, 3, and 4 represent the accuracy, search rate, and $F1$ values of the classification results of the two weighting algorithms under the KNN classifier, respectively.

Table 2 The accuracy of KNN classifier

	TFIDF	*Improved TFIDF*
News	0.7426	0.8096
Entertainment	0.8957	0.9489
Sports	0.8234	0.8786
Financial	0.7929	0.8783
IT	0.8331	0.9068
Business	0.9389	0.9696
Female	0.6725	0.7686
Real estate	0.6569	0.7585
Macro_P	0.7945	0.8624

Table 2 shows that the classification accuracy of the improved TFIDF algorithm has been correspondingly improved, especially the real estate. The macro average accuracy of the improved TFIDF algorithm is 6.56% higher than the TFIDF algorithm, indicating that the improved TFIDF has high classification accuracy.

Table 3 The search rate of KNN classifier

	TFIDF	*Improved TFIDF*
News	0.7122	0.7889
Entertainment	0.9833	0.9264
Sports	0.8097	0.8768
Financial	0.9228	0.9561
IT	0.6502	0.8873
Business	0.8854	0.9387
Female	0.8321	0.8078
Real estate	0.6821	0.8691
Macro_R	0.8097	0.8814

Table 3 shows that the improved TFIDF algorithm has higher search rate than the TFIDF algorithm. Its macro average search rate is 7.06% higher than the TFIDF algorithm, indicating that the improved TFIDF is better than TFIDF on stability.

Table 4 The *F*1 values of KNN classifier

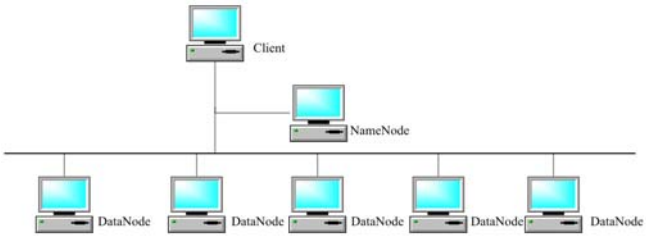
	TFIDF	*Improved TFIDF*
News	0.7271	0.7991
Entertainment	0.8165	0.8777
Sports	0.9375	0.8375
Financial	0.8529	0.9554
IT	0.7304	0.8969
Business	0.8871	0.9441
Female	0.7438	0.7771
Real estate	0.6693	0.8119
Macro_F1	0.7986	0.8758

Table 4 shows that the macro average *F*1 values of the TFIDF and the improved TFIDF algorithm is 79.86% and 87.58%. The improved TFIDF is better than TFIDF on Classification effect.

6.2 Improved TFIDF algorithm in different environments

The experiments are operated in a cluster composed of six computers, each of which is IBM System X3850 X6 server. The CPU is Intel Xeon E7-4809 v2 with six-core processor. The memory of each server is 32G and the operating system is Windows 7. One server is used as host node of Name-Node and job-tracker while the others are slave nodes of Data-Node and task-tracker configured Quadro K4000 GPU of NVIDA and Sun JDK 1.7. The experimental environment shows in Figure 5.

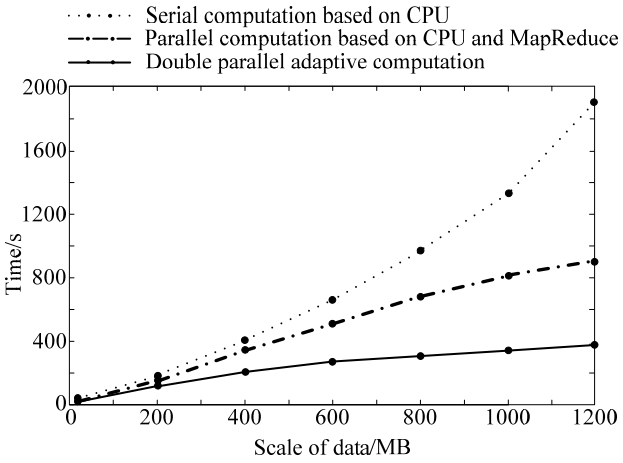
Figure 5 Experiment environment (see online version for colours)



The execution efficiency of improved TFIDF algorithm in different environments is depicted in Figure 6. Three different environments are serial computing of CPU, parallel computing of CPU and MapReduce, dual parallel adaptive computing. The serial computing based on CPU is that the improved TFIDF algorithm is merely operated in a PC and unable to run in distributed and parallel way. Five computers are adopted as computing nodes in two parallel computing environments.

