Computers, Materials & Continua CMC, vol.63, no.2, pp.979-993, 2020

Massive Files Prefetching Model Based on LSTM Neural Network with Cache Transaction Strategy

Dongjie Zhu[1], Haiwen Du[6], Yundong Sun[1], Xiaofang Li[2], Rongning Qu[2], Hao Hu[1], Shuangshuang Dong[1], Helen Min Zhou[3] and Ning Cao[4, 5, *]

Abstract: In distributed storage systems, file access efficiency has an important impact on the real-time nature of information forensics. As a popular approach to improve file accessing efficiency, prefetching model can fetches data before it is needed according to the file access pattern, which can reduce the I/O waiting time and increase the system concurrency. However, prefetching model needs to mine the degree of association between files to ensure the accuracy of prefetching. In the massive small file situation, the sheer volume of files poses a challenge to the efficiency and accuracy of relevance mining. In this paper, we propose a massive files prefetching model based on LSTM neural network with cache transaction strategy to improve file access efficiency. Firstly, we propose a file clustering algorithm based on temporal locality and spatial locality to reduce the computational complexity. Secondly, we propose a definition of cache transaction according to files occurrence in cache instead of time-offset distance based methods to extract file block feature accurately. Lastly, we innovatively propose a file access prediction algorithm based on LSTM neural network which predict the file that have high possibility to be accessed. Experiments show that compared with the traditional LRU and the plain grouping methods, the proposed model notably increase the cache hit rate and effectively reduces the I/O wait time.

Keywords: Massive files, prefetching model, cache transaction, distributed storage systems, LSTM neural network.

1 Introduction

For law enforcement agencies and other digital forensic practitioners, the distributed

[1] School of Computer Science and Technology, Harbin Institute of Technology, Weihai, 264209, China.

[2] School of Science, Harbin Institute of Technology, Weihai, 264209, China.

[3] School of Engineering, Manukau Institute of Technology, Auckland, 2241, New Zealand.

[4] College of Mathematics and Computer Science, Xinyu University, Xinyu, 338004, China.

[5] College of Information Engineering, Sanming University, Sanming, 365004, China.

[6] School of Astronautics, Harbin Institute of Technology, Harbin, 150001, China.

*Corresponding Author: Ning Cao. Email: ning.cao2008@hotmail.com.

Received: 01 March 2019; Accepted: 27 November 2019.

storage system must meet the high concurrency needs of multi-user and multi-application situations since there is a need of timely acquisition and preservation of data from cloud storage [Jonas, Pu, Venkataraman et al. (2017); Quick and Choo (2013)]. As the data storage medium of the cloud computing platform, distributed storage system is widely used in internet companies like Facebook, Netflix, Yahoo and Amazon to manage the massive unstructured data [Bende and Shedge (2016)]. Storage systems like HDFS [Shvachko, Kuang, Radia et al. (2010)] and Weil et al. [Weil, Brandt, Miller et al. (2006)] can provide high-performance file access in most instances. However, there exists massive amount of small multimedia data and new small multimedia data keeps growing exponentially. Until 2017, Spotify and Yahoo store about 50% of small files with a size less than 1 MB and the number of I/O operations on small files accounts for about 50% of the total number of I/O operations [Niazi, Ismail, Haridi et al. (2017)]. Distributed storage systems meet serious performance problems when processing massive small files. For example, HDFS can put excessive pressure on Namenode when index massive small files [Sheoran, Sethia and Saran (2017)]. As for Openstack Swift, it creates high CPU and I/O load when assigning updates in this situation [Gracia-Tinedo, Sampé, Zamora et al. (2017)]. It will seriously affect the file access efficiency of distributed storage systems.

Researchers have done a lot of works in optimizing the read performance of massive small files in distributed storage systems. [Cassell, Szepesi, Summers et al. (2018)] shows that prefetching technology can reduce CPU load and increase storage system concurrency. However, the prefetching technique can reduce response delay only when the number of prefetched files far exceeds the user's reading demand. Otherwise, the computational overhead of the prefetch operation itself will offset or even exceed the overhead saved on I/O [Li, Shen and Papathanasiou (2007)]. Therefore, the model for prefetching in units of files does not apply in this situation. Consequently, a group based prefetching model that merge the files into groups and prefetch the files in the unit of groups is proposed by Zhu et al. [Zhu, Du, Qiao et al. (2018)], which can solve the above problems effectively. Grouping model needs to calculate the similarity between files. However, the model will consume too much computing resources in massive small files situation. Therefore, it is necessary to design an efficient and accurate file group access.

