

Progressive Review Counters for proficient Cooperative Caching in Web Search Engines

S. Ramaiah

M. Tech (CSE),
Audisankara Institute of Technology,
Gudur, Andhra Pradesh, India.

P.Venkateswara Rao

Professor & Head, Department of CSE,
Audisankara Institute of Technology,
Gudur, Andhra Pradesh, India.

ABSTRACT

Search engines are the only prevalent resource of web. When people are looking for something online they go to a search engine first. If not they know the site they want to go to and navigate to it by typing their domain into the browser. Up-and-coming search engines are moving several steps further than the naive sack of words rough calculation prime in today's information retrieval models. Appreciably more computationally luxurious than the existing search engines, because those search engines are fuse the collection of credentials to a set of data objects that are interpretable by the system. The production of these data structures is expensive.

As a Solution we suggest and evaluate a disseminated cooperative caching approach based on the Progressive Review Counters (PRC). PRC is a new data structure that stores an approximated documentation of the data accesses in every computing node of a search engine. The PRC detain the regularity of accesses to the essentials of a data collection, and the progression of the access patterns for every node in a network. The PRC can be competently reviewed into what we call PRC-reviews to obtain fairly accurate information of the document entries accessed by each computing node. We use the PRC-reviews to initiate two algorithms that supervise our disseminated caching strategy, one for the sharing of the cache contents, PRC-placement, and another one for explore of documents in the disseminated cache, PRC-search. The Result analysis of PRC shows that cooperative caching approach outperforms status of the art models in hit rate, throughput, and location evoke for numerous scenarios.

Keywords

SOA; Progressive Review Counters; cooperative Communications; data cahing; search engines.

I INTRODUCTION

Up-and-coming search engines are pitiful several steps ahead of the naive words for estimate main in today's

information repossession models. For example, multimedia search requires a unfathomable scrutiny of the multimedia substance (e.g. music, images, movies), instead of just toning query keywords to the subtitle of the multimedia object. Or, question answering systems perform profound syntactic and semantic study of the document texts in order to provide short, exact answers to normal language questions.

This paper concentrated on up-and-coming generation of search engines, which include numerous possessions in frequent: they are all drastically more computationally exclusive than present search engines, and they all produce the set of documents to a set of data objects that are interpretable by the engine, e.g., textual passages with profound normal language analysis [1] or multimedia objects with visual/audio features [2]. Since the creation of these data structures is pricey, they are high-quality candidates for in-memory caching when their exploit is recurrent. In addition, changes to the data object sets used by these search engines are usually inadequate. They classically consist of new document add-ons, for example, when fresh entries of a blog are published, but these changes do not adapt earlier versions of a document. In further words, once a document is added to the collection, it can be measured read only.

As a solution to above this paper dPRCribes a novel cooperative caching strategy for such disseminated search engines that is entirely implemented on service hardware. Our approach manages cache contents according to current practice information. The base of our construction is a collection of local caches (one per system node) that are associated mutually by a cooperative protocol that provides system wide lucidity. Our cooperative caching approach proposed a innovative data structure, which we call *Progressive Review Counters* (PRC). The PRC keep a trace of the recent data accesses of a node. This information is stored in count up bud filters alike to other proposals such as [3],[4], nevertheless, and as we feature additional in this document, our suggestion applies these compacted data structures to record the regularity of access for a specified time window, and at the same time preserve its current history. This information turns to be incredibly important in order to get better the position and locality of data in a cooperative cache, because it collaborates recency and frequency of historical data accesses.

The PRC information uses to covenant with two imperative issues in the cooperative cache administration: the *placement* and the *search* of the cached information. The placement algorithm reins wherever information is cached in the network and is driven by two principles: (a) information must be cached in the nodes where it is most often accessed and (b) information normally accessed should be cached at slightest in one node. We suggest PRC-*placement*, which is an algorithm that achieves good cache area by sending documents to nodes that accessed them normally in a current time casing, and by avoiding the duplication of documents occasionally accessed. As a second concern for the cooperative cache administration, we revise data search, i.e., how to place the node that is at present caching a convinced data unit. We intend PRC-*search*, which is a search algorithm that estimates the prospect of verdict a document in a node vigorously, dropping the number of nodes queried. PRC provides good performance for both the placement and search of documents in a disseminated search engine system.

