Caching in Multi-Agent based Architecture for Distributed Information Retrieval Systems

Hemali Majithia
Center for Architecture and Advanced Data-driven Information Processing
Dept. of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, MD 21250
hema1@cs.umbc.edu

Abstract—
Caching is an effective technique for improving performance in Databases and Information Retrieval (IR) systems. Traditional IR systems access the collection indices to perform searches. Such searches on large corpora for queries often repeated can be computationally redundant. In addition, querying remote sources can be expensive because of large communication overheads and frequent unavailability (if dealing with proprietary databases). One way of dealing with this problem is to cache query results. This can provide results to similar future queries. In this paper, I focus on problems related to caching in Distributed IR (DIR) systems. An investigation of research issues like caching of documents and metadata routing is provided. Caching implementation efforts are tested on the CARROT-II DIR system. Caching has been known to have a strong effect on query response times and throughput. The goal of this project is to provide efficient caching mechanism and design proper cache maintenance strategies that will improve performance in a distributed retrieval environment.

I. INTRODUCTION

CARROT-II [4] [3] (Collaborative Agent-based Routing and Retrieval of Text), is an ongoing effort in the field of DIR. It is an agent-based architecture for DIR and document collection management. In CARROT II, agents index and provide search services over local document collections, which might be hosted on a single web server. A CARROT II system can consist of an arbitrary number of agents, distributed across a variety of platforms and locations. CARROT II agents advertise content-derived metadata that describes their local document store; this metadata is sent to other CARROT II agents which agree to act as brokers for that collection - every agent in the system has the ability to serve as such a broker. A user query is sent to any CARROT II agent, which can decide to answer the query itself from its local collection, or to send the query on to other agents whose metadata indicate that they would be able to answer the query, or send the query on further. Search results from multiple agents are merged and returned to the user. In a scenario like this, the inter-agent communication across the architecture causes a huge overhead due to network latency and traffic. By caching the queries and documents (if necessary) at the agent's end itself, this overhead can be relieved to a great extent. As can be seen in the following discussion on related work, most modern-day searching strategies implement caching to increase availability of results together with improvement in the execution time of a query.

In this paper, I put forth the idea of caching in the CARROT-II system to maximise performance. The target is to handle the issue of improving the performance without giving up or compromising on the efficiency of the system. Along with caching of queries with the retrieved results, I intend to cache the documents themselves at a certain threshold. I also plan to look at the influence of routing queries by maintaining meta-data caches and using a multi-source caching architecture that considers all operational cases (equivalence, containment and intersection).

II. BACKGROUND

Caching documents [8] [7] to reduce access latency is extensively used on the web. Most web browsers cache documents in the client's main memory and disk. To improve cache hit rates by aggregating the requests of several users, caching proxies are widely used. Proxies employ large caches which they use to serve a stream of requests coming from a large number of clients. Since even large caches may eventually fill up, cache replacement policies have been the subject of intensive recent research. One of the first cache replacement policies considered was LRU (Least Recently Used) and its variations. LRU is based on the heuristic that the documents not used recently will probably not be requested in the near future. After studying the access patterns of several web clients, it was realized that caching small documents may result in better performance (especially in a memory-limited environment). Thus, LRU was extended with heuristics that favor caching of small documents, ei-
ther by precluding caching of large documents, or by eagerly removing large documents. Given that clients experience different latency when communicating with different servers, cache replacement policies may also take network latency into account in order to avoid replacing documents that may take a lot of time to download. Some policies aggregate all the above factors (access recently, size, latency) into a weight or value and try to keep in the cache the documents with the largest values. To improve hit rates even further, cache proxies are frequently organized into a hierarchy that can achieve better performance. Other approaches identify popular documents which are then cached for a predefined period of time (usually a day or so). Finally, some caches employ intelligent prefetching methods to improve the hit rate even further.

Chilickovski et al. [2] have done extensive research in semantic caching strategies to optimize queries as opposed to standard keyword-based query matching. I intend to follow a similar strategy here, looking beyond simple pattern-matching to draw up results and caching the same. This also comes into play when retrieving results from a local cache. In many cases, users refine queries, simply by adding or removing a term or two in the process. In such a case, semantic query matching would prove to be effective in computing the results and also in looking up the cache. To achieve this, the query can be split up into the probe query which retrieves the results from the local cache and the remainder query which then goes out and tries to retrieve the result of the rest of the query from the web, or in this case, from another agent’s cache.