In this paper, we propose a massive files prefetching model based on long short-term memory (LSTM) neural network [Sundermeyer, Schlüter and Ney (2012)] with cache transaction strategy. Firstly, we propose a file partitioning algorithm based on temporal locality and spatial locality, which divides files into file blocks according to spatial space locality and number of accesses. Secondly, we propose a file block feature extracting algorithm based on cache transactions. Lastly, we propose a file access prediction algorithm based on LSTM neural network, which takes the file block feature as the input of LSTM neural network and output the file block that have high possibility to be accessed at the next moment.

2 Related work

2.1 File block partitioning

Most file relevance calculation methods use the file as basic unit and the degree of association is calculated by defining the distance between files [Tamersoy, Tamersoy and

Chau (2014); Wildani and Miller (2016)]. However, there are billions of files on a distributed storage system, each file to be analyzed needs to be calculated once with each other. For n files, since the time complexity is $O(n^2)$, the calculation takes a long time. Moreover, the relationship between files is time-sensitive, the strength of the relationship will gradually decrease over time [Xiao, Wang, Liu et al. (2018)]. Therefore, there is a need for an efficient way to determine the relationship between files.

We conducted a statistical analysis of the access of the files and found that the files have obvious spatial and temporal locality characteristics, as shown in Fig. 1. It can be seen that for most correlated file, the total number of accesses and the disk offset where they are saved is alike. We call these files have OA Similarity (Offset-Access Similarity). Therefore, we propose a cluster-based method for preliminary block partitioning of files with OA Similarity and the partition result block is called OA Block. Since the time complexity of the clustering method is $O(kn)$, the time complexity can be greatly reduced, where k is the number of clusters we specified. In the test dataset, the distribution of raw offset and access times data is shown in Fig. 2. Accordingly, there needs an efficient file block partitioning method according to the offset access distribution.

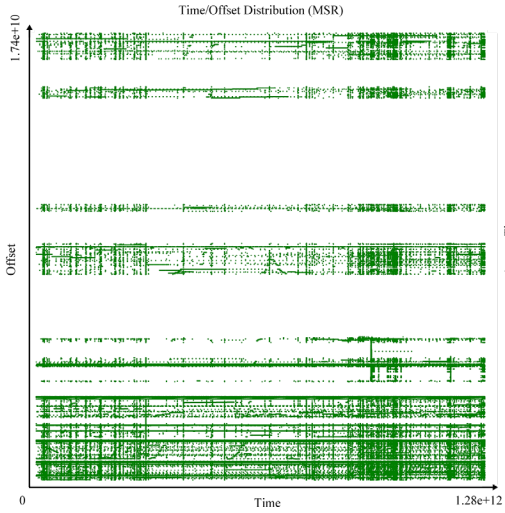


Figure 1: Time / Offset Distribution of trace

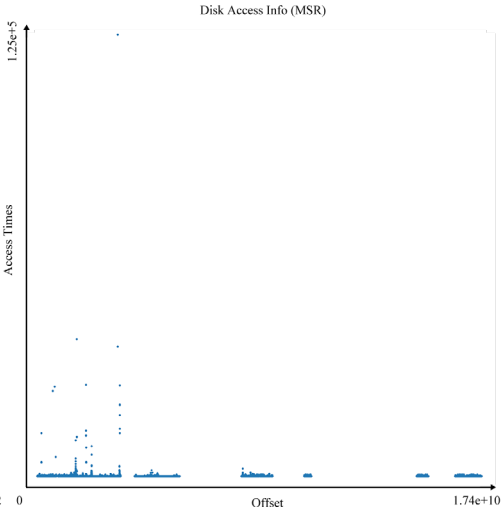


Figure 2: OA Similarity distribution

2.2 Cache transaction and file distance calculation

OA Similarity cannot fully reflect the association between files since the number of access can only represent a statistical feature, it lacks features that reflect the temporal locality of the files. Therefore, we still need to mine relationship according to the temporal locality. The traditional classification model generally uses the time interval [Du, Li, Mao et al. (2016)] or the number of file access intervals [Yang, Karimi, Sæmundsson et al. (2017)] as the distance to define the association between files, as shown in Fig. 3.

However, these calculation methods have their inherent limitations. The time interval

based distance calculation methods overlook the relationship between the overall access speed of the file system and the files access time interval. e.g., in the case of a situation which the file system is frequently accessed, two files that are accessed between 20 ms should have lower degrees of association compared with the infrequently accessed file system. The file access intervals based distance calculation method cannot perceive the size of files. Supposing that cache size is 4 MB, files A, B and C with sizes of 1 MB, 3 MB and 1 MB are accessed sequentially showed in Fig. 4, it is clear that file A and C have no possibility to appear in cache together. Therefore, files A and C have no association although there is only 1 interval between their accesses.