We recapitulate the assistance of this paper as: (i) we propose the PRC as a compacted data structure to record the current account of data accesses, (ii) we intend PRC-*placement* that is an algorithm to distribute the cache contents in a bunch of computers competently, (iii) we advise PRC-*search*, which is a locality process to know with high prospect if a document is accessible and where in the cooperative cache; and (iv) we evaluate our approaches to the most recent and significant approaches using a fully-fledged semantic search engine based on question answering technology.

The reminder of the paper is structured as. In Section II, we review the related work. Then, Section III dPRCribes the basic construction that we target with our cooperative cache proposals. In Section IV, we dPRCcribe the PRC data structure and the PRC-*placement* and PRC-*search* algorithms. Finally, we conclude in Section V.

II RELATED WORK

The sufficiency and the cooperative caching considerations differ in view of the function area. For example, Wolman et al. converse systematically the application of cooperative caching for web data in WANs [5]. One of their conclusions is that any easy cooperative caching algorithm is closing adequate to an oracle cache policy for caching web data. But, this setup is not normally suitable for all the applications, and cooperative caching has been useful in multiple areas successfully [6]. In this work, we spotlight on the placement and the search of the partial computations of a complex search engine, stored in the main memory of the dissimilar nodes that are consistent by a LAN. This circumstance too differs from the assumptions by Wolman since the charge to process the documents to solve a multipart query is larger than the time to read an HTML document and the communication expenses are lower in a LAN than in a WAN.

In single of the primary studies for data placement in cooperative caches, Dahlin et al. proposed n-chance forwarding, which is a approach that selects the goal node for forwarding at

random, so no information from other nodes are used. Afterward approaches focused on how to improve this algorithm, taking advantage of statistics of the cache. Feeley et al. proposed GMS, which implements a centralized process where all the nodes send statistics about the age of their cache contents to a coordinating node [8]. Ramaswamy et al. projected Expiration Age in [10], where nodes switch over its current cache expulsion rate and decide to duplicate the data on nodes with petite controversy. Dominguez-Sal et al. projected Broadcast Petition Recently, which uses a time changeable to limit duplicates in the network but does not use global system information such as the PRC. It is also known which the best placement is if all the data access statistics are available, as shown by Koropolu. This algorithm, though, is static and does not get a feel for if the distribution changes. Finally, some placement solutions are based on hash functions, but the major problem of these approaches is that the application of hashes does not encourage the locality to access the data. There has been also imperative endeavor on the locality of documents in a disseminated environment. Sarkar and Hartman execute placement and search in a distributed cache using imprecise information, or hints, about the state of a client-server system. The hints are disseminated among the clients, but, unlike our search mechanism, it relies finally on the information in the main central server to locate the data. Rhea and Kubiatowicz proposed attenuated bloom filters that summarize the information of distant nodes for routing data [4]. However, they do not count frequencies and attenuated bloom filters mix the data location of several nodes in a bloom filter, which reduces its precision.

Fan et al. use a bloom filter-based data structure to carry out the search [3]. though, they do not provide a placement method, nor a history of content accesses, where as our PRC provides this with a single data structure. In our investigational section, we confirm that the combination of occurrence and recency improve the locality recall over approaches that only store a bit indicating if the cached entry was available at a certain moment of time, like summary caches.

In addition, PRC are a general data structure that may be used in diverse contexts. For example, Dominguez-Sal et al. argue load balancing algorithms for disseminated systems that are attentive of the cache filling in order to get better the throughput and the data locality. Nonetheless, it is not believe the placement and search problems, which we do in the current paper. Besides, many algorithms have appeared from the peer to peer neighborhood to find information in large networks using distributed hash tables (DHT): Chord, Pastry etc.

Even though DHTs were intended to be used in WANs, some prototypes realize them in networks with small latencies. For example, Shark proposed a cooperative cache for disseminated file systems, and Squirrel is a web server cache based on cooperation. But, one limitation of DHTs is that it needs to contact several nodes in succession during the locality procedure, which introduces latency in the search. Our proposal overcomes this problem because we are able to contact a small subset of nodes in parallel. Finally, the aspects related to the

organization of the limited memory pool dedicated to caching are orthogonal to the cooperative caching strategies proposed in this paper. For instance, if the document size exhibits a large variability, the largest documents may be divided into sections, paragraphs, or pages with a fixed number of characters. Also, we used an LRU strategy in our system as the local policy, but our disseminated approaches can be abstracted away from the local policy. Thus, we believe more advanced local cache policies such as the proposed by Jiang et al. [9], can also benefit from our placement and search algorithms.