Spink et al. [6] [10] have analyzed the transaction logs of queries posed by users of EXCITE, a major Internet Search Engine. They have focused on how do users search the web, and on what do they search on the web. They present a large number of findings. Among the most interesting ones are that users usually ask short queries and view no more than 2-3 pages of the query’s results. This gives some leeway in storing a query’s results and reduces the space requirements for the same.

Query result caching has been previously studied in order to evaluate new queries based on the results of previous queries with similar constraints, or in cases when the source database is not readily available. The focus of previous work in query result caching was to reduce the cost of query execution (communication and computation overhead) in a distributed database system by caching the results of similar queries, while the focus of my work is to also serve as basis for query routing in distributed systems.

Summarizing, although there has been significant work on web caching, there has been no previous systematic study on the existence of locality in Search Engine queries, and of evaluating the performance of caching those queries’ results. In this paper I want to show that caching in distributed environment is significant. In addition also evaluate several caching methods that exploit the existing locality.

III. RELATED WORK

There have been several techniques proposed for caching. In addition query clustering is an issue which closely related to caching. Techniques have been proposed with respect to a particular architecture. However I plan to make use of these techniques and apply them to a distributed environment. The Harvest system is a system closely associated with the CARROT-II system. The Harvest indexer offers a distributed solution to the problems of indexing data made available on the web. With each web server running a local Gatherer feeding into a central Broker many of the problems of web crawling are avoided. Caching problem is Harvest is tackled with caching the requests that are made to the server. As mentioned earlier, the algorithm proposed [2] does not use direct pattern matching. Instead, it proposes splitting up the query to take advantage of the cache. The query can be split up into the probe query whose results are present in the cache and the remainder query whose results need to be fetched from the agent/node containing the matching results. A standard querying language has been proposed for more efficient retrieval of results. In my design, instead of returning all results of a query, i can instead choose to return only the top n-ranked queries, ranked by order of their importance (degree of matching). The algorithm used by Boris takes into account checkable and non-checkable query results. It strongly advocates use of the LRU algorithm as a replacement strategy for cache replacement. In case there are more than one contender for replacement, the one with the shortest retrieval time is chosen as the victim. Data caching has been intensively studied in the context of database management systems. Intelligent query caching has been used in the SIMS project [1]. Query Caching and Optimization in Distributed Mediator Systems [9] is based on the invariant mechanism and uses query rewriting techniques and semantic information about sources to collect some source statistics and build optimal query plans. However, the mechanism assumes the subquery equivalence and does not consider the containment and intersection cases. A semantic cache [2] mechanism for web queries based on signature files has been proposed. This method uses signature based region descriptions to efficiently manage both containment (entire query in cache) and intersection (partial
query stored in cache) cases. However the cache structure proposed here is for interrogating one information source only. The same scenario can be extended to distributed database systems, where chunks or slices of the database are maintained by different entities. Here special efforts must be taken to maintain disconnected operation in case of network partitioning [5].

IV. RESEARCH ISSUES

There are a number of different issues which affect the design and operation of the caching mechanism in a distributed system like CARROT-II. The important issues involve mainly cache maintenance and consistency issues.

A. Cache Maintenance

A key issue is the way in which caches should be maintained to maximize the performance. Caches can be maintained either on disk or in memory. It is essential to use a maintenance strategy which is going to be efficient. This issue is of prime importance because if the cache is maintained in the memory, and the system is very large then there is going to be considerable memory constraints. Hence the cache should be small in size. On the other hand, if the cache is stored on disk, there will be a considerable lag while loading the cache from disk to memory if the cache is very large in size. This gives rise to the issue of the cache size. Another issue is whether a global cache should be maintained or a local cache at each node will be more efficient. In case of a local cache the issue of cache coherency arises whereas in case of a global cache, issues like single point of failure and overloading need to be considered. Other issues involved are structure of the cache, cache storage, page sizes, presence or absence of prefetching mechanisms, etc.