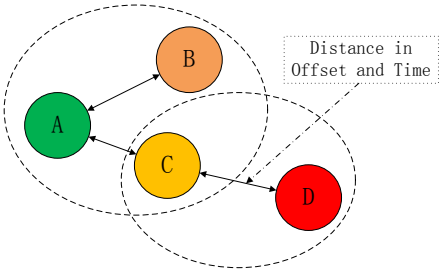


Figure 3: Distance calculation method of traditional classification model

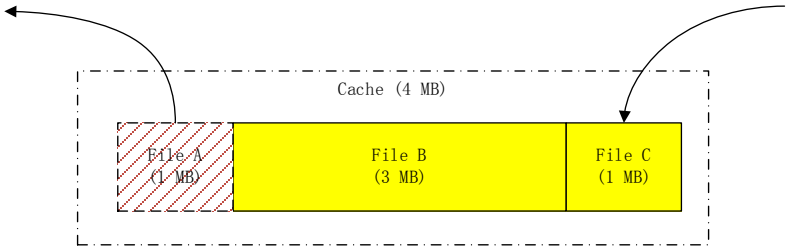


Figure 4: File A is swapped out from cache when file C is accessed

Here, we propose the concept of cache transaction, a cache transaction can be understood as a snapshot of all the files in the cache. As shown in Fig. 5, we assume that the cache transaction space boundary size is M, the cache transaction is recorded every M sized data is accessed. For all files in a cache transaction, one relationship strength is added between each pair of them. This method can minimize the amount of calculations while ensuring that the relationships between files are fully exploited.

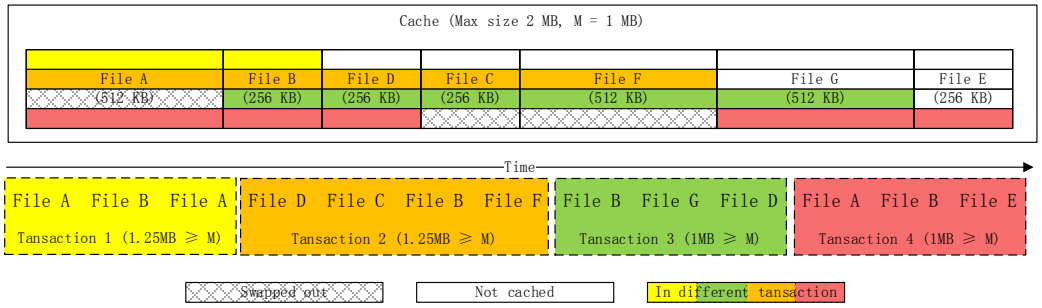


Figure 5: A sample of file transaction calculation process

2.3 LSTM and plain file grouping

File grouping algorithm can organize related files into groups, but the max size of grouped file has a limit in space (generally 1/10 of the cache size). The reason is that when a large file groups is read, excessive files is put to cache. The function of cache will be affected since the files will fill almost the entire cache space. Therefore, the mining result of the file grouping algorithm is a short-term relationship of files that lacks a predictive approach to access trends. In addition, the method we proposed above is a spatial-locality-first partitioning method. The relationships between the files that has a large offset space distance are neglected as shown in Fig. 6.

LSTM neural network is a method for classifying samples using time series data. Compared with the traditional Recurrent Neural Network (RNN) model, the introduced forgetting gate solves the gradient disappearing problem. Therefore, for the timing characteristics of file access, we propose an LSTM-based file access prediction algorithm that uses file block access data to predict the next accessed file block. However, in a distributed storage system, it is difficult to extract features since the file has only Size and Offset info. If one-hot file block representation is used, information such as the number of file occurrences and file access characteristics cannot be indicated. In this study, we propose a file block feature extraction method based on cache transactions to vectorize the feature of file blocks. Using the vectorized file block feature as the input vector of the LSTM, the neural network is trained so that it can output more accurate predict result.

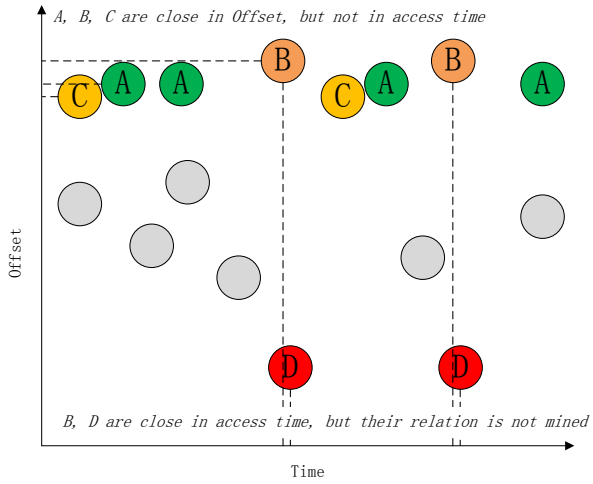


Figure 6: Files B and D should have relation in access time, but they are divided into different file blocks according to disk offset

3 Models design

In this section, we will introduce the designation of proposed model. The calculating method of OA Blocks is detailed in 3.1, the cache transaction based file feature extracting method is introduced in 3.2 and the LSTM based file block predicting model is presented in 3.3.

