A Performance Analysis of Searching Algorithm in Distributed System

S.Ramaiah
Asst.Professor,KMMITS
TIRUAPATI
ramaiahsali@gmail.com

P. Hemanthkumar
Asst.Professor,KMMITS
TIRUAPATI
hemanthmtechcse@gmail.com

N.Jaya krishna
Asst.Professor, SREC
TIRUAPATI
jaya1238@gmail.com

***Abstract* -The caching mechanism is used to develop the server load balancing retrieval models. We are focusing on the user who can search through keyword in search engine, and also the user how many times they accessed data will be stored in cached information. The search engine follows some common techniques, the collection of data to a set of data objects or data sets that are stored by the machine, but in this method there is no data retrieving easily. So we use the ESC information to deal with two important issues in cooperative cache management. The placement algorithm controls where the information is cached in the network clients and is driven by two principals. Information should be cached in the nodes where it is most often accessed and it distrusted into the several clients and information frequently accessed should be cached at least one node and also they will be having the log of the system. Through this the server can do server load balancing and accurate search concepts will be implemented.**

Keywords: Distributed System, Cooperative Caching, Count Boot Filter, Evolution recent Retrievals.

—————————— ◆ ——————————

I. INTRODUCTION

Distributed system is a collection of independent computers that appears to a single coherent system. Emerging search engines are moving several steps beyond the naïve bag-of-words approximation predominant in today's information retrieval models. For example, multimedia search requires a deep analysis of the multimedia content (e.g. music, images, and movies) instead of just matching query keywords to the caption of the multimedia object. Or, question answering (QA) systems perform deep syntactic and semantic analysis of the document texts in order to provide short, exact answers to natural language questions.

In this paper we mentioned the Evolutive Summary Counters for increase the file access speed based on that we will show how many times they access one file and user can search the client have that file is cache or not with the help of Cooperative cache algorithms.

A novel cooperative caching strategy for such distributed search engines that is fully implemented on commodity hardware. Our approach manages cache contents according to recent usage information. The foundation of our architecture is a set of local caches (one per system node) that are linked together by a cooperative protocol that provides system wide transparency. Our cooperative caching strategy relies on a new data structure, which we call Evolutive Summary Counters (ESC). The ESC keep a record of the recent data accesses of a node. This information is stored in count bloom filters (CBF) similarly to other proposals such as [2], however, and as we detail further in this document, our proposal applies these compact data structures to record the frequency of access for a given time window, and at the same time maintain its recent history. This information turns to be very valuable in order to improve the placement and location of data in a cooperative cache, because it combines decency and frequency of historical data access.

The ESC information to deal with two important issues in the cooperative cache management: the placement algorithm controls where information is cached in the network and is driven by two principals: 1) information should be cached in the nodes where it is most often accessed and 2) information frequently accessed should be cached at least in one node. The ESC placement algorithm achieves good cache locality by sending documents to nodes that accessed them frequently in a recent time frame, and by avoiding the replication of documents infrequently accessed. As a second issue for the cooperative cache management, we study data search, i.e., how to locate the node that is currently caching a certain data unit. ESC-Search algorithm that estimates the probability of finding a document in a node dynamically, reducing the number of nodes queried. A single data structure provides a good performance for both placement and search of documents in a distributed search engine system.

We summarize the contributions of this paper as:

1. The ESC as a compact data structure to record the recent history of data accesses,
2. The ESC- placement that is an algorithm to distribute the cache contents in a cluster of computers efficiently.
3. The ESC-Search, which is a location procedure to know with high probability if a document is available and where in the cooperative cache.
4. In this we focus on increasing file access speed in cooperative cache and how many time the user can access a single file and finally the file is cache memory or not will be shown with the help of

ESC-Summaries.

