An Extreme Learning Approach to Fast Prediction in the Reduce Phase of a Cloud Platform

Qi Liu[1], Weidong Cai[1], Jian Shen[1(✉)], Baowei Wang[1], Zhangjie Fu[1], and Nigel Linge[2]

[1] Nanjing University of Information Science and Technology, 219 Ningliu Road, Nanjing 210044, Jiangsu, China
S_shenjian@l26.com
[2] The University of Salford, Salford, Greater Manchester M5 4WT, UK

Abstract. As a widely used programming model for the purposes of processing large data sets, MapReduce (MR) becomes inevitable in data clusters or grids, e.g. a Hadoop environment. However, experienced programmers are needed to decide the number of reducers used during the reduce phase of the MR, which makes the quality of MR scripts differ. In this paper, an extreme learning method is employed to recommend potential number of reducer a mapped task needs. Execution time is also predicted for user to better arrange their tasks. According to the results, our method can provide fast prediction than SVM with similar accuracy maintained.

Keywords: MapReduce · Extreme learning · Fast prediction

1 Introduction

MapReduce (MR) [1] has become the most popular distributed computing model used in a cloud environment, where large-scale datasets can be handled/processed using map and reduce procedures in the cloud infrastructure transparently. Two types of nodes are maintained in a cluster based on the MR framework; they are JobTracker and TaskTracker nodes. The Jobtracker, which runs on the data node, coordinates MapReduce jobs.

The MR, as well as loud computing has become a hotspot in the academia [2]. Many people try to optimize it. Proposals in [3–6] may predict the execution states of mapreduce, but they cannot precisely predict it. In this paper, a novel prediction model based on the ELM algorithm is proposed to facilitate the execution of reduce operations in a cloud environment.

The rest sections are organized as followed. Related work is given in Sect. 2, followed by Sect. 3, where our prediction approach is detailed. In Sect. 4, testing environment and corresponding scenarios are design for the verification and evaluation. Finally, conclusion and future work are discussed in Sect. 5.

Z. Huang et al. (Eds.): ICCCS 2015, LNCS 9483, pp. 417–423, 2015.
DOI: 10.1007/978-3-319-27051-7_35

2 Related Work

Offline or online profiling has been proposed by previous work to predict application resource requirements by using benchmarks or real application workloads. Wood et al. [3] designed a general approach to estimate the resource requirements of applications running in a virtualized environment. They profiled different types of virtualization overhead and built a model to map file in the local system into the virtualized system. Their model focused on relating the resource requirements of real hardware platform to the virtual one. Islam et al. [4] studied the changing workload demands by starting new VM instances, and proposed a prediction model for adaptively resource provisioning in a cloud. Complex machine learning techniques were proposed in [5] to create accurate performance models of applications. They estimated the usage state of resource by an approach named PQR2. Jing et al. [6] presented a model that can predict the computing resource consumption of MapReduce applications based on a Classified and Regression Tree.

3 A Prediction Model Based on NO-ELM

Artificial neural networks (ANNs), as an effective method have been widely applied in applications involving classification or function approximation [7]. However, the training speed of ANNs is much slower than what a practical application needs. In order to overcome this drawback, the approximation capability of feed is employed to advance neural networks, especially in a limited training set. One of the most important achievement of this work is putting forward a novel learning algorithm in single hidden layer feed forward neural network (SLFNs) [8], i.e. ELM [8–13].

3.1 Number of Hidden Neurons Optimized ELM (NO-ELM)

In the basic ELM algorithm, the number noted as L, is usually generated through iterating. To find the min $RMSE$ or R^2 that is close to 1, L needs to be trained into the best value. However, original method is has the disadvantage that the number may be different through different experiments. An optimized algorithm is therefore introduced to achieve the process, as shown in Algorithm 1.

Algorithm 1. Generate the number of hidden neurons

Input:
TS: size of the training set
IT: times of iteration
RT: times of running

Output: *L*: the number of hidden neurons

Steps:

1. While times of running is smaller than *RT*
2. While times of iteration is smaller than *IT*
3. If the accuracy *Acc* get this time is smaller the *RMSE*
4. RMSE = Acc
5. L_{RT} equals the number of iteration
6. End While
7. If there is no L_{RT} in the Collection <K, V>
8. Add <L_{RT}, 1> to the result Collection <K, V>
9. else
10. Get the number of element *value* in Collection <K, V>
11. Add < L_{RT}, *value*+1> to the result Collection <K, V>
12. End If
13. End While
14. For each L_{RT} in Collection <K, V>
15. Find L_{RT} with the max *value*
16. Record L_{RT} as *L*
17. End For
18. Return *L*

In Algorithm 1, the size of training set, as well as the time of iteration and execution is collected as input parameters. The number *L* is generated as the output.