B. Caching Algorithm

Selection of the caching algorithm is the most significant problem. The caching algorithm should be able to maximize the performance in terms of efficiency of query retrieval, prefetching, page replacement, etc. The structure of the cache (in-memory or on-disk) would play an important role in deciding which algorithm to use. The LRU policy is one of the most commonly used caching strategies. The LRU scheme is the easiest to implement and the initial stage of the design will use this algorithm with a view to replacing it by a more efficient algorithm once the prototype for the entire system is built up. The initial strategy would be to use a scheme where incoming queries and their results are cached in a queue based on their arrival times, and the cache management module would, in the event of a full cache, replace the least recently referenced one by an incoming query and its results. When a query is referenced again while it is in this cache queue, it is simply moved to the tail of the queue to indicate latest time of reference.

C. Cache Coherency and Update Strategies

In case of distributed local caches, coherency is of prime importance. Using the concept of hints in a cache, a delayed write mechanism could be implemented. In this manner, updates to other caches can be done in a batch manner instead of propagating individual updates. This is a desirable quality in caching as it provides a reduction in the traffic in the system (if the system is very large i.e. a large number of agents exist, there will be considerable traffic in the system).

D. Measuring the Level of Query Similarity

Effective use of the query in the cache is an essential feature of a caching mechanism. In DIR systems, a query need not be an exact match. Hence a query which is similar to the query in the cache can be treated as an exact query. However this is to decide on the basis of some similarity factor. The cosine similarity measure between the new queries and the queries existing in the cache can taken as the decision factor. However the key issue is that the efficiency of the results along with the performance and throughput should not be affected.
E. Caching Influence on Query Routing Mechanism

In the CARROT II system, agents perform query routing primarily by submitting the query to any arbitrary part of the system. A CARROT II agent first tries to answer the query out of its own information store and if this fails, passes on this query to other agents whose metadata indicate that they are capable of answering the query. Since this method involves rerouting of queries on a large scale, caching of the results at the source of the request would enhance the performance. The proposed method is to either cache the results (together with the query) at the agent itself or in case of results being fetched from agents running under non-local nodes, at the node level. As with any distributed database system, query caching would involve issues such as cache coherency, replacement policy, replication policy and a query resolution mechanism that would take into consideration distributed cache content. Cache coherency would require a mechanism for validating cached data periodically with the actual stored data. The replication policy indicates which content needs to be replicated/cached across multiple agents/nodes (e.g., in case a query is accessed very frequently, it would need to be cached locally and be assigned a higher priority to ensure it is the least likely to be swapped out in case of cache replacement, assignment of priorities would be necessary to avoid redundancy as well as thrashing). The query resolution mechanism would need to define how the query routing would be done effectively. Mechanisms to deal with failure of individual nodes/agents would also need to be examined.

V. CARROT-II

Carrot-II (see 2) as explained before is a large-scale, distributed query routing and IR system which serves as a testbed for a large variety of important research problems. The CARROT II system is implemented with wrappers that extend various IR engines like MG [11]. It also uses the SIRE (InHouse) IR engine as a testbed for heterogeneous indexing data sources. The initial goal of the CARROT II project is integration of several heterogeneous information sources into a single metadata based search system. CARROT II system is flexible in a way that it can be easily created, configured and accessed by another information system, and so can be employed to extend the search capabilities of an existing project. There are various issues related to the CARROT II architecture for example, meta-data routing, representation and manipulation of metadata, query routing and results fusion.

A. Architecture of Carrot-II System

There are three essential components to the CARROT II architecture namely, a collection of distributed CARROT II agents which perform the central work. There is a network of infrastructure agents (node agents, platform agents and cluster agents) that facilitate control and communication of the system. Lastly, there exists a small set of support agents that facilitates access to the system and coordinates its activities (ans, collection manager agent, logger agent and query manager agent). The CARROT II agents form the core or heart of the CARROT II system. The role of each CARROT II agent is to manage a certain corpus, accept queries, and either answer them itself or to forward them to other CARROT II agents in the system. This is achieved by each CARROT II agent creating and distributing metadata describing its own corpus.

B. CARROT II Infrastructure & Support Agents

The CARROT II agent is a Java agent and is responsible for one information source. It uses Jackal as a communication tool to communicate with other agents in the system. The infrastructure of CARROT II system is hierarchical (see 3). The CARROT II system starts up by starting the Master node, which in turn starts a node agent on each participating machine. The node is divided into independent Java Virtual Machines or Platforms, each governed by a Platform Agent. This Agent in turn creates a set of Cluster Agents and these agents maintain a single instance of Jackal. Each Cluster Agent creates a series of CAR-
KAROT II agents. In addition to the above agents, there is an Agent Name Server that provides a basic communication facilitation among the agents. The Master Agent manages the operation of the entire system. The Logger Agent monitors log traffic and allows the system to assemble information about the details of the operation. Lastly, the Collection Manager serves in distributing the data and metadata.