II. RELATED WORK

ESC are a general data structure that may be used in different contexts. for example, Dominguez-Sal et al. discuss load balancing algorithms for distributed systems that are aware of the cache contents in order to improve the throughput and the data locality [4]. Nevertheless, in [4], it is not consider the placement and search problems. Many Algorithms have appeared from the peer to peer community to find information in large networks using distributed hash tables (DHT): Chord[6], Pastry [7].etc. Although DHTs were designed to be used in WANs, some prototypes implement them in networks with small latencies. For example, shark implements a cooperative cache for distributed file systems [11], and Squirrel is a web server cache base on cooperation too [12]. But, one limitation of DHTs is that it needs to contact several nodes sequentially during the location procedure, which introduces latency in the search. Depending on the application area, the adequacy and the cooperative caching considerations differ. For example, Walman et al. discuss analytically the application of cooperative caching for web data in WANs [10].some placement solutions are based on hash functions[8]. But the main drawback of these approaches is that the application of hashes does not encourage thee locality to access the data. Sarkar and Hartmann implement placement and search in a distributed cache using inaccurate information, or hints, about the state of a client- server system [9]. The hints are distributed among the clients.

III. PROBLEM STATEMENT

A P2P network may consist of several peers. Each node has a relatively powerful data center, and the downloading peers need to access the data centers to get needed files. And suppose the source is very long distance from downloading peers means the download process taking much more time. The neighboring peers tend to have similar downloading process and thus share common interests. If one peer has accessed a data item form the data center, at that time another peer download the same file means it make delay.

We focus on search engines that are composed of a set of computing blocks, which we treat as black boxes, and the output of the system is obtained as a sequence of computations of the blocks. Some of the computing blocks might be computationally expensive, and thus, their outputs are ideal candidates for caching. Each computationally intensive block has a pool of memory to store the data related to a document following a local cache policy. We select LRU as the local cache policy because its wide usage in many applications and its proven adequacy to the workload of search engines.

The Question Answering system with three computing blocks (question processing, passage retrieval, and answer extraction) and its corresponding local caching that is shown fig. 1.

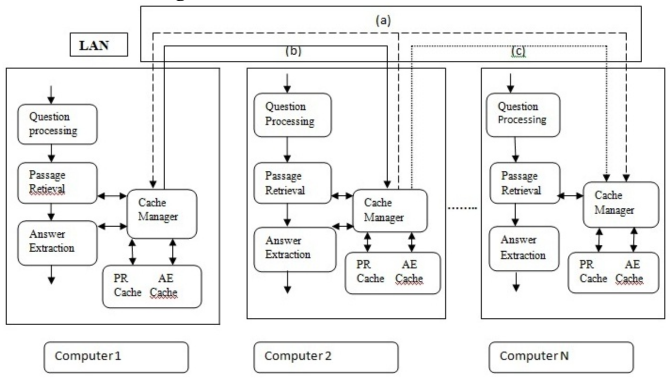

Fig.1. Question Answering System of three sequential blocks QP, PR AND ER.PR and AE can request and store data into the cache using the cache manager.he Question Processing (QP) that parses and analyzes the natural language query given by the user. The Passage Retrieval (PR) that retrieves from the document collection the most relevant documents from the collection and Answer Extraction (AE) that Analyzes the retrieved documents in PR with natural language tools, and returns the most adequate answers to the user.

On top of the search engine, we implement a distributed cooperative cache with no centralized process, which intuitively works like a peer to peer network (with no central node) where all nodes can directly contact the rest of nodes of the network. The communication between nodes is facilitated by the following operations.

- Request/Response: these operations obtain a cached entry from a remote node. Once a node has local miss, it requests the document through a multicast operation (operation (a) in Fig. 1). The request includes the document identifier and a parameter to identify the computing block that requests the data. The receivers of the request respond with the data if the entry is available 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 ©).

The access to the collection content can be optimized by using shared disks a distributed file system optimized for read operation or simply replication the collection if the data set fits in a computer. These architecture can be easily extended to very-large-scale setting. 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

We describe the data structure that we propose in this paper and is shared by all our cooperative caching algorithms, the Evolutive summary Counter (ESC). Then, we explain how to apply the ESC-summaries to the placement (ESC-placement) and the location of data (ESC-search).