The scale of data is increasing continuously from 10 MB to 1,200 MB in this experiment. As shown in Figure 6, the gap is not obvious between improved TFIDF algorithm based on parallel computing with CPU, map-reduce and the algorithm, which is on the basis of dual parallel adaptive computing model. The execution time is 18.4 seconds. The execution time of serial computation is 21.3 seconds, which is slightly more than other two methods. The consumed time is necessary for the maintenance of parallel platform and network transmission. Thus, the advantage of parallel computing platform for small data set files is not obvious.

Figure 6 Comparison of improved TFIDF algorithm in different environments



With the data size rising, the execution time of serial computation is increasing rapidly. When the scale of data is 1,200 MB, the execution time of improved TFIDF algorithm based on serial computation is 1,942 seconds. The improved algorithm based on MapReduce Cloud computing model needs 908 seconds. However, only 407 seconds is needed in this research. The curve in Figure 4 indicates that the consumed time of improved TFIDF based on dual parallel adaptive computing increases slowly when the data size rises. This experiment shows that improved TFIDF based on dual parallel adaptive computing with massive data set is more effective.

According to the previous experiments, the classification efficiency of the improved TFIDF algorithm has been improved under the dual parallel adaptive computing environment, and the improved TFIDF algorithm greatly improves the accuracy of the text categorisation compared to the TFIDF algorithm. Then the performance of the improved TFIDF algorithm based on dual parallel adaptive computing is verified.

One thousand fifty texts about female, news, business, sports, real estate, finance, entertainment, and IT were randomly selected from SogouCS as training texts for the experiment, and 25,000 texts were selected as test texts. The experiment is compared the improved TFIDF algorithm based on dual parallel adaptive computing and the TFIDF algorithm based on the CPU, and then used the KNN classifier to classify the test set text. The efficiency and accuracy of the text categorisation were evaluated by comparing the macro average accuracy, macro average search rate, and macro average *F*1 values, namely Macro_P, Macro_R, Macro_F1, and the running time of the classification. Table 5 shows the classification results of the experiment.

Table 5 shows that the improved TFIDF based on dual parallel adaptation has an increase of 6.48% on Macro_F1

compared to the TFIDF based on CPU, and the operating efficiency has increased by nearly seven times. Experiments show that improved TFIDF using dual parallel adaptive computing has better feature selection effect and can improve the classification accuracy of text. In addition, it can also improve the overall operating efficiency of the algorithm.

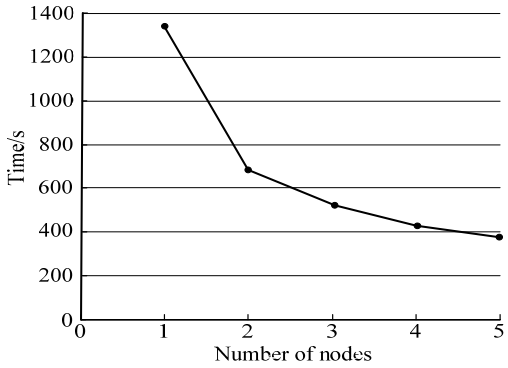
Table 5 Comparison of experimental results

Algorithm	*Macro_P*	*Macro_R*	*Macro_F1*	*Run time*
TFIDF (CPU)	80.08%	81.45%	80.76%	3,259
Improved TFIDF (dual parallel adaptive computation)	86.39%	88.12%	87.25%	389

6.3 The run time under different numbers of nodes

In Figure 7 it is shown that the run time of improved TFIDF algorithm with the number of nodes increases from one to five in the dual parallel platform when the scale of data is 1,000 MB.

