Implementation of Skip-Gram-Based File Correlation Analysis For Efficient Large-Capacity Caching in Cloud Storage

K.VISHNU VARDHAN[1], H.ATEEQ AHMED[2]

[1,]PG Student, Department of Computer Science & Engineering,
Dr.K V Subba Reddy Institute of Technology, Kurnool, AP, India.
[2] Assistant Professor, Department of Computer Science & Engineering,
Dr.K V Subba Reddy Institute of Technology, Kurnool, AP, India.

ABSTRACT: Creating a high-capacity cache is vital for enhancing cloud storage accessibility. Compared to conventional data access, cloud storage data access exhibits different patterns, and traditional caching techniques struggle with prefetching and replacing non-frequently accessed data. Various studies indicate that optimizing caching and prefetching strategies in cloud storage can be achieved through file correlation. However, defining file correlations from multiple perspectives is complex, thereby complicating the optimization process. To tackle these challenges, this research introduces a file similarity approach based on skip-gram analysis of user access behaviors. This method enhances prefetching and file replacement in a large-capacity cache by evaluating file correlations. Implementing this approach allows for prefetching and dynamically inserting files into the cache based on their correlations, significantly improving the cache hit rate in simulation benchmarks. Furthermore, the method creates an index table after each training phase, which requires minimal time during online operations. The time complexity for establishing the index during training is O(N * log(V)), and the indexing time complexity is O(1).

1. INTRODUCTION

In the digital age, a tremendous amount of data is being created and stored from various sources, including individuals, governmental bodies, and corporations. As cloud technology continues to advance, more traditional applications are being migrated to cloud-based platforms. However, due to limitations in network setup and backend management, many applications still store data in cloud storage while maintaining the application server as the service access point. This trend is evident across multiple sectors, such as publishing, archiving, and public safety, as depicted in the figure, and represents a shift from traditional to cloud-based application architectures.

Cloud storage systems can offer reliable data access services to users, but their performance is affected by the speed of input/output (I/O) operations. Generally, to improve the average access performance of cloud storage caching systems, enhancements can be made in several areas, such as reducing miss costs and increasing hit rates [1], [2]. A typical method for reducing miss costs in network storage environments involves using a memory hierarchy and implementing multilevel cache strategies. As early as 1990, Muntz and Honeyman applied multilevel caches to distributed file systems [3]. By positioning cache servers between clients and servers to deliver cached file services, this hierarchical caching strategy increased the cache hit rate of distributed storage file systems. Chen, Zhang, and Zhou evaluated different data scheduling algorithms in multilevel caches [4]. Using trace data from existing commercial systems, they compared multilevel caches with cooperative caches and found that layered caches performed

better in increasing cache hit rates. Research mainly focuses on enhancing hit rates by choosing well-known replacement algorithms, such as the widely used LFU, LRU, and ARC policies in caches [5], [6]. According to tests by Cantin and Hill using SPEC2000, the hit rates of LRU and random policies are almost identical when the cache is sufficiently large. However, when the cache capacity is small, LRU has the highest hit rate [7]. Of course, the computational complexity of cache policies is also a factor. For instance, fully associative mapping can lower miss rates even more than the LRU algorithm, but its computational cost is too high for full hardware implementation, making it suitable only for virtual memory. This indicates that operating systems balance between complex software calculations and higher miss rates.

Researchers have discovered that integrating prefetching techniques with caches can significantly boost hit rates [8]. In 1994, Griffioen and colleagues measured the likelihood of two files being accessed simultaneously in a file system based on I/O sequences and implemented prefetching methods to enhance file system access performance. They improved file system access performance by 280% and reduced cache space usage by 50% [9]. To address the complex scenario of sequential and random file access by multiple applications, Li, Varki, and Bhatia predicted I/O sequences generated by various applications and used them in RAID, resulting in significant improvements in RAID access performance [10]. Jouppi suggested using a small sacrifice cache along with prefetching techniques to further increase cache hit rates [11]. Using a small sacrifice cache to store blocks swapped out of the higher-level cache can improve memory hit rates by 20-90%.

Researchers have applied the principle of locality to enhance network data prefetching efficiency. For example, Lee et al. predicted read operations on distributed file systems on the server side and dispatched unread requests to clients [12]. Gong et al. optimized the Ceph system for random read/write operations by caching data only when a user sends a write request and synchronizing with the server through dirty data merging [13]. Du et al. proposed a framework based on data frequency prediction involving data tracing, machine learning modeling, and cache preprocessing and replacement steps [14]. Liu et al. introduced a cache replacement strategy that used user behavior analysis by combining association rule mining techniques and approximate linear computing models to analyze the cache cost of associations among numerous small files and user access patterns, effectively addressing the limitations of general cache policies in accounting for diverse user behaviors [15].

