ISSN 2229-5518

Prefetching Algorithm for Layered Storage

Mr. Shivakash Sahu, Mrs. Roshni Dubey

Abstract: The goal is to motivate on challenging the immediate character of the currently used replacement algorithm, Least Recently Used. Furthermore, achievements in former researches provide the motivation for replacing the algorithm with a proactive one. The concept is called prefetching, meaning that the algorithm fetches files to store on primary storage before (therefore 'pre-') a user has requested them.

Here first we start with LRU and then by challenging the Least Recently Used algorithm to be used in the current situation. Subsequently, related literature is used to motivate the research towards a prefetching algorithm based on data mining results. Furthermore, it states what this research contributes to former researches. Then we introduce the research questions. Afterwards description of which methodologies are used in order to answer these research questions.

1 LRU-K

Here we are talking about a self-dependent page-replacement algorithm which has been derived from classical Least Recently Used (LRU). It was projected for management of buffer areas in database management systems. Here both regency and frequency information are integrated to make replacement decisions. In this algorithm the page is dropped from the buffer which from a long time has not been accessed, and on requirement of a new buffer, it limits itself to only the time of the last reference. Particularly, distinguish between often and least referenced pages in case of LRU is not well, until and unless a lot of resources of the system has been wasted in keeping of infrequently referenced pages in the buffer for an extensive period. Best performance of LRU-K algorithm can be proved among all replacement algorithms that are solely based on stochastic information about past references.

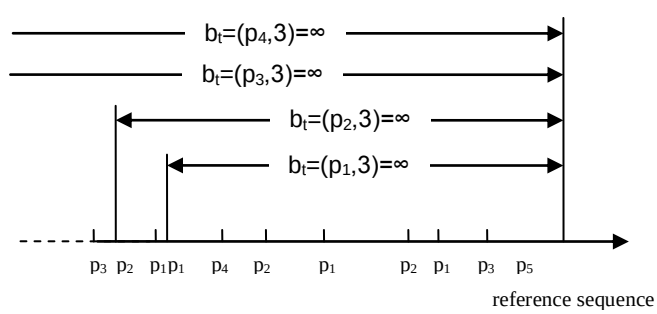
For classical LRU pages with the best estimate for inter-arrival time i.e. with the shortest such intervals are the ones kept in the buffer. In case of LRU-K tracking is done of the times of the last K references to popular database pages, and by usage of this information the inter-arrival time of such references on a page-by-page basis can be estimated.

A set of disk pages has been given, denoted by the set of positive integers N = {1, 2, …, n} and that a series of references to the pages specified by the reference string: r_1, r_2,…, r_i,… of the database system under study, where $r_t = p(p \in N)$ means that r_t is a reference to disk page p. Clearly, each disk page p has an expected reference inter-arrival time, which is the time between successive occurrences of p in the reference string. The system then tries keeping in memory buffers the pages with the shortest access inter-arrival times, or equivalently the greatest probability of reference. The classical LRU algorithm takes an arithmetical approach, of keeping in memory the pages that seem to have the shortest inter-arrival time.

In case of LRU-K Algorithm a page replacement policy works, when a buffer slot is needed for a new page from disk, it says the page p to be dropped is the one whose Backward K-distance, b_t (p, K), is the maximum of all pages in buffer. Given a reference string known up to time t, r_1, r_2,…, r_t _ , the Backward K-distance b_t (p, K) is defined as the distance backward to the Kth most recent reference to the page p :

$$b_t(p, K) = \begin{cases} x, \text{ if } r_{t-x} \text{ has the value } p \text{ and there have been exactly } K-1 \text{ other values } i \text{ with } t-x < i \le t, \text{ where } r_i = p. \\ \infty, \text{ if } p \text{ doesn't appear at least } K \text{ times in } r_1, r_2, .., r_t \end{cases}$$

Ambiguity occurs when more than one page has b_t (p, K) = ∞. Then, a supplementary policy, like classic LRU, helps in replacement of victim among the pages with infinite Backward K-distance selection.



Above figure depicts a simplified example of LRU-3 for a sequence of accesses to pages p_1, p_2,…, p_n. In case of an incoming request for a page p_5 that is absent and at the same time buffer is full, from the point of the new access based on the backward K-distance a victim is chosen. In above example, both p_3 and p_4 have the backward K-distance of infinity, so a secondary policy is required for tie breaking.