A. *Evolutive Summary Counters*

The Evolutive summary counters(ESC) are a data structure deployed in each computing node, which records a window of the recent history of the local accesses. The ESC is composed by a linked list of k Count Bloom filters (CBF) [2]. During a certain period of time, t, the CBF at the head of the list is active, i.e, it counts the occurrences of the elements in a streamed data set. After t time units, a sliding operation is applied, so that a new 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 ESC with k CBF, which monitor the last T. K time units. The ESC- summary is computed as the aggregation of k count filters of the ESC. We evaluated two possible summary implementations:

Plain Summary: A simple addition of all the CBFs.

Liner 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.

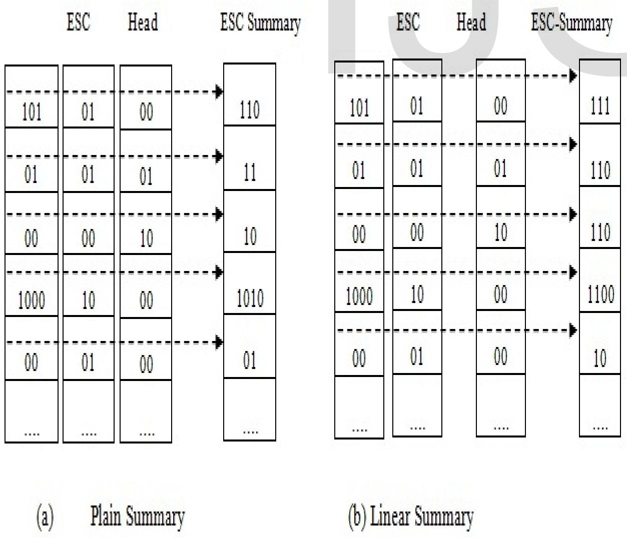

Fig: 2. Summarization example of an ESC (counters are in binary)

B. *ESC-Placement*

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.

Esc-Placement automatically evicts from the cache entries scarcely accessed i.e, documents not read after several forwarding operation. 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, ESC-placement repeats the forwarding procedures until it finds a document that can be discarded because it is not frequently accessed a document with multiple copies in the network.

The ESC-Placement main objectives are send data to where it is being used. Keep at least a cached copy and avoid too much copies not frequent documents and finally it reduce forward chains.

Algorithm 1 : ESC Placement

```
Input: Map <IpAddress, ESCSummary> escMap,
           List<IpAddress> nodesAvailable,
            Document cache Victim
Output: IpAddress

If (cacheVictim.forwardCounter > MAXIMUM
                       FORWARDS)   then
        // Do not forward if the cache victim
          Exceeded   the number of allowed
          Forwards
   return
        NULL;
end

  IPAddress mostAccessedAddr: = NULL;
  int mostAccessedESCValue := -1;
foreach(IpAddress currentAddr : nodesAvailable)
   do
     // Iterate and find the most accessed node
      ESCSummary currentESCSummary :=
                    escMap.get(currentAddr);
 int                   currentESCValue                   :=
currentESCSummary.getCount(cacheVictim.id);
   if (currentESCValue > mostAccessedFrequency)
     then
        mostAccessedAddr := currentAddr;
     mostAccessedESCValue := currentESCValue;
     end
end
return
 most Accessed;
```

C. ESC-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 message may become a bottleneck.

The below diagram shows the ESC-Search Algorithm. The input is a data structure that stores the ESC-summary of each node in the network, the list of all the nodes in the network, and the document identifier that has been missed in the local cache. Initially, the nodes are sorted by their individual probability of having the missing document available. Then, the main loop of the algorithm adds more nodes to query until the estimated probability of avoidable miss is below the threshold