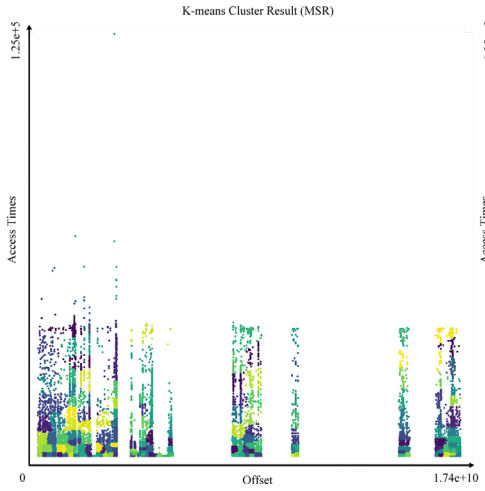
3.1 File blocking algorithm based on spatial and temporal locality

In this module, we propose a spatial locality file blocking algorithm based on OA similarity, which classifies files similar to offset and access Times. For file, we define its coordinate $H_{F_i} = \left(O_{F_i}, A_{F_i}\right)$ where O_{F_i} is disk offset the F_i saved and A_{F_i} is times F_i is accessed in trace. Subsequently, we normalize the points and cluster the files according to the coordinate values.

However, the distribution of the file accesses number exhibits a normal distribution feature, where 80% files having fewer than 50 accesses. If we cluster these file directly, most files will be concentrated in low-access area since that points is denser in that area, so that a large scale low-access files will be grouped in few groups. What's more, for two files F_i and F_j, $O_{F_i} \approx O_{F_j}$, we assume two cases. In the first case, the number of visits $A_{F_i} = 10$ and $A_{F_j} = 100$, in the other case, $A_{F_i} = 1000$ an $A_{F_j} = 1090$. In both cases, F_i and F_j have the same distances. But it is clear that in the second case, the probability of F_i and F_j being associated is greater than in the first case. Therefore, we propose a coordinate scaling method that uses a distance correction function to scale the A coordinate values.

$F_{(i)}$ is file with i^{th} order by access times. The correction function enlarges the difference

degree of the A coordinate value of the file with a low number of access times and reduces the difference degree of the A coordinate of the file with a high number of access times. The result is shown in Figs. 7 and 8.



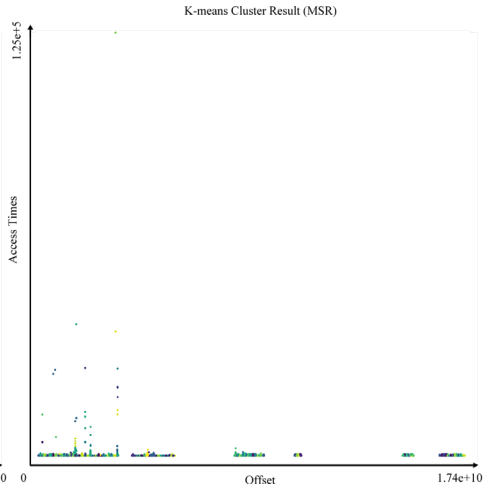


Figure 7: Result before coordinate scaling **Figure 8:** Result after coordinate scaling

3.2 File access relationship classification

According to the concept of the cache transaction we proposed in 2.2, we build an inverted index on the occurrence of files in cache transactions. As shown in Fig. 9.

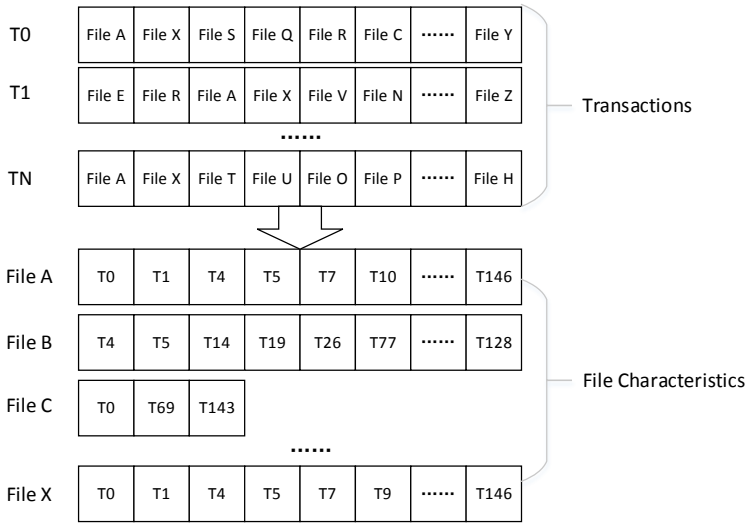


Figure 9: Inverted index for files with cache transaction

We define a vector T to represent the occurrence of a file in transactions.