III SYSTEM ARCHITECTURE

This paper focus on search engines that are serene of a collection of computing blocks, which we treat as black boxes, and the output of the system is obtain as a succession of computations of these blocks. A few of the computing blocks may be computationally expensive, and thus, their outputs are ideal candidates for caching. Every computationally rigorous block has a pond of memory to store up the data related to a document following a local cache strategy. We select LRU as the local cache strategy because its wide usage in many applications and its proven adequacy to the workload of search engines.

To build the disseminated system, we consider that all nodes have an occurrence of the search engine, and can access the document set and its connected indexes. This approach follows preceding approaches for complex search engines, where each node is able to compute queries separately. We illustrate an revision of the formerly dPRCribed design to a question answering (QA) system with three computing blocks (question processing, passage retrieval, and answer extraction) and its corresponding local caching pools in Figure 1.

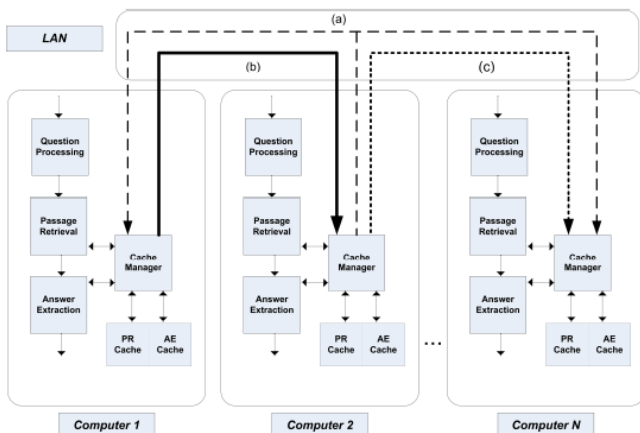


Figure 1. QA system composed of three sequential blocks QP, PR and AE. PR and AE can request and store data into the cache using the cache manager.

We employ a distributed cooperative cache with no federal processes, which instinctively works like a p2p network

without central node, where all nodes can directly get in touch with the rest of nodes of the network. The communication between nodes is facilitated by the following operations:

- *Request/Response*: these operations attain a cached entry from a distant node. Once a node has a local miss, it desires the document through a multicast operation (operation (a) in Figure 1). The request includes the document recognizer and a parameter to recognize the computing block that requests the data. The receivers of the request respond with the data if the entry is obtainable in their caches (operation (b)).
- *Forward*: this operation transfers the least recently used cache entry from a layer to the same layer of another node in the network (operation (c)).

The access to the group content can be optimized by using communal disks, a disseminated file system optimized for read operations or simply duplicating the set if the dataset fits in a computer. This architecture can be easily extended to very-large-scale settings. For example, one can envision a setup where a very large collection is partitioned and each different partition is replicated among a group of nodes, where each group implements our architecture.

IV COOPERATIVE CACHING ALGORITHMS

This section presents the new distributed cooperative caching algorithms proposed in this paper. We first present the data structure that we recommend in this paper and is shared by all our cooperative caching algorithms, the Progressive Review Counters (PRC). Then, we explain how to apply the PRC summaries to the placement (PRC-placement) and the location of data (PRC-search).

A. Progressive Review Counters

The Progressive Review Counters (PRC) is a data structure deployed in each computing node, which records a window of the recent history of the local accesses. The PRC is collected by a linked list of k Count Bloom Filters (CBF). Through a convinced period of time, T , the CBF at the head of the list is active, i.e., it counts the occurrence of the elements in a streamed data set. After T time units, a sliding operation is applied, so that a new CBF is used during the next T time units. On reuse, the CBF at the back of the list is reset (all counters are set to 0) and it becomes the head of the list, that is, the active CBF.

Each access to a document is recorded into the active CBF locally. Thus, each computing node keeps an PRC with k CBF, which monitor the last $T \cdot k$ time units. After T time units, when a sliding operation is triggered, each node computes a summary of the PRC that is broadcasted to the nodes in the network, the PRC-summary. Therefore, each node receives the PRC summaries from the rest of nodes, and the distributed algorithms check the summaries to obtain a dPRCRIPTION of the system state in order to dynamically update its decisions. The PRC-summary is computed as the aggregation of the k count filters of the PRC. We evaluate two possible summary implementations:

- *Plain summary*: A simple addition of all the CBFs.

- **Linear summary:** A weighted addition where the most recent CBF is multiplied by k , the second most recent CBF is multiplied by $k - 1$, and so on, up to the k -th CBF, which is multiplied by 1.