Algorithm 2: ESC Search

```
Input: Map <IpAddress, ESCSummary >
        escMap, List<IpAddress> nodes Available, int
documented
Output: List <IpAddress>

     // Sort the list of nodes by the probability of having
the document cached in each node

List<IpAddress>L:=nodesAvailable.sortByProbability
(escMap, documentId);
int s := 0;
double probAvoidableMiss := ProbAvoidableMiss
(escMap, nodesToQuery);
while (probAvoidableMiss >) ANDs < L.size () )
do
   // Update the avoidable miss probability   until we
reduce it below
   // probAvoidableMiss computes
$P_{AvMiss}$(s,L)probAvoidableMiss :=
probAvoidableMiss(escMap, s, L);
   s++;
end
         // List of nodes that ESC-search
           selects to    query is the sublist of    s nodes.
            List< IpAddress> nodesToQuery := L.subList(0,
         s);
 Return
     nodesToQuery;
```

The below diagram shows the example of ESC-Search.

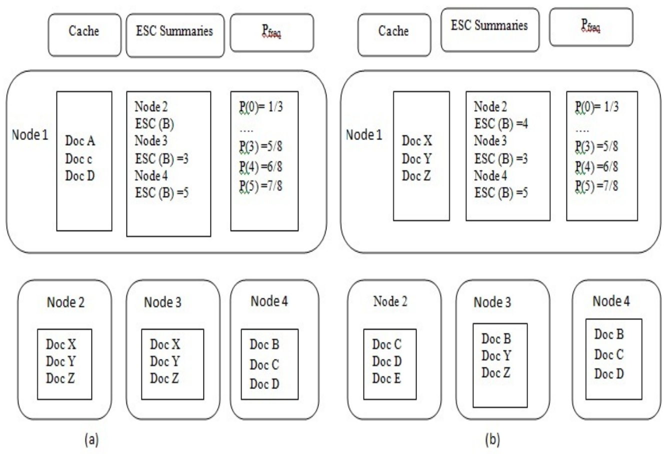

Fig: 3:.Example of ESC-Search

In this example, we show the steps that ESC-search (configured with Ɛ =0.10) performs to locate documents in a network with four computers. The initial state of the system is depicted in Figure 3(a), in which N1 is computing a query that reads "Doc B", but it is not cached locally. The node tries to locate the document in the cooperative cache. First, N1 sorts the node list according to the P_{freq}:L={N4, N2, N3},where N4 is the node with the highest probability of storing a copy of "DocB". Then, N1 calculates the probability of an avoidable miss when no node is queried:

$$P_{AvMiss}(\{\},\text{"Doc B"}) = \left\lfloor 1 - \prod_{i \in N4,N2,N3} \left(1 - P_{freq(ESCS(i,d))}\right) \right\rfloor$$

$$= \left[1 - \frac{1}{8}.\frac{2}{8}.\frac{3}{8}\right] \quad =0.99$$

Likely that the document is available in the cooperative cache), N1 estimates the probabilities of an avoidable miss if more nodes were queried. The first node that is included is the one which is more likely to store the document, which is N4 because "Doc B" has been recently accessed 5 times and its estimated miss probability is the largest:

$$P_{AvMiss}(\{N4\},\text{"Doc B"}) = \left\lfloor \prod_{i \in N4,N2,N3} \left(1 - P_{freq(ESCS(i,d))}\right) \right\rfloor$$

$$\left\lfloor 1 - \prod_{i \in ,N2,N3} \left(1 - P_{freq(ESCS(i,d))}\right) \right\rfloor$$

$$= \left[\frac{1}{8}\right]\left[1 - \frac{2}{8}.\frac{3}{8}\right] = 0.11$$

However, the probability is still above 0.10 and it is necessary to extend the search to more nodes:

$$P_{AvMiss}(\{N4,N2\},\text{"Doc B"}) = \left[\frac{1}{8}\cdot\frac{2}{8}\right]\cdot\left[1-\frac{3}{8}\right]$$