For file F_a , $T_{F_a}(i)=1$ if F_a occurs in the i^{th} transaction, otherwise $T_{F_a}(i)=0$. We calculate the vector T for the same category of files after the first classification and define the distance D between files, see Eq. (1).

$$D_{F_i,F_j}=\left|T_{F_i}-T_{F_j}\right|*\omega \tag{1}$$

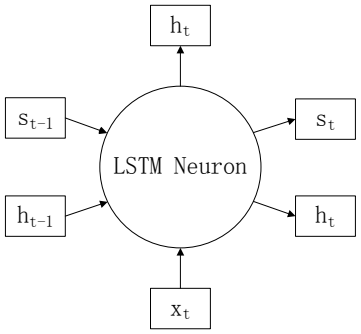


Figure 10: LSTM neuron structure

We propose an improved hierarchical clustering method to classify the vectors F . Due to the errors caused by random fluctuations, files with a large number of access times should be less sensitive to transaction differences. Therefore, we dynamically adjust the clustering stop condition t according to the dimension of T . Finally, we get all file blocks $B=\{B_1,B_2,...,B_m\}$, where m is the total number of file blocks.

3.3 File access prediction algorithm based on LSTM

We propose a file access prediction algorithm based on LSTM neural network. The input vector is the cache transaction feature of the file block.

For cache transaction feature of file blocks, it has same calculation method with cache transaction feature of files. For a file block B_a , $T_{B_a}(i)=1$ if any file $F_x\in B_a$ and F_x occurs in the i^{th} transaction, otherwise $T_{B_a}(i)=0$.

When the t^{th} file block in time sequence B_t is read, the transaction feature of the previous $K-1$ file blocks and the transaction feature of the file block T_{B_t} are combined into an input matrix $E=(T_{B_{t-k+1}},T_{B_{t-k+2}},...,T_{B_t})$ where the transaction feature of B_t is its occurrence in all transactions. Each column of the matrix represents the feature of the file blocks in the time series. The LSTM neural network can learn the temporal access association between file blocks through the learning process. The structure of LSTM neuron is shown in Fig. 10.

The output layer of the LSTM network uses feedforward neural network which maps the intermediate LSTM outputs to a single value, i.e., the next accessed file block and the short-term LSTM based file predicting framework is given as Fig. 11.

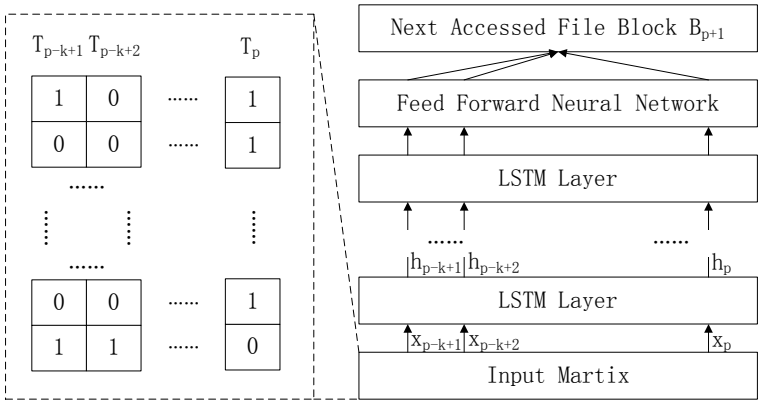


Figure 11: Proposed LSTM based predicting framework

The training process of LSTM based predicting method can be described as following steps as Tab. 1.

Table 1: LSTM training process

Steps	Description
Step 1	Input a new file block B_i .
Step 2	Get the feature of B_i as T_{B_i} according to cache transaction.
Step 3	Input T_{B_i} to LSTM neural network.
Step 4	Check the size of LSTM input number, if size is less than K, go to Step 1.
Step 5	Calculate according to ordinary LSTM neural network layer by layer
Step 6	Predict the next accessed file by a feed forward neural network.
Step 7	Update each of the weights in the network according to ordinary LSTM.