Figure 7 The run time of double parallel improved TFIDF



As is depicted in Figure 7, the run time of algorithm will decrease along with the increasing number of nodes in cluster environment. The processing capacity of system with the same data size can be greatly improved by increasing the number of nodes. The run time of improved algorithm is 1,367 seconds with only one node while the run time significantly decreases to 679 seconds with two nodes, and to 383 seconds with five nodes. The experiment indicates that the efficiency of improved TFIDF algorithm with dual parallel adaptive computing model is more effective while the number of nodes the computing time is inversely proportional to is increasing.

7 Conclusions

In this paper, we put forward the conception that Hadoop cloud computing platform is embedded in GPU to improve execution efficiency of text categorisation algorithm under massive data set. The research adopting improved TFIDF algorithm has implemented this algorithm with double parallel adaptive computing model based on GPU and MapReduce. The result of the experiments under different circumstances indicates that based on dual parallel adaptive computing platform, execution efficiency of improved TFIDF algorithm is more effective. Meanwhile, the capacity of improved TFIDF text categorisation algorithm processing massive data is directly proportional to the number of nodes involving in parallel computing. The research will constantly improve the algorithm performance to enhance the efficiency of massive data classification. In this paper, KNN classification algorithm is used to classify the text data with the TFIDF algorithm. Subsequent work will explore the use of other text classification algorithm to implement the classification of the algorithm with the dual parallel platform.

Acknowledgements

Supported by The National Natural Science Fund (61640211and 61803050).

References

Aslam, A., Ahmad, N., Saba, T., Almazyad, A.S., Rehman, A., Anjum, A. and Khan, A. (2017) 'Decision support system for risk assessment and management strategies in distributed software development', *IEEE Access*, Vol. 5, No. 12, pp.20349–20373.

Bechini, A., Marcelloni, F. and Segatori, A. (2016) 'A MapReduce solution for associative classification of big data', *Information Sciences*, Vol. 332, No. 3, pp.33–55.

Benalia, N.E-H., Djedi, N.E., Bitam, S., Ouannes, N. and Duthen, Y. (2018) 'An improved CUDA-based hybrid metaheuristic for fast controller of an evolutionary robot', *International Journal of Embedded Systems*, Vol. 10, No. 5, pp.406–422.

Chen, C., Li, K., Ouyang, A. et al. (2018a) 'FlinkCL: an OpenCL-based in-memory computing architecture on heterogeneous CPU-GPU clusters for big data', *IEEE Transactions on Computers*, Vol. 67, No. 12, pp.1765–1779.

Chen, C., Li, K., Ouyang, A. et al. (2018b) 'Gflink: an in-memory computing architecture on heterogeneous CPU-GPU clusters for big data', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 29, No. 6, pp.1275–1288.

Deshmukh, S. (2017) 'Cuda enabled Hadoop cluster for faster processing', *International Journal of Emerging Technology and Computer Science*, Vol. 2, No. 2, pp.79–88.

Digalwar, M., Gahukar, P., Raveendran, B.K. and Mohan, S. (2017) 'Energy efficient real-time scheduling algorithm for mixed task set on multi-core processors', *International Journal of Embedded Systems*, Vol. 9, No. 6, pp.523–534.

Fang, W., He, B., Luo, Q. et al. (2011) 'Mars: accelerating MapReduce with graphics processors', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 4, pp.608–620.

He, Q., Shang, T., Zhuang, F. et al. (2013) 'Parallel extreme learning machine for regression based on MapReduce', *Neurocomputing*, Vol. 102, No. 2, pp.52–58.

Li, J., Chen, Q. and Liu, B. (2017) 'Classification and disease probability prediction via machine learning programming based on multi-GPU cluster MapReduce system', *The Journal of Supercomputing*, Vol. 73, No. 5, pp.1782–1809.

