February 2022

Cache Management Using Machine Learning

Deniz Altınbüken

Kevin Chen

Ramki Gummadi

Xiao Ju

Nikhil Sarda

See next page for additional authors

Inventor(s)
Deniz Altınbüken, Kevin Chen, Ramki Gummadi, Xiao Ju, Nikhil Sarda, and Zhenyu Song

Cache Management Using Machine Learning

ABSTRACT

Various types of cache eviction policies can be utilized to identify cache objects for eviction from a cache. Effective cache eviction policies that improve cache hit rates can improve performance. This disclosure describes the use of machine learning for cache management. A base policy, e.g., least recently used (LRU) or other policy, is used to organize cache objects by eviction order. A machine learning based ranking model re-ranks the eviction order such that a certain cache object is the first object scheduled to be evicted. Upon eviction, future data requests are observed and serve as feedback to improve the ranking model. The ranking model takes access patterns of cache objects to determine eviction rank.

KEYWORDS

- Memory cache
- Random access memory (RAM)
- Cache eviction
- Cache object
- Cache management
- Least recently used (LRU)
- Machine learning
- Cache eviction rank
- Cache insertion

BACKGROUND

In modern computers, random access memory (RAM) can serve as a cache for a solid-state storage device (SSD) or hard disk, such that frequently accessed data is stored in RAM which is faster to access than the SSD or hard disk. Cache capacity is limited and is managed using various caching strategies. For example, a cache object currently in cache that is less frequently accessed is evicted to make room for another that sees higher level of accesses. Some examples of common cache-eviction policies include least recently used (LRU), weighted LRU, least frequently used (LFU), first in first out (FIFO), greedy dual size frequency (GDSF), etc. It is valuable to have a cache eviction policy to optimize the cache hit rate - the proportion of data requests that are fulfilled by the cache.

DESCRIPTION

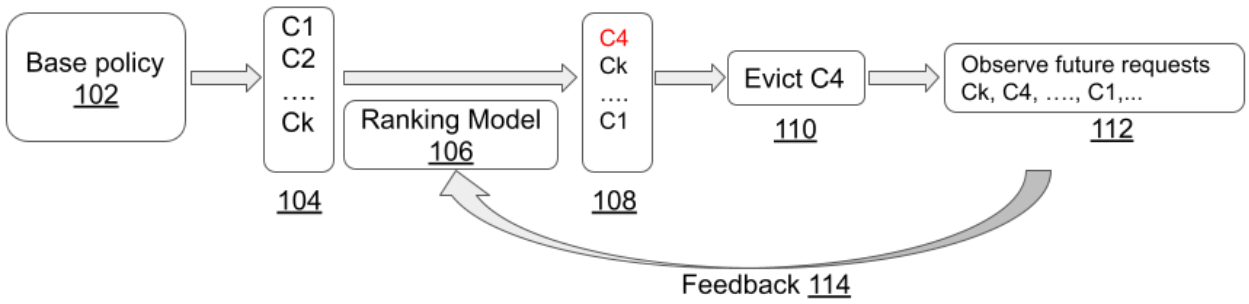

Fig. 1: Machine learning based memory cache management

This disclosure describes techniques of cache management based on machine learning. As illustrated in Fig. 1, a base policy (102), e.g., LRU, LFU, etc., is used to organize cache objects (104) C1, C2, …, Ck by eviction order. A cache object is also referred to as a block of data. A ranking model (106) that is implemented using machine learning (such as a neural network) re-ranks the eviction order such that (in this example) cache object C4 is the first object scheduled to be evicted (108). Upon eviction (110) of C4, future data requests are observed (112)

and serve as feedback (114) to improve the machine learning ranking model. The base policy, e.g., LRU, LFU, etc., is applied to each object in the cache prior to the use of the ranking model to prune the list of eviction candidates. The base policy also provides safety in worst-case scenarios.