4 Experiment

In this section, we adopt the architecture in Fig. 12 to implement the cache model. The model we proposed runs and is trained on storage server. The storage server reads the disk I/O access logs actively to train the prefetching model. The file reading process of model can be described as following: Firstly, when a new I/O request arrives, the cache model checks if the file is already in the cache. If it is, the file is returned. If the file is not in the cache, all of files that contained in the file block are prefetched into the cache for subsequent requests and the cache transaction feature of file block is put to LSTM neural network for predicting next file block to be accessed. Then the predicted file block is prefetched to cache.

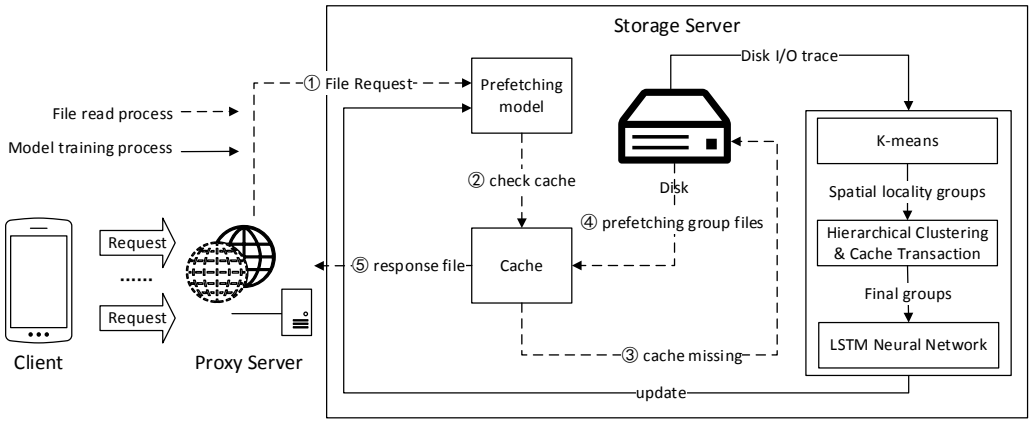


Figure 12: The architecture we implement the entire cache model

In order to prove that our proposed model can be applied to various situations, we use the trace that are available on the Internet and have been tested in similar studies to evaluate the effectiveness of our model.

We write code in Python to test our model. The improved k-means algorithm ran 3 minutes, file block feature extract algorithm took less than 1 minute minutes with optimization and LSTM training process ran 20 minutes on a server with 16-core Xeon E5-2620 v4 processor and 64 GB of RAM. However, we did not use GPU to optimize the LSTM training process since our model runs on storage nodes. The training process may take less time if GPU is used.

The first trace is Florida International University (FIU) [Narayanan, Donnelly and Rowstron (2008)] and traces researchers' local storage. This is a multiuser, multi-application trace, with activities including developing, testing, experiments, technical writing and plotting. There are 17,836,701 access times and over 33% of accesses were to duplicate blocks. The second trace is 1 week of block I/O traces from multipurpose enterprise servers used by researchers at Microsoft Research (MSR) [Koller and Rangaswami (2010)], Cambridge. This dataset was very write heavy. The data format and sample is shown in Tabs. 2 and 3.

Table 2: Format and sample of FIU

Timestamp	PID	Process	LBA	Size	R/W	Maj. Device #	Min. Device #	MD5
0	4892	syslogd	904265560	8	W	0	0	531e779...
39064	2559	kjournald	926858672	8	W	6	0	4fd0c43...
467651	2522	kjournald	644661632	8	W	6	0	98b9cb7...

Table 3: Format and sample of MSR

Timestamp	Type	Block Offset	Size	Response Time
128166372003061629	Read	7014609920	24576	41286
128166372016382155	Write	1317441536	8192	1963
128166372026382245	Write	2436440064	4096	1835

In order to show the superiority of the method we proposed, we compare our model with the ordinary LRU cache replacement method and existing file grouping cache replacement method. Overall, we propose a strategy for file grouping in distributed storage systems. Compared with existing group methods, the cache hit rate has a significant improvement. The results of our experiment are shown in Figs. 13 and 14.