3.2 The Process to Build the Prediction Model Based on NO-ELM

The building progresses of the prediction model for the number of reducers and the execution time are as follows:

Step 1: **Data preprocessing.** First, samples that may contain great network congestion need to be removed. Then, the refined datasets will be split into training samples and test samples. The training samples are used for training prediction model and test samples are used to check if the prediction model has been well trained.

Step 2: **Model training.** To build the prediction model, training parameters of the model are obtained by using the training samples generated in Step1. The Specific processes include:

(a) randomly generate the weights between input layer and hidden layer, where hidden layer neurons w and the threshold b are set;
(b) calculate the output matrix H *of* hidden layer;
(c) work out output layer weights.

Step 3: **Data validation.** Use the data generated in Step 1 to validate the NO-ELM prediction model. According to the parameters trained in step 2 to get the predictive value of test set, and compare with the actual value to verify prediction performance of the model.

For the model to predict the number of reducers, the data format is set as *{reducer_no, execution_time, input_data_vol}*. Under the default circumstance, the prediction model recommends the number of reducers that can complete the task as soon as possible. The input format can then be simplified as *{reducer_no, input_data_vol}*. If the complete time of a task needs to be specified, the prediction model will recommend corresponding number of reducers. For doing that, the input format is as *{execution_time, input_data_vol}*.

4 Experiment and Analysis

In order to test the performance the new prediction model, a practical Hadoop environment was built consisting of personal computers and a server. Each personal computer has 12 GB of memory, a single 500 GB disk and dual-core processors. The server is equipped with 288 GB of memory, a 10 TB SATA driver. Eight virtual instances are therefore created in the server with same specification as personal computers, i.e. the same amount of memory and storage space, as well as the same number of processors. In terms of role configuration, the server suns as the name node, whilst the virtual machines and personal computers run as the data nodes.

A shared open dataset [6] was manipulated as the input workload containing 26 GB of text files. The dataset was further separated into 5 groups for testing purposes. A K-means (KM) clustering algorithm provided by Purdue MR Benchmarks Suite was used for partitioning operation in the cloud platform.

Before training the NO-ELM prediction model, the samples are prepared following the equation below in order to meet the requirement of the model:

$$h_t = (s_t - \bar{s})/(\bar{s}) \tag{1}$$

where $\bar{s}$ is the mean value of sample series, s_t is the value of one sample. Here, we remove s_t from the samples if h_t is greater than 5 % and s_t is greater than $\bar{s}$ considering the cases where these samples may be affected by the network congestion.

The sample data are then normalized following the equation below:

$$s_t = (s_t - s_{\min})/(s_{\max} - s_{\min}) \tag{2}$$

where $s_{\min}$ is the minimum value of sample series, $s_{\max}$ is the maximum value of samples. After normalization, the variation range of sample data is [0, 1].

In order to keep the generality, experiments in all performance evaluation parameters were run 50 times to get the average value. All the experiments bellow were operated under the circumstances that reduce tasks started when map tasks had finished using "Sigmoidal Function" as activation function. To verify the performance of the NO-ELM prediction model, we compare the predicted values of the NO-ELM prediction model with the test set samples (real values) and the SVM model.

4.1 NO-ELM for Predicting the Number of Reducers

The input data size varies from 1 GB to 23.5 GB, while the number of reducers is selected from 4 to 8. As seen in Fig. 1, the predicted values generated by NO-ELM show a better trend following the real results than the SVM.

In Table 1, 12 groups of the training time are depicted running the application with NO-ELM and SVM consumed, where the NO-ELM consumes less time than SVM in the train stage.

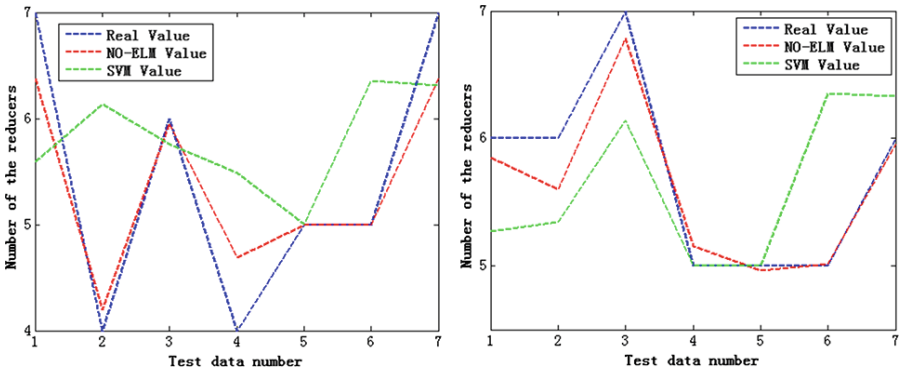

Fig. 1. Experiment comparison for prediction model on the number of reducers

Table 1. Comparison between NO-ELM and SVM in training time

	1	2	3	4	5	6	7	8	9	10	11	12
NO-ELM (ms)	32	47	36	45	54	32	33	43	47	40	41	39
SVM (ms)	384	446	415	394	363	347	407	396	341	387	471	468

4.2 NO-ELM for Predicting Execution Time

In this section, two samples are prepared in each group for training and testing purposes, as shown in Table 2. The simulation results are depicted in Fig. 2.