Several researchers have improved prefetching and replacement methods by analyzing data access patterns. For example, Cao and colleagues developed a model to assess the value of web objects in the cache by considering their size and insertion cost, helping to determine which object to remove during a cache miss [16]. Liu and team used direct correlation to minimize directory access latency in network file systems and employed the knapsack model to decide whether to prefetch [17]. Liao and colleagues noted that I/O requests for distributed file systems by applications show a certain degree of repetition within specific periods, predicting data request repetition probabilities to decide on prefetching [18]. Chen and team captured data correlations through text analysis of access records and used prediction algorithms to forecast likely accessed files [19]. Pang and colleagues

proposed a new active data prefetching technique for cloud-based distributed file systems that enhances system performance by improving data prefetching efficiency and reducing communication overhead [18].

To better determine data access correlations, researchers have developed models to characterize correlations between data access requests, such as the access distance model IAD [20], independent reference model IRM [21], and stack distance model SDM [22]. Researchers have also found that cloud data access follows the Zipf-like law [23] and have used this principle to design cache and prefetch systems. Shi et al. proposed a web prefetching and cache model based on Petri nets, combining prefetching with caching to improve the service quality of distributed storage systems [24].

2. EXISTING SYSTEM

According to Cantin and Hill's tests with SPEC2000, the hit rates of LRU and random policies are nearly identical when the cache size is sufficiently large. However, with smaller cache capacities, the LRU policy achieves the highest hit rate [7]. The computational complexity of cache policies is also important. Fully associative mapping, for instance, can reduce miss rates more effectively than the LRU algorithm, but its high computational cost limits its use to virtual memory rather than hardware-implemented caches. This highlights a trade-off in operating systems between complex calculations and lower miss rates.

Researchers have found that integrating prefetching techniques with caches can significantly enhance hit rates [8]. In 1994, Griffioen and colleagues evaluated the likelihood of simultaneous file accesses in a file system using I/O sequences, implementing prefetching methods that improved access performance by 280% and reduced cache space usage by 50% [9]. Addressing the complexities of sequential and random file access by multiple applications, Li, Varki, and Bhatia predicted I/O sequences for various applications in RAID, leading to substantial improvements in RAID access performance [10]. Jouppi suggested further boosting cache hit rates by using a sacrifice cache alongside prefetching techniques [11]. A small sacrifice cache, which stores blocks swapped out of the higher-level cache, can enhance memory hit rates by 20-90%.

Scholars have leveraged the principle of locality to boost network data prefetching efficiency. For example, Lee et al. predicted read operations on distributed file systems from the server side and sent unread requests to clients [12]. Gong et al. optimized the Ceph system for random read/write operations by caching data only upon user write requests and synchronizing with the server through dirty data merging [13]. Du et al. developed a framework based on data frequency prediction, incorporating data tracing, machine learning modeling, and cache preprocessing and replacement steps [14]. Liu et al. introduced a cache replacement strategy that used user behavior analysis, combining association rule mining techniques with approximate linear computing models to comprehensively analyze the cache cost of associations among numerous small files and user access patterns. This strategy effectively addressed the limitations of general cache policies in accommodating diverse user behaviors [15].

Several researchers have refined prefetching and replacement methods by studying data access patterns. For instance, Cao and colleagues created a model to evaluate the value of web objects in the cache by considering their size and insertion cost, helping determine which object to remove

during a cache miss [16]. Liu's team used direct correlation to minimize directory access latency in network file systems, employing the knapsack model to decide on prefetching [17]. Liao and colleagues observed that I/O requests for distributed file systems by applications exhibit a certain degree of repetition within specific periods. They predicted the probability of data request repetition to decide on prefetching [18]. Chen's team analyzed access records through text analysis to capture data correlations and used prediction algorithms to forecast potential file accesses [19]. Pang and colleagues proposed a new active data prefetching technique for cloud-based distributed file systems, improving performance by enhancing prefetching efficiency and reducing communication overhead [18].

To better understand data access correlations, researchers have developed models like the access distance model (IAD) [20], independent reference model (IRM) [21], and stack distance model (SDM) [22]. They have also found that cloud data access follows Zipf-like laws [23] and used these principles to design cache and prefetch systems. Shi et al. proposed a web prefetching and cache model based on Petri nets, combining prefetching with caching to enhance the quality of distributed storage systems [24].