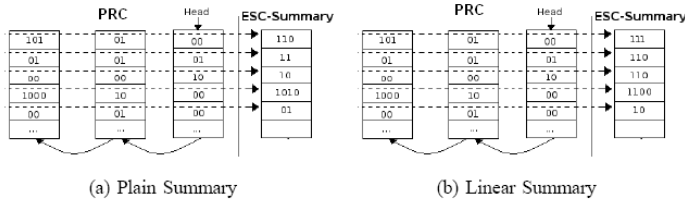


Figure 2. Summarization example of an PRC (counters are in binary).

We depict a diagram of an PRC in Figure 2 with $k=3$. The figure compares how we combine the three CBFs of the PRC in a plain and a linear summary. The counters in the PRC-summary take the weighted sum, by rows, of the CBFs.

The PRC-summary generated is also a CBF. The operations related to the PRC have low complexity. The recording of an access to a document in the PRC has constant complexity. The summary and slide process take linear time with respect to the CBF size, and the look-up of the frequency in a PRC-summary has also constant complexity. In the supplementary material, we show that the network overhead introduced by the PRC is small enough to permit systems with throughput of thousands of queries per second.

B. PRC-Placement

Our proposal performs the distributed placement when a local cache is full and a document is evicted from memory: the document is forwarded to another node if the algorithm decides that this document is valuable enough to be kept in some node of the network, otherwise it is simply discarded. Our objective is to keep the documents in the node where they are accessed, and to encourage the availability of frequent documents in some node of the network.

The target node for forwarding the evicted entry is decided according to the PRC-summaries that have been sent by the remote nodes. The algorithm selects the node whose PRC-summary contains the highest value for the entry being processed and transfers the entry to that node. If the entry is already present in the receiving node, our algorithm marks the entry as the most recently used entry for the local LRU policy. Also, if more than one node has the same largest value, a random selection among these nodes is performed. The pseudocode of PRC-placement is summarized in the supplementary material.

We have an additional counter for each entry in the cache, which accounts for the number of forwarding actions since the last access for each document. Each time a document is forwarded, its respective counter is incremented. If a document is accessed while staying in a cache, its counter is reset. The objective of this counter is to limit the number of forwarding

actions to avoid excessive network traffic. We limit this number to 2 in accordance to the results obtained.

PRC-placement automatically evicts from the cache entries scarcely accessed, i.e., documents not read after several forwarding operations. If an entry is evicted from the local cache we are not deleting it but trying to reduce the number of copies: the destination node is the one with the highest probability of holding a copy. If the destination node holds a copy of the entry it is not necessary to remove any other entry, and hence the forwarding procedure is finished. If the destination node does not hold a copy, PRC-placement repeats the forwarding procedure until (a) it finds a document that can be discarded because it is not frequently accessed (with a forwarding counter exceeding the threshold), or (b) a document with multiple copies in the network.

C. PRC-Search

The search algorithm is in charge of locating the node where a document is stored. The objective is to reduce the number of *avoidable misses* in the system: an avoidable miss is a remote cache miss for a document that is cached somewhere in the network, but it is not found because the system did not query the proper nodes. Although the broadcast of the requests reduces the number of avoidable misses to zero, the number of messages may become a bottleneck. The PRC-search procedure is called whenever an entry d is not found in the local cache. First, PRC-search builds a list ($L=n1, \dots, nN$) with all the nodes in the system. The nodes in L are sorted in decreasing order of access frequency to d , according to the PRC-summaries locally available. Then, PRCsearch selects the first h nodes from L , and requests the entry from all of them in parallel (note that the selection process is computed without network communication). We compare two methods to select h :

Static: h is set to a constant value which is decided at the initialization of the system.

Dynamic: h is calculated every time a document is searched in order to keep the probability of an avoidable miss below a defined threshold. The dynamic PRC-search policy divides the nodes into two lists: the candidate nodes to be queried and the rest. Initially, the candidate node list is empty, and PRC-search estimates the probability of an avoidable miss if no node is queried ($PAvMiss(0, d)$). Then, following a greedy procedure,