Table 2. Comparison between NO-ELM and SVM in training time

Group no.	Input data size	Number of split datasets	Number of training set	Number of test set
1	1 GB	68	66	2
2	2 GB	5	3	2
3	5 GB	63	61	2
4	8.5 GB	56	54	2
5	10 GB	15	13	2
6	12.5 GB	57	55	2
7	16 GB	53	51	2
8	17 GB	16	14	2
9	19.5 GB	54	52	2
10	23.5 GB	62	60	2
11	25 GB	8	6	2
12	26.5 GB	67	65	2

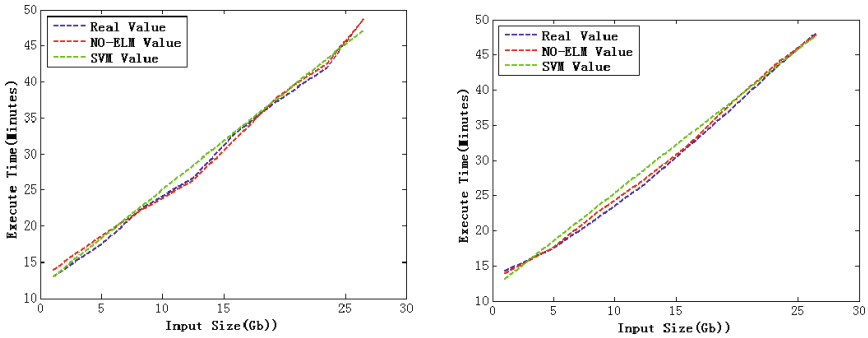

Fig. 2. Experiment comparison for prediction model of execution time

5 Conclusion

In this paper, an extreme learning machine with the number of hidden neurons optimized (NO-ELM) has been introduced to analyze and predict the data. The NO-ELM method has been implemented in a real Hadoop environment, where the SVM algorithm has also been replicated for comparison purposes. Through the results, the NO-ELM has depicted better performance in the prediction of execution time and the number of reducers to be used.

Acknowledgement. This work is supported by the NSFC (61300238, 61232016, U1405254, 61373133), Basic Research Programs (Natural Science Foundation) of Jiangsu Province (BK20131004), Scientific Support Program of Jiangsu Province (BE2012473) and Suzhou City (SYG201315), and the PAPD fund.

References

1. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
2. Fu, Z., Sun, X., Liu, Q., Zhou, L., Shu, J.: Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing. IEICE Trans. Commun. **E98-B**(1), 190–200 (2015)
3. Wood, T., Cherkasova, L., Ozonat, K., Shenoy, P.D.: Profiling and modeling resource usage of virtualized applications. In: Issarny, V., Schantz, R. (eds.) Middleware 2008. LNCS, vol. 5346, pp. 366–387. Springer, Heidelberg (2008)
4. Islam, S., Keung, J., Lee, K., Liu, A.: Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener. Comput. Syst. **28**(1), 155–162 (2012)
5. Matsunaga, A., Fortes, J.A.B.: On the use of machine learning to predict the time and resources consumed by applications. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 495–504. IEEE Computer Society (2010)
6. Piao, J.T., Yan, J.: Computing resource prediction for MapReduce applications using decision tree. In: Sheng, Q.Z., Wang, G., Jensen, C.S., Xu, G. (eds.) APWeb 2012. LNCS, vol. 7235, pp. 570–577. Springer, Heidelberg (2012)
7. Oong, T.H., Isa, N.A.: Adaptive evolutionary artificial neural networks for pattern classification. IEEE Trans Neural Networks **22**, 1823–1836 (2011)
8. Huang, B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**, 489–501 (2006)
9. Samat, A., Du, P., Liu, S., Li, J., Cheng, L.: E2LMs: ensemble extreme learning machines for hyperspectral image classification. IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens. **7**(4), 1060–1069 (2014)
10. Bianchini, M., Scarselli, F.: On the complexity of neural network classifiers: a comparison between shallow and deep architectures. IEEE Trans. Neural Netw. Learn. Syst. 1553–1565 (2013)
11. Wang, N., Er, M.J., Han, M.: Generalized single-hidden layer feedforward networks for regression problems. IEEE Trans. Neural Netw. Learn. Syst. **26**(6), 1161–1176 (2015)
12. Giusti, C., Itskov, V.: A no-go theorem for one-layer feedforward networks. IEEE Trans. Neural Netw. **26**(11), 2527–2540 (2014)
13. Huang, G., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. IEEE Trans. Syst. Man Cybern. B Cybern. **42**(2), 513–529 (2012)