Many scholars have explored deep learning methods for cache prefetching. Hui et al. suggested a file prefetching mechanism using RNN and word embedding technology to capture complex file access patterns [25]. Chen et al. treated data prefetching as a classification problem, employing a neural network-based framework to improve accuracy. Feng et al. introduced a web access prediction and cache prefetching method using a Markov tree, while Lee proposed a memory prefetching method based on semantic similarity and reinforcement learning. Saami et al. used a pruning algorithm based on hardware performance events to identify a concise feature set, subsequently using three learning models to find the optimal prefetch configuration. The LSTM model has gained significant attention, with Chen et al. proposing a data prefetching method, SGDP, based on Stream-Graph neural networks, modeling the LBA Delta sequence. Although more powerful for learning complex patterns, these neural network models have higher computational costs compared to traditional methods.

Currently, there is limited study related to cache replacement strategies based on file correlation that use Skip-Gram and Continuous Bag of Words techniques to correlate files.

3. PROPOSED SYSTEM

We implement a file prefetching framework based on file correlation. Following are important aspects of this implementation:

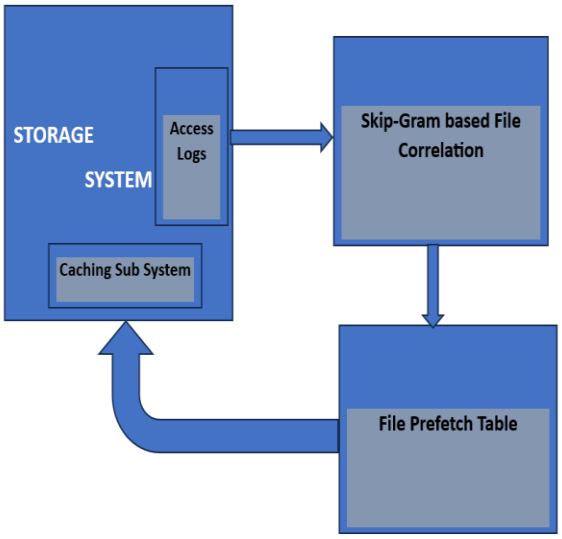
1) Implemented a high-performance cache prefetching mechanism that utilizes only the file correlation mechanism for prefetching. This approach has a runtime complexity of O(1) and imposes minimal performance overhead compared to many machine learning strategies.

2) We implemented a dynamic and similar file cache replacement mechanism that maintains the prefetching hit rate's stability while enhancing the overall cache hit rate.

3) The proposed algorithm is portable and easily integrates with various cache replacement strategies, allowing it to adapt to different runtime scenarios.

CACHING PREFETCHING STRATEGY -- In this approach, during the offline phase, files of interest in the sacrificial zone are first encoded, and both the file relevance vector and the most relevant file table are generated. The most relevant file table then guides file prefetching during the online phase.

4. ARCHITECTURE DIAGRAM



Architecture of Skip-Gram Based File Correlation, Prefetching and Caching

5. IMPLEMENTATION

Training Module

In this module, using storage systems access logs as training data, a Machine Learning model based on Skip-Gram technique, vectorizes the words, then builds a model that correlates files. This module is the crux of this project which is responsible for building machine learning model which is in turn used for predicting file relevancy.

Plotting Module

This module is used for 3D-plotting vectorized file access requests. This module uses Principal Component Analysis(PCA) for reducing high dimensional data to three dimensions so that it can be visualized and insights can be gained in case of trivial data.

Fetch Related Files Module

This module uses the trained model to fetch relevant files of a file and this is in turn used for building file relevancy tables whose entries are useful for prefetching related files so as to increase cache hit rate.

Comparison Module

This module provides a framework for incorporating other cache replacement or file pre-fetching strategy into the application so that it can be compared and contrasted against skip-gram based file correlation analysis.

6. TECHNOLOGY STACK

Python programming language is used for implementing as the ecosystem of libraries useful for carrying out machine learning tasks is very mature. The important libraries used in this implementation are:

Genism – A robust and mature library for natural language processing effective for topic modelling and file similarity. Word2vec call of genism library is the main workhorse in this project that vectorizes file access requests.

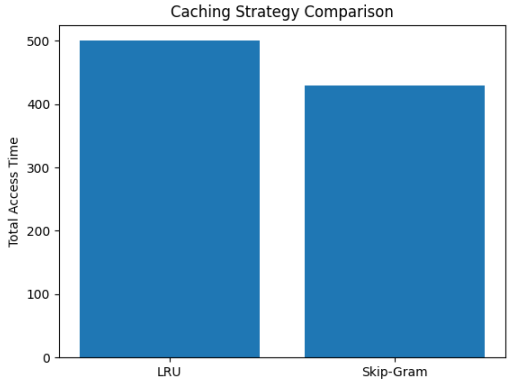
Pandas – A powerful data manipulation and analysis library.