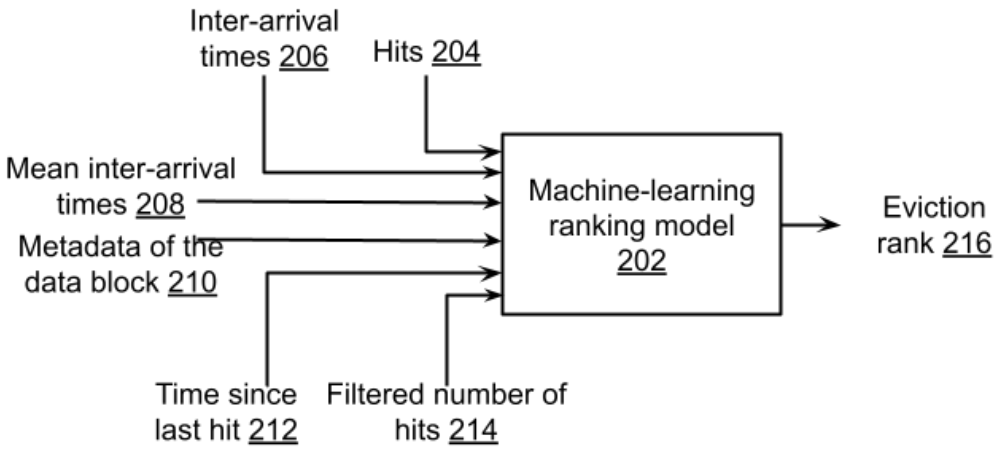

Fig. 2: Machine-learning to determine eviction rank

Fig. 2 illustrates an example of a machine learning ranking model (202) to determine the eviction rank (216) of a given cache object. Some example features (inputs) used by the machine learning model to determine the eviction rank of a given cache object include: the number of hits (204) for that cache object; a certain number (e.g., between 10 and 1000) of inter-arrival times between those hits (206); a statistic, e.g., mean, variance, etc., of the inter-arrival times (208); metadata (size, type, dates of creation, etc.) of the cache object (210); the time since the last hit (212); a filtered version of the number of hits (214), filtered using, e.g., exponential decay, which gives greater weightage to recent hits; other access patterns of cache objects; etc.

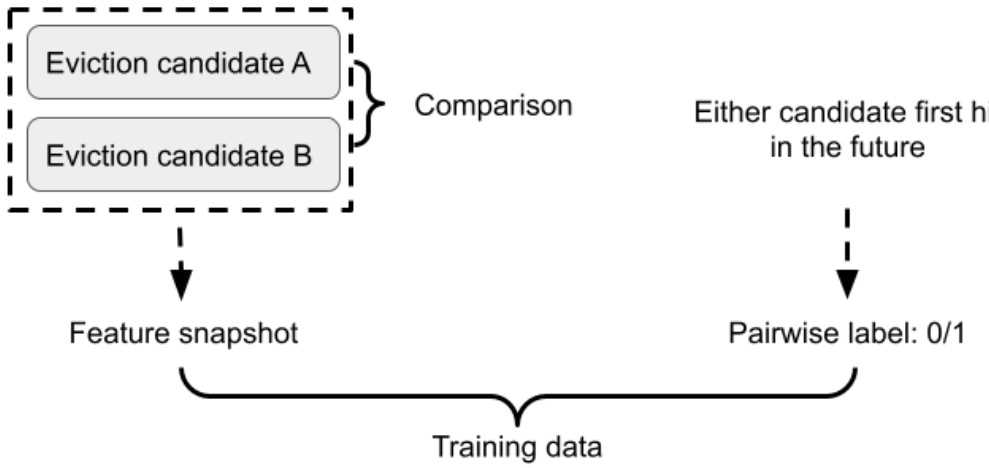

Fig. 3: Training the machine learning model

Fig. 3 illustrates training the machine learning model. Two cache objects, A and B, each with their features, are candidates for eviction. These can be previously compared pairs of candidates. One of the candidates, e.g., candidate B, is observed to be the first to receive a hit. Candidate B is labeled as stay-in-cache (binary label '1'), while candidate A is labeled as eviction candidate (binary label '0'). In this manner, the respective features of the cache objects A and B, labeled with stay-in-cache or eviction labels, form the ground-truth training data. Training can also occur during operation, in the form of feedback, as explained earlier with reference to Fig. 1.