- *Mr. Shivakash Sahu is currently pursuing masters degree program in engineering in Shri Ram Institute of Technology, Jabalpur, India.*
- *Mrs Roshni Dubey is currently working in Shri Ram Institute of Technology, Jabalpur, India.*

LRU-K accesses the history of each page to better distinguish pages that have to be kept in the cache. This is based on the hypothesis that the reference string is a sequence of random accesses with special distribution, and each disk page p has a well defined chance, βp, should be the next referenced page by the system. If access patterns are changed it may also change the page reference probabilities, but the probabilities βp, have comparatively long periods of stable values, and on top of that assume to be independent of t. From the above discussion of LRU-K, it's conjecturable that for K > 2, the LRU-K algorithm shall somewhat provide an improved performance over Classical LRU for stable patterns of access, however is less receptive to changes in access patterns.

Especially in cases where K ≥ 2, careful consideration is required for ensuring proper caching behavior. The first one, Early Page Replacement, arose in conditions where a page that has been recently read into memory buffer and is not preserved in the buffer because of standard LRU-K criteria, for example, because the page has a b_t (p,K) value of infinity. We will surely drop this page from the cache relatively more quickly, to save memory resources for more worthy disk pages. Though, a page that is not normally popular shortly after being referenced for the first time, may still experience a burst of linked references. This issue is addressed with a Correlation Time-Out parameter in LRU-K Algorithm, so that a page is not dropped immediately after its first reference, and is kept around for a Correlated Reference Period for elimination of the likelihood of a dependent follow-up reference.

The second feature is to preserve history information of references for objects that are not currently present in the cache. This is referred to as the Page Reference Retained Information Problem. The LRU-K Algorithm addresses this problem with a Retained Information Period parameter so that after most recent access the system maintains history information about any object for that period. The possibility of repeatedly referencing a page is removed as soon as it is evicted and no record of prior references are kept and is dropped because each time the backward K-distance is estimated as infinity.

2 Challenging the Least Recently Used Algorithm

Time taken to access a data set that present on disk, when compared to the time user takes to actually use the dataset is negligible. But, in case of absence from the disk, it can take up to multiple hours for reading the data from tape.

Hit rate can be used to measure the performance of a layered storage environment. It is calculated by dividing the number of accesses of files when they are present on primary storage (disk) by the total amount of accesses. Presently used replacement policy, the Least Recently Used (LRU) algorithm, keeps most recently used files on disk and while others on tape.

The system is different from active storage environments. Besides, a user actively puts his data in passive working storage (archive) or copies data from the passive to his active storage environment. The difference between these storages can be observed from what a user can do with the data: our processing is executed from active storage. This situation is known as 'active archiving' situation.

In 'active archiving', presumption is that just archived files will not be used for a long period. As well, just de-archived files are moved immediately to a user's active storage environment for working. Therefore, the same files will perhaps not be accessed sooner. LRU algorithm being chosen as replacement policy for deciding upon the content of the disk does not align with the situation of 'active archiving'.

Also, the implemented algorithm must not constrain the system heavily. That is, although an algorithm may theoretically be far more optimal in terms of hit rate, it may increase system overhead such that the overall performance is decreased. Therefore, performance should be calculated in algorithm overhead.

3 Literature on Replacement Algorithms for Layered Storage

The challenge to LRU algorithm as a replacement algorithm has already been done before. The standard LRU algorithm is static and the field of research is layered storage. The same amount of files is considered by standard LRU for migration. Though, this acts as a constraint in the system, as the replacement algorithm takes a standard overhead cost always. With the increase in traffic on the system the response time gets constrained. That is why when more traffic is there, the replacement algorithm must be lowered in activity. The number of file replacements is considered as a variable and are dependent on the traffic. The overhead of the system decreases, due to this adaptation. A numerical, theoretical simulation can be used to show the overhead of the dynamic LRU can become overhead of the static LRU. But it lacks that it does not test the assumptions in a real situation. Moreover the sizes of the different storage layers are not taken into account. Consequently the total system will use more storage than the maximum capacity, if traffic is heavy remains same for a long period. As well, only access history can be used to predict file demand, but other factors may be likely to predict file demand as well.

The LRU policy is also adapted in the field of Web Cache Management, which specifies the web pages to be stored in different layer of web-servers. All parts must pass the primary storage to reach a user. This condition is, comparable to a layered digital archive. The last access of a file is under consideration in standard LRU and adds components for former accesses. The strength of this approach is that files are all considered as unique with unique access statistics. On the other hand, this research still lacks to incorporate other factors than access requests.

Prefetching is a pro-active replacement policy. Prefetching means that the algorithm can determine to data to be

migrated from secondary to primary storage. Our focus is on hierarchical storage models. The magnetism of files can be determined by means of a genetic algorithm and establish a fitness function with a regular update. 42% hit rate performance improvement is achieved. Prefetching algorithm can also be extended. Through mining web access logs, a decision tree can be constructed and frequently observed patterns of file accesses can be found. In comparison to the LRU algorithm both the approaches enhance the performance in terms of hit rate. Herewith, it can be concluded that the 'knowledge' extracted from historical access data changes over time. Certain web-pages become unpopular, either for all or for one person.