SciKit-Learn – Robust library for various data analysis related tasks and is mainly used in this implementation for Principal Component Analysis.

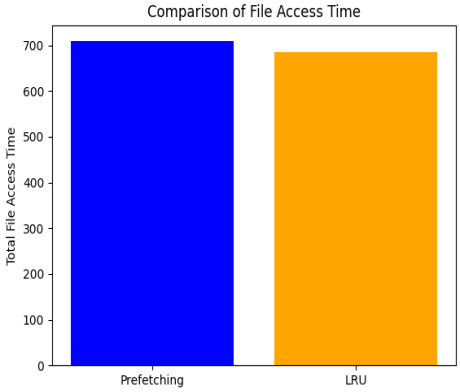
Matplotlib – Library used for creating 3D plots of vectors.

7. RESULTS

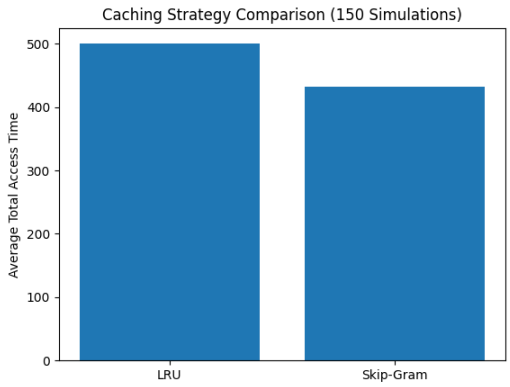
Simulations are carried out to analyze the effectiveness of skip-gram based file correlation analysis for pre-fetching related files and is compared to LRU(Least Recently Used) in the context of file replacement of non-hot data. The data used as file access requests is computer generated random file request data in a hypothetical environment. Few results of simulation are given below



Above picture shows a result from a scenario where skip-gram based file correlation for pre-fetching is better than LRU



Above picture shows a result from a scenario where skip-gram based file correlation for pre-fetching is not better than LRU



Above picture shows results from 150 scenarios where skip-gram based file correlation for pre-fetching is better than LRU

7. CONCLUSION

The Skip-Gram-Based File Correlation Analysis For Efficient Large-Capacity Caching in Storage Systems is implemented. For the randomly generated file access log the Skip-Gram-Based file prefetching strategy for page replacement improved on the traditional LRU page replacement strategy in terms of total file access time and cache efficiency. By leveraging file access patterns and correlations, the Skip-Gram-Based method can more effectively prefetch files that are likely to be accessed soon, reducing the overall cost of file access in large-capacity caching systems. However, it is important to consider the additional computational resources required for training and maintaining the predictive model. This conclusion is limited within the context of computer generated random file access log and the results may not be relevant to real time access logs which inherently has access characteristics favorable to Least Recently Used cache replacement strategies based on principle of locality. This project is limited to comparison of computer generated random test data only. And this project is also limited to non-contextual skip-gram analysis only.

As part of future work skip-gram implementations considering contextual vectorization may be explored

REFERENCES

1] M. Liu, L. Pan, and S. Liu, ‘‘Cost optimization for cloud storage from user perspectives: Recent advances, taxonomy, and survey,’’ ACM Comput. Surv., vol. 55, no. 13s, pp. 1–37, Dec. 2023.

[2] C. B. Tan, M. H. A. Hijazi, Y. Lim, and A. Gani, ‘‘A survey on proof of retrievability for cloud data integrity and