PRC-search includes the node which is most likely to contain d in the candidate node list until the probability of an avoidable miss is below a threshold $_$. PRC-search estimates the probability of an avoidable miss with the following formula, which assumes that the probability of finding a document in one node is independent among nodes:

$$\begin{aligned}
 P_{AvMiss}(s, d) &= \text{prob}(d \text{ is not in nodes } (n_1..n_s)) \cdot \\
 &\quad \text{prob}(d \text{ is in any of the nodes } (n_s..n_N)) \\
 &= \left[\prod_{i=1}^{i=s} (1 - P_{freq}(ESCS(i, d))) \right] \cdot \\
 &\quad \left[1 - \prod_{i=s+1}^{i=n} (1 - P_{freq}(ESCS(i, d))) \right],
 \end{aligned}$$

where $PRCS(i, d)$, is the value in the PRC-summary received from the i th node for document d ; s is the number of nodes that are queried to locate document d ; and $P_{freq}(x)$ is the probability that a document with frequency x is available in the memory of the remote node. h is selected as the smallest number of nodes such that $P_{AvMiss}(s, d) < \epsilon$. In the worst case (which is unlikely), the algorithm will query all nodes, but will query one or even none if the document is very unlikely to be in cache. We show the pseudocode for this search procedure with an example in the supplementary material.

In order to evaluate $P_{AvMiss}(s, d)$, each node keeps an array with a local estimation of $P_{freq}(x)$, for each possible value of the counters in the PRC-summaries. This probability is updated dynamically according to the results of the search history. For example, in a system that had requested 5 times documents with frequency 2 and they were only found once, then the current value of $P_{freq}(2)$ would be $1/5$. After each search, this value is updated. For instance, if the document was found in the node whose PRC-summary indicated a frequency of two then $P_{freq}(2)$ becomes $2/6$.

The algorithm is able to adapt to a query distribution because $P_{freq}(x)$ is computed on the fly. The computational cost of this procedure is $O(N)$ in the worst case because the algorithm checks the summaries from all nodes for a given document. Nevertheless, $P_{AvMiss}(s, d)$ is a monotonic decreasing function, so as soon as it reaches a value below ϵ , this becomes the final result, and there is no need to further compute it for a larger set of nodes.

In the supplementary material, we model the overhead of the communication generated by the search algorithm. This analysis shows that the additional cost to transmit the PRCsummaries yields a significant reduction of nodes queried and the overhead of the algorithm is thus significantly smaller than that of the broadcast protocol.

V CONCLUSION

Search engines are the only prevalent resource of web. When people are looking for something online they go to a search engine first. If not they know the site they want to go to and navigate to it by typing their domain into the browser. Up-and-coming search engines are moving several steps further than the naive sack of words rough calculation prime in today's information retrieval models. Appreciably more computationally luxurious than the existing search engines, because those search

engines are fuse the collection of credentials to a set of data objects that are interpretable by the system. The production of these data structures is expensive.

As a Solution we suggest and evaluate a disseminated cooperative caching approach based on the Progressive Review Counters (PRC). PRC is a new data structure that stores an approximated documentation of the data accesses in every computing node of a search engine. The PRC detain the regularity of accesses to the essentials of a data collection, and the progression of the access patterns for every node in a network. The PRC can be competently reviewed into what we call PRC-reviews to obtain fairly accurate information of the document entries accessed by each computing node. We use the PRC-reviews to initiate two algorithms that supervise our disseminated caching strategy, one for the sharing of the cache contents, PRC-placement, and another one for explore of documents in the disseminated cache, PRC-search. The Result analysis of PRC shows that cooperative caching approach outperforms status of the art models in hit rate, throughput, and location evoke for numerous scenarios.

REFERENCES

- [1] D. Roussinov, W. Fan, and J. Robles-Flores, "Beyond keywords: automated question answering on the web," *Commun. ACM*, vol. 51, no. 9, pp. 60–65, 2008.
- [2] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *ICCV*, 2003, pp. 1470–1477.
- [3] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.
- [4] S. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *INFOCOM*, 2002.
- [5] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the scale and performance of cooperative web proxy caching," in *SOSP*, 1999, pp. 16–31.
- [6] T. Anderson, D. Culler, and D. Patterson, "A case for now (networks of workstations)," *IEEE Micro*, vol. 15, no. 1, pp. 54–64, 1995.
- [7] M. Raunak, "A survey of cooperative caching," *Technical report*, 1999. [Online]. Available: <http://citeseer.ist.psu.edu/raunak99survey.html>
- [8] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, and C. Thekkath, "Implementing global memory management in a workstation cluster," in *SOSP*, 1995, pp. 201–212.
- [9] S. Jiang, F. Petrini, X. Ding, and X. Zhang, "A locality-aware cooperative cache management protocol to improve network file system performance," in *ICDCS*, 2006, p. 42.
- [10] L. Ramaswamy and L. Liu, "An expiration age-based document placement scheme for cooperative web caching," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 5, pp. 585–600, 2004.