Features associated with each candidate relate to the history of its prior accesses. Therefore, for every access to the cache, the features of a candidate are updated in parallel with cache operations such as evictions, lookups, inserts, etc. Since feature metadata is lightweight in comparison to the actual size of a cache object, features of a cache object can continue to be stored even after an object is evicted from the cache (up to a limit, e.g., ten times the number of objects in the cache, sufficient to track history). Evictions of feature metadata can be performed using LRU or another eviction procedure.

The training data generated from the above procedure is stored in an in-memory replay buffer. A separate thread periodically wakes up and uses the training data in the replay buffer to create mini-batches, which are used to update the ML model that powers the candidate ranking.

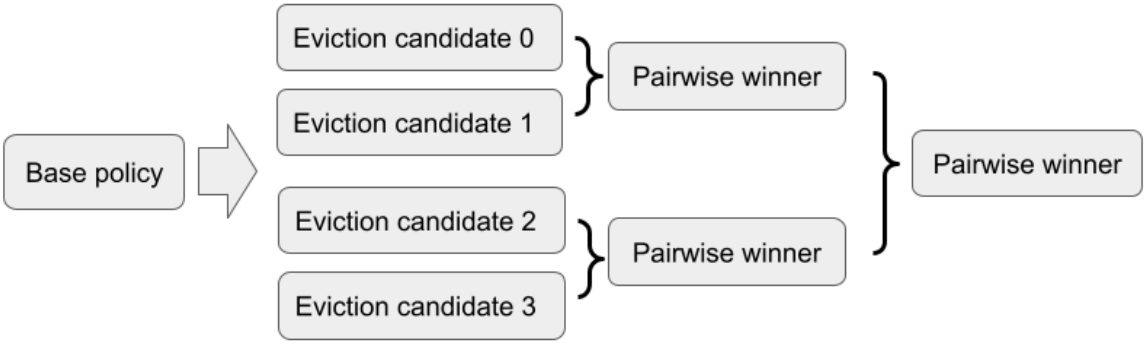

Fig. 4: Pairwise eviction procedure can order a larger list of cache objects by eviction rank

Fig. 4 illustrates that the pairwise eviction ranking procedure described earlier can be used to order a larger list of cache objects. For each eviction, the base policy is first used to determine a small set of eviction candidates. The machine learning model performs pairwise ranking to select the final eviction candidate. The final candidate is evicted, and the remaining candidates are re-inserted into the base policy. For example, upon every eviction, four eviction candidates can be sampled from the last twenty objects in the WLRU tail. After three pairwise comparisons, the final winner is selected. Combining pairwise ranking together with candidate selection results in strictly improved eviction decisions, e.g., eviction decisions that result in higher hit rates and lower miss rates.

In this manner, machine learning models can be trained and applied to pick candidates from the bottom of an LRU list (items that have lower value) to determine a candidate with the lowest score. The lowest scoring candidate corresponds to a lower likelihood of being accessed sooner than other candidates and is a candidate for eviction. The machine learning model takes access patterns and other features into account.

CONCLUSION

This disclosure describes the use of machine learning for cache management. A base policy, e.g., least recently used (LRU) or other policy, is used to organize cache objects by eviction order. A machine learning based ranking model re-ranks the eviction order such that a certain cache object is the first object scheduled to be evicted. Upon eviction, future data requests are observed and serve as feedback to improve the ranking model. The ranking model takes access patterns of cache objects to determine eviction rank.

REFERENCES

[1] Zhenyu Song et al., "Learning Relaxed Belady for Content Distribution Network Caching," https://www.usenix.org/system/files/nsdi20-paper-song.pdf accessed Jan. 15, 2022.