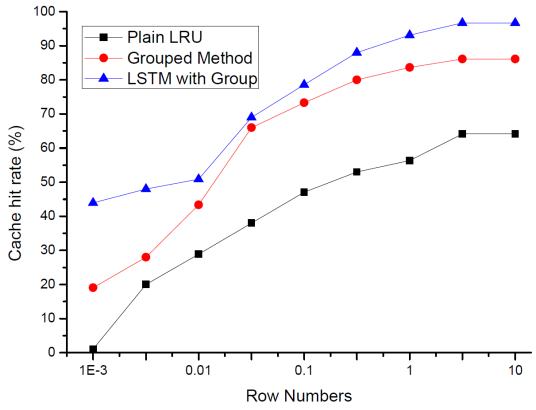


Figure 13: The cache hit rate of the FIU dataset

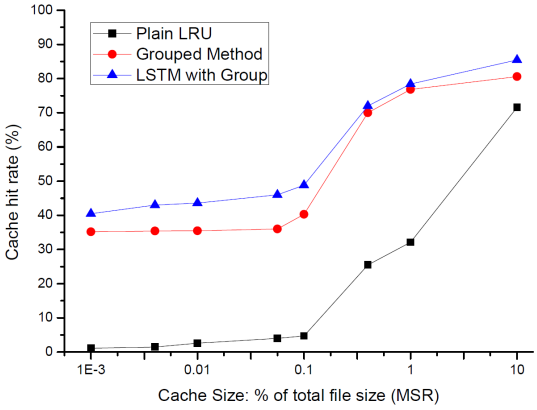


Figure 14: The cache hit rate of the MSR dataset

We use cross-validation to evaluate the number of disk seeks of the proposed algorithm. For existing datasets, our model can be trained in 30 minutes with 1 day access log. Therefore, the training process can be trained at time when the server with low load such as night time. Moreover, the user's access mode to the file may change over time. Here, we divide the access log in one day into a training-validate set. The experimental results show that the proposed model can significantly reduce the number of disk seeks. As shown in Fig. 15.

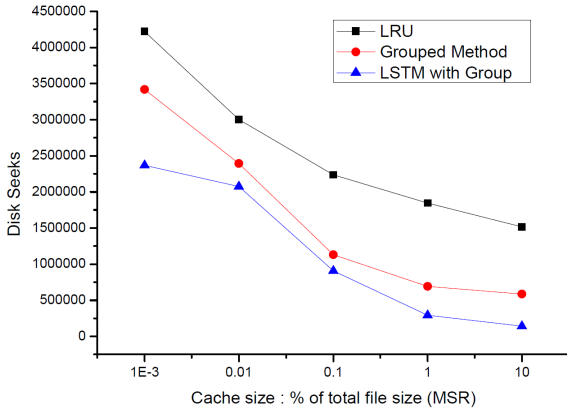


Figure 15: The proposed method provide significant improvement on disk seeks than LRU and group-only based methods

Using prediction based or statistical based method as a lone prefetching model is an unstable approach to manage cache. We compare our model with popular NN-based prefetching method and group only method [Patra, Sahu, Mohapatra et al. (2010)]. The result in Fig. 13 shows that our model performs better in stability and accuracy by average cache hit rate over time with FIU trace.

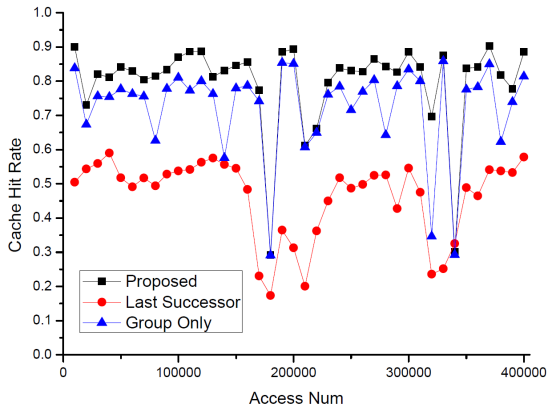


Figure 16: Average accuracies of the cache prefetching models over time

In Fig. 16, we can find that the proposed method provides more stable cache hit rate except in several interval. We analyzed the file accesses in 170,000 to 180,000 and 340,000 to 350,000. The result shows that most accesses in these intervals did not appear in training set.

5 Discussion

The main contribution of our work is the proposal of cache transaction. In this paper, we use the cache transaction to extract feature of files. Compared with traditional file feature extraction methods, the cache transaction based method can provide more information in

file or file access pattern. Furthermore, the cache transaction can find more accurate file relation since it is cache oriented. Files that have no relation on cache transaction will not be regard as related files even though they have less access time or access number interval. In summary, cache transactions can be used as a new way to measure file relationships. It can be widely used in access pattern recognition problem in file systems. Furthermore, it can provide solutions to other types of problems after some transformation.

LSTM neural network is widely used in natural language processing and speech recognition, and can effectively extract the temporal characteristics of data. The innovation of this research is that the input layer of LSTM neural network is optimized by using the cache transaction feature of the file block, so that it can predict the files to be accessed according to the temporal characteristics of the file access. Experiment results demonstrate the feasibility of LSTM neural network for file access prediction problems. Based on the idea of recurrent neural networks such as LSTM, there is still a lot of research space for neuron structure optimization, which can further improve recognition accuracy and training efficiency.

However, clustering operations are performed during the calculation of the model, which will consume a large amount of memory. We have tried 16 million file accesses and 640,000 files. The peak memory usage during the model training process will reach 3 GB. We will improve our work on resource saving in the further research.