availability: Cloud storage stateof- the-art, issues, solutions and future trends,'' J. Netw. Comput. Appl.,vol. 110, pp. 75–86, May 2018.
[3] D. A. Muntz, P. Honeyman, and C. J. Antonelli, ''Evaluating delayed write in a multilevel caching file system,'' in Distributed Platforms. IEEE, 1996, pp. 415–429.
[4] Z. Chen, Y. Zhang, Y. Zhou, H. Scott, and B. Schiefer, ''Empirical evaluation of multi-level buffer cache collaboration for storage systems,'' in Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst., Jun. 2005, pp. 145–156.
[5] N. Megiddo and D. S. Modha, ''ARC: A self-tuning, low overhead replacement cache,'' in Proc. Fast, vol. 3, 2003, pp. 115–130.
[6] A. J. Smith, ''Cache memories,'' ACM Comput. Surv., vol. 14, no. 3, pp. 473–530, 1982.
[7] J. F. Cantin and M. D. Hill, ''Cache performance for selected SPEC CPU2000 benchmarks,'' ACM SIGARCH Comput. Archit. News, vol. 29, no. 4, pp. 13–18, Sep. 2001.
[8] E. A. Shriver, C. Small, and K. A. Smith, ''Why does file system prefetching work?'' in Proc. USENIX Annu. Tech. Conf., Gen. Track, 1999, pp. 71–84.
[9] J. Griffioen and R. Appleton, ''Reducing file system latency using a predictive approach,'' in Proc. USENIX Summer, 1994, pp. 197–207.
[10] M. Li, E. Varki, S. Bhatia, and A. Merchant, ''Tap: Table-based prefetching for storage caches,'' in Proc. FAST, vol. 8, 2008, pp. 1–16. [11] N. P. Jouppi and A. Eustace, ''Data processing system and method with small fully-associative cache and prefetch buffers,'' U.S. Patent 5 261 066, Nov. 9, 1993.
[12] S. Lee, S. J. Hyun, H.-Y. Kim, and Y.-K. Kim, ''APS: Adaptable prefetching scheme to different running environments for concurrent read streams in distributed file systems,'' J. Supercomput., vol. 74, no. 6, pp. 2870–2902, Jun. 2018.
[13] Y. Gong, C. Hu, Y. Xu, and W. Wang, ''A distributed file system with variable sized objects for enhanced random writes,'' Comput. J., vol. 59, no. 10, pp. 1536–1550, Oct. 2016.
[14] S. Arora and A. Bala, ''An ensembled data frequency prediction based framework for fast processing using hybrid cache optimization,'' J. Ambient Intell. Hum. Comput., vol. 12, no. 1, pp. 285–301, Jan. 2021.
[15] C. Liu, S. Ding, L. Ye, X. Chen, andW. Zhu, ''Cache replacement strategy based on user behaviour analysis for a massive small file storage system,'' in Proc. 14th Int. Conf. Comput. Autom. Eng. (ICCAE), Mar. 2022, pp. 178–183.
[16] P. Cao and S. Irani, ''Cost-aware WWW proxy caching algorithms,'' in Proc. USENIX Symp. Internet Technol. Syst., 1997, vol. 12, no. 97, pp. 193–206.
[17] Z. Liu, F. Dong, J. Zhang, P. Zhou, Z. Xu, and J. Luo, ''A client-side directory prefetching mechanism for GlusterFS,'' in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), Oct. 2016, pp. 3942–3947.
[18] J. Liao, F. Trahay, G. Xiao, L. Li, and Y. Ishikawa, ''Performing initiative data prefetching in distributed file systems for cloud computing,'' IEEE Trans. Cloud Comput., vol. 5, no. 3, pp. 550–562, Jul. 2017.
[19] Y. Chen, C. Li, M. Lv, X. Shao, Y. Li, and Y. Xu, ''Explicit data correlations-directed metadata prefetching method in distributed file systems,'' IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 12, pp. 2692–2705, Dec. 2019.
[20] C. Roadknight, I. Marshall, and D. Vearer, ''File popularity characterisation,'' ACM SIGMETRICS Perform. Eval. Rev., vol. 27, no. 4, pp. 45–50, Mar. 2000.

[21] S. Vanichpun and A. M. Makowski, ‘‘The output of a cache under the
independent reference model: Where did the locality of reference go?’’ in Proc. Joint Int. Conf. Meas. Modeling Comput. Syst., Jun. 2004, pp. 295–306.
[22] L. Cherkasova and G. Ciardo, ‘‘Characterizing temporal locality and its impact on web server performance,’’ in Proc. 9th Int. Conf. Comput. Commun. Netw., 2000, pp. 434–441.
[23] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, ‘‘Web caching and Zipf-like distributions: Evidence and implications,’’ in Proc. Conf. Comput. Commun. 18th Annu. Joint Conf. IEEE Comput. Commun. Soc.
Future Now (IEEE INFOCOM), vol. 1, Mar. 1999, pp. 126–134.
[24] L. Shi, Y. Han, X. Ding, L. Wei, and Z. Gu, ‘‘SPN model for web prefetching and caching,’’ in Proc. 1st Int. Conf. Semantics, Knowl. Grid, Nov. 2005, p. 24.
[25] H. Chen, E. Zhou, J. Liu, and Z. Zhang, ‘‘An RNN based mechanism for file prefetching,’’ in Proc. 18th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci. (DCABES), Nov. 2019, pp. 13–16.