Li, Y.L. and Dong, J. (2012) 'Study and improvement of MapReduce based on Hadoop', *Computer Engineering and Design*, Vol. 33, No. 8, pp.3110–3116.

Liao, C.L., Lee, S.J., Chiou, Y.S. et al. (2018) 'Power consumption minimization by distributive particle swarm optimization for luminance control and its parallel implementations', *Expert Systems with Applications*, Vol. 96, No. 4, pp.479–491.

Liu, D., Liu, Y., Pei, Z. et al. (2009) 'A feature weighting scheme for text categorization based on feature importance', *Journal of Computer Research and Development*, Vol. 46, No. 10, pp.1693–1703.

Liu, L., Zhang, Y., Liu, M. et al. (2017) 'A-MapCG: an adaptive MapReduce framework for GPUs[C]', *International Conference on Networking, Architecture, and Storage (NAS)*, IEEE, pp.1–8.

Neshatpour, K., Malik, M., Sasan, A. et al. (2018) 'Energy-efficient acceleration of MapReduce applications using FPGAs', *Journal of Parallel and Distributed Computing*, Vol. 119, No. 9, pp.1–17.

Rathore, M.M., Son, H., Ahmad, A. et al. (2017) 'Real-time video processing for traffic control in smart city using Hadoop ecosystem with GPUs', *Soft Computing*, Vol. 22, No. 5, pp.1533–1544.

Rathore, M.M., Son, H., Ahmad, A. et al. (2018) 'Real-time big data stream processing using GPU with spark over Hadoop ecosystem', *International Journal of Parallel Programming*, Vol. 46, No. 3, pp.630–646.

Smirnov, A.V. (2016) 'FIESTA4: optimized Feynman integral calculations with GPU support', *Computer Physics Communications*, Vol. 204, No. 7, pp.189–199.

Suzuki, Y., Yamada, H., Kato, S. et al. (2018) 'Cooperative GPGPU scheduling for consolidating server workloads', *IEICE Transactions on Information and Systems*, Vol. 101, No. 12, pp.3019–3037.

Tang, B., Kay, S. and He, H. (2016) 'Toward optimal feature selection in naive Bayes for text categorization', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 28, No. 9, pp.2508–2521.

Trejo-Sánchez, J.A., López-Martínez, J.L., García, J.O.G., et al. (2018) 'A multi-agent architecture for scheduling of high performance services in a GPU cluster', *International Journal of Combinatorial Optimization Problems and Informatics*, Vol. 9, No. 1, pp.12–22.

Wang, H., Zhu, F., Xiao, B. et al. (2015) 'GPU-based MapReduce for large-scale near-duplicate video retrieval', *Multimedia Tools and Applications*, Vol. 74, No. 23, pp.10515–10534.

Wang, W., Zhu, K., Ying, L. et al. (2016) 'Maptask scheduling in MapReduce with data locality: throughput and heavy-traffic optimality', *IEEE/ACM Transactions on Networking (TON)*, Vol. 24, No. 1, pp.190–203.

Wang, X., Wang, W., Yang, L.T. et al. (2018) 'A distributed hosvd method with its incremental computation for big data in cyber-physical-social systems', *IEEE Transactions on Computational Social Systems*, Vol. 5, No. 2, pp.481–492.

Xu, Y., Li, J., Wang, B. et al. (2008) 'A study on constraints for feature selection in text categorization', *Journal of Computer Research and Development*, Vol. 45, No. 4, pp.596–602.

Zhang, D., Shou, Y. and Xu, J. (2017) 'An improved parallel K-means algorithm based on MapReduce', *International Journal of Embedded Systems*, Vol. 9, No. 3, pp.275–282.

Zhang, Y. and Zhang, D.X. (2011) 'Improved feature weight algorithm', *Computer Engineering*, Vol. 37, No. 5, pp.210–212.