The algorithm finishes here because it estimates that the probability of an avoidable miss is 0.02 if nodes 2 and 4 are queried, which is sufficient to satisfy our probability miss requirements. Note that up to this point there has been no communication among nodes because the ESC-summaries from the other nodes are already stored in N1.Inorder to retrieve "Doc B", N1 sends a request message to nodes 2 and 4 and discovers that the document is only available in N4, which transfers it to N1. Finally, N1 updates the values of P_{freq} according to the final result: (a) it decreases $P_{freq(4)}$ to 6/9 because the document was not found in N2 (where "DocB" was accessed four times; and (b) it increases $P_{freq(5)}$ to 8/9 because the document was found in N4 (where "Doc B" was accessed five times). Later, N1 computes another query that requests "Doc B", which again is not available locally as depicted in Figure 3(b). The steps are similar to the one described for the previous request but the algorithm finishes after only two steps.

$$P_{AvMiss}(\{\},\text{"Doc B"}) = \left[1-\frac{1}{9}\cdot\frac{3}{9}\cdot\frac{3}{8}\right] = 0.99$$

$$P_{AvMiss}(\{Node\ 4\},\text{"Doc B"}) = \left[\frac{1}{9}\right]\cdot\left[1-\frac{3}{9}\cdot\frac{3}{8}\right] = 0.09$$

Therefore, ESC-search would only query N4 but not N2. We observe ESC-search is an algorithm that adapts to the previous experience of the system. Given that in the previous search the document was only available in one node, it updated the corresponding $P_{freq(x)}$ and now it is able to reduce the number of severs queried.

V. CONCLUSION

In this we proposed a state-of-the-art distributed question answering system. The QA System gives extracting answer from the retrieval passage with the help of cache manager. If the data present in the system it will be return otherwise it goes to another nearest system and so on. The core of our distributed cache environment is the ESC, which is an efficient data structure that captures the frequency of accesses to the document rom big text collections and generates summaries efficiently. The user performance can be evaluated with the help of ESC-Placement and ESC-Search Algorithm. In this we can increase the file accessed with the help of cached algorithms and also the user can search the client have that file is in cache memory or not.

REFERENCES

[1]. D. Roussinov, W. Fan, and J. Robles-Flores, "Beyond keywords: auto-mated question answering on the web," Commun. ACM, vol. 51, no.9, pp. 60–65, 2008.
[2]. A. Broder and M. Mitzenmacher, "Network applications of bloom filters:A survey," Internet
Mathematics, vol. 1, no. 4, 2003
[3] M. Surdeanu, D. Moldovan, and S. Harabagiu, "Performance analysis of a distributed question/answering system," IEEE Trans. Parallel Distrib.Syst., vol. 13, no. 6, pp. 579–596, 2002.
[4]. D. Dominguez-Sal, M. Surdeanu, J. Aguilar-Saborit, and J. Larriba-Pey, "Cache-aware load balancing for question answering," in CIKM, 2008, pp. 1271–1280.
[5].M. Dahlin, R. Wang, T. Anderson, and D. Patterson, "Cooperative caching: using remote client memory to improve file system performance," in OSDI, 1994, pp. 267–280..
[6]. I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek,F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Trans. Netw., vol. 11,
no. 1, pp. 17–32, 2003.
[7] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in Middleware, ser. LNCS, vol. 2218. Springer, 2001, pp. 329–350.
[8].T. Cortes, S. Girona, and J. Labarta, "Design issues of a cooperative cache with no coherence problems," in IOPADS, 1997, pp. 37–46.
[9] P. Sarkar and J. Hartman, "Efficient cooperative caching using hints," in OSDI, 1996, pp. 35–46.
[10.]. 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.
[11]S. Annapureddy, M. Freedman, and D. Mazi`eres, "Shark: Scaling file servers via cooperative caching," in NSDI, 2005.
[12] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a decentralized peer-to-peer web cache," in PODC, 2002, pp. 213–222.