It has already been proposed to use data mining for a prefetching algorithm. Sequence miner for 2-sequences from web log data is used. The result of the sequence miner is knowledge in the form of rules of the format $A \rightarrow B$. The probabilities of files to be accessed are calculated from these rules. The files that are most likely to be accessed soon are 'buffered', which is similar to prefetching. The size of the buffer can be adapted in case of heavy traffic, hence makes the algorithm dynamic for traffic changes. However, traffic can increase heavily due to a prefetching algorithm. Therefore, argument remains for a trade-off between prefetching and computing overhead.

Some researches successfully challenge the LRU algorithm. Both general and access likelihoods due to relations between files are considered. Yet lack of generalized incorporation of this 'knowledge' by means of data taxonomy is there.

4 Contribution of my Work

The previous researches show the probable implementation of a prefetching algorithm based on data mining results. But, as they focus on web-page Cache Management, data taxonomy is not considered. Hence, generalized data mining knowledge is not extracted.

Our area of focus uses the concept of pro-actively prefetching data for digital archiving. Since layered storage is used in these fields, comparison can be done in the replacement algorithms. At the same time, digital archiving can get benefit from research in Web Cache Management. Furthermore LRU is not the most suitable replacement algorithm. The relations should be extracted in a generalized format since only a small percentage of an archive is de-archived. That is why prefetching based concept on data mining results is extended by generalization of extracted knowledge. Generalization of data mining results concept has not yet been implemented in a prefetching policy. Particularly, generalization on spatial coordinates shall be tested.

5 Some Research Question and Answers

Presented here are some research questions derived from the former and current situation of research, which show the probability of extracting knowledge from historical access data through data mining. Data mining is powerful in case of large databases. In addition, the knowledge is useful in successfully pro-actively prefetching data to higher storage layers, which can help in improving the hit rate of layered storage systems.

Question 1. What are the appropriate data mining techniques for extracting useful knowledge from historical access data?

Data mining shows to be very useful in extracting knowledge from very large databases as is in our case. There are many areas of data mining for which definitions and application exist. This goes in alignment with the number of available techniques. Previous researches show different techniques of their application, which indicates about choice to be made.

Question 2. What knowledge can be extracted to be used in a prefetching algorithm?

This question is required to answer the first research question as input. To turn the process of data mining, the experts from the data mining industry have developed a common standard. Since the framework is based on results of data mining experts so it will be followed closely. The framework explains separate stages of a data mining process: business understanding, data understanding, data preparation, modeling, evaluation and deployment.

Question 3. Based on the results of the knowledge extraction process, what does a prefetching algorithm look like?

Input for the prefetching algorithm is generated from the results of the data mining process. The LRU algorithm is modeled, based on the results of the data mining process. Reason to use same simulation tool for both algorithms is to make comparison easy.

Question 4. What is the performance of the prefetching algorithm compared to the currently used LRU algorithm?

Derived from the literature the performance comparison between the two models is based on two elements:

1. the hit rate – is a predictor of the rate of the system that constraints the user to wait for accessed files, which is useful in improving the ratio.

2. algorithm overhead - Although the hit rate is of use, but in case of too large algorithm it fails because the system may be constrained in its primary activities.

References

[1] J. Zhou, P. Martin, and H. Hassanein. QoS Differentiation in Switching-based Web Caching. In Proc. of the 23rd IEEE International Performance, Computing, and Communications Conference (IPCCC '04), 453-460, Phoenix, AZ. Apr 2004.

[2] Q. Zou, P. Martin and H. S. Hassanein. Transparent Distributed Web Caching with Minimum Expected Response Time. In Proc. of the IEEE

International Performance, Computing, and Communications Conference (IPCCC '03), Phoenix, AZ. Apr 2003.

[3] W. Chen, P. Martin and H. S. Hassanein. Differentiated Caching of Dynamic Content using Effective Page Classification. In Proc. of the 23rd IEEE International Performance, Computing, and Communications Conference (IPCCC '04), 293-298, Phoenix, AZ. Apr 2004.

[4] Huang, Y.-F., Hsu, J.-M. Mining web-logs to improve hit ratios of prefetching and caching. Knoledge-Based Systems, 2007

[5] Balamash, A., Krunz, M., Nain, P.,Performance analysis of a client side caching/prefetching system for Web traffic. Computer Networks, 2007.