6 Conclusion

In this paper, we propose massive files prefetching model based on LSTM neural network with cache transaction strategy to mine file group efficiently. Experiments show that compared with the traditional LRU and the plain grouping methods, the proposed model notably increase the cache hit rate and effectively reduces the I/O wait time. Our model can effectively improve the file access efficiency of distributed storage systems, and thus guarantee the real-time and accuracy in information forensics under massive small file environments.

Acknowledgment: This work is supported by 'The Fundamental Research Funds for the Central Universities (Grant No. HIT.NSRIF.201714)', 'Weihai Science and Technology Development Program (2016DXGJMS15)' and 'Key Research and Development Program in Shandong Provincial (2017GGX90103)'.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Bende, S.; Shedge, R. (2016): Dealing with small files problem in hadoop distributed file system. *Procedia Computer Science*, vol. 79, pp. 1001-1012.

Cassell, B.; Szepesi, T.; Summers, J.; Brecht, T.; Eager, D. et al. (2018): Disk prefetching mechanisms for increasing HTTP streaming video server throughput. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 3, no.

2, pp. 1-7.

Du, S.; Li, C.; Mao, X.; Yan, W. (2016): The optimization of LRU algorithm based on pre-selection and cache prefetching of files in hybrid cloud. *17th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 125-132.

Gracia-Tinedo, R.; Sampé, J.; Zamora, E.; Sánchez-Artigas, M.; García-López, P. et al. (2017): Crystal: software-defined storage for multi-tenant object stores. *Proceedings of the 15th Usenix Conference on File and Storage Technologies*, pp. 243-256.

Jonas, E.; Pu, Q.; Venkataraman, S.; Stoica, I.; Recht, B. (2017): Occupy the cloud: distributed computing for the 99%. *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 445-451.

Koller, R.; Rangaswami, R. (2010): I/O deduplication: utilizing content similarity to improve I/O performance. *ACM Transactions on Storage*, vol. 6, no. 3, pp. 1-13.

Li, C.; Shen, K.; Papathanasiou, A. E. (2007): Competitive prefetching for concurrent sequential I/O. *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 189-202.

Narayanan, D.; Donnelly, A.; Rowstron, A. (2008): Write off-loading: practical power management for enterprise storage. *ACM Transactions on Storage*, vol. 4, no. 3, pp. 1-10.

Niazi, S.; Ismail, M.; Haridi, S.; Dowling, J.; Grohsschmiedt, S. et al. (2017): HopsFS: scaling hierarchical file system metadata using newsql databases. *Proceedings of the 15th Usenix Conference on File and Storage Technologies*, pp. 89-104.

Patra, P. K.; Sahu, M.; Mohapatra, S.; Samantray, R. K. (2010): File access prediction using neural networks. *IEEE Transactions on Neural Networks*, vol. 21, no. 6, pp. 869-882.

Quick, D.; Choo, K. K. R. (2013): Forensic collection of cloud storage data: does the act of collection result in changes to the data or its metadata? *Digital Investigation*, vol. 10, no. 3, pp. 266-277.

Sheoran, S.; Sethia, D.; Saran, H. (2017): Optimized mapfile based storage of small files in hadoop. *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2017, pp. 906-912.

Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. (2010): The hadoop distributed file system. *IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1-10.

Sundermeyer, M.; Schlüter, R.; Ney, H. (2012): LSTM neural networks for language modeling. *Thirteenth Annual Conference of the International Speech Communication Association*.

Tamersoy, A.; Roundy, K.; Chau, D. H. (2014): Guilt by association: large scale malware detection by mining file-relation graphs. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1524-1533.

Weil, S. A.; Brandt, S. A.; Miller, E. L.; Long, D. D.; Maltzahn, C. (2006): Ceph: a scalable, high-performance distributed file system. *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pp. 307-320.

Wildani, A.; Miller, E. L. (2016): Can we group storage? Statistical techniques to identify predictive groupings in storage system accesses. *ACM Transactions on Storage*, vol. 12, no. 2, pp. 1-7.

Xiao, B.; Wang, Z.; Liu, Q.; Liu, X. (2018): SMK-means: an improved mini batch k-means algorithm based on mapreduce with big data. *Computers, Materials & Continua*, vol. 56, no. 3, pp. 365-379.

Yang, J.; Karimi, R.; Sæmundsson, T.; Wildani, A.; Vigfusson, Y. (2017): Mithril: mining sporadic associations for cache prefetching. *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 66-79.

Zhu, D.; Du, H.; Qiao, X.; Liu, C.; Kong, L. et al. (2018): An access prefetching strategy for accessing small files based on swift. *Procedia Computer Science*, vol. 131, pp. 816-824.