VI. CACHING IN DISTRIBUTED INFORMATION RETRIEVAL SYSTEM (CARROT-II SYSTEM)

The idea of caching in distributed IR is a current research topic. I focus on experimentally analysing the effect of caching on the CARROT II system which is an essential part of my research. Carrot-II is the successor of an earlier project (Collaborative Agent-based Routing and Retrieval of Text) (originally CAFE). This system is composed of a flexible hierarchy of query routing agents. These agents communicate with one another using KQML and the Jackal platform, and may be distributed across the Internet. Agents interact with information sources via a well-defined interface. Queries presented to any agent in the system are routed, based on the content of the query and metadata about the contents of the servers, to the appropriate destination. Agents themselves are uniform and extremely simple. The flexibility of this system opens up a wide range of opportunities.

Having described the idea of caching in a distributed IR environment, I intend to implement this in the Carrot-II scenario describing the architecture and implementation of the caching mechanism in the Carrot-II system. Consider the nodes in the CARROT II system each residing on a data source. When a query enters the CARROT II system, initially it routed to any CARROT II agent in the system. The agent receiving the query first checks in it's cache. This is done by using exact matching or cosine similarity between the queries. If a match is obtained the results are directly fetched from the cache indicating cache hit. However if the agents encounters a cache miss, it checks into its metadata pool. If it is capable of answering the query, it performs a search and updates it's local cache using the LRU algorithm. There is also a global cache maintained at each of the nodes. The query is updated in the global node cache too. However if the agent is not capable of answering the query, it compares the query with the metadata of the other agents and forwards the query to the most relevant agent. Currently a broadcasting mechanism is used for metadata distribution. However this is not very efficient when the system has a large number of agents. Hence it will difficult to know where the query should be routed in a efficient way. The global caches will be beneficial in this aspect. The global caches will maintain information about queries that have been answered by CARROT II agents residing on that node. In this way, when a CARROT II agent who is not capable of answering the query and does not find the most relevant CARROT II agent to route the query to, it will do a lookup on the global cache. The query is routed to the CARROT II agent found in the global cache. In addition the CARROT II agent can request metadata for that particular agent where the query is routed. In this way the broadcasting mechanism of metadata distribution can be avoided and thereby will result in increased performance.

A. Effect of Caching on Carrot-II and it’s Measurement

The effect of caching mechanism can be measured in a number of ways in a distributed IR system. I propose to show that caching brings about considerable increase in the performance and throughput by measuring several quantities of prime importance in an IR system. Firstly measure the precision and recall values using the caching mechanism. This is compared against the precision and recall values of the same corpora when caching is not used. The precision and recall values confirm the efficiency of retrieval of documents. This is done when a scored corpora is used. Precision is the calculated as a fraction of retrieved documents that are relevant. Similarly recall is measured as a fraction of relevant documents that are retrieved. The performance of the system is shown to increase considerably by evaluating the time taken for
retrieval when the queries are cached against the time taken to retrieve the documents when no caching is used. The precision and recall values can be checked for only when scored corpora is used. Caching of the queries along with their results will show a considerable improvement in performance when a large number of agents exist in the CARROT II system. As a result all the CARROT II agents will not have information of the metadata of the other CARROT II agents and in such a scenario caching will play a significant role. With the help of global caches the routing of queries will be more or less accurate and considerable amount of time will be saved in results retrieval.

VII. IMPLEMENTATION

Using CARROT-II's architecture, I plan to implement a local cache at each node of the Beowulf. The caching algorithm will be a combination of the algorithm proposed by Boris [2]. Initially I intend to use the simple approach of storing the entire query and it's result in the cache. I plan to use a cache in the form of a file which maintains the queries and it's corresponding results. The replacement policy will be the LRU. The global caches at each node will also be maintained in a similar fashion. These caches will hold the queries which have been fired by the agents on that node. A lookup on the global cache will serve as means of routing the query to some other node. This serves as a distributed system because each node can be treated as a stand alone machine. The implementation is going to be done in Java. I have chosen Java because of it's platform independent nature and because of the presence of numerous threads in the Jackal communication tool.

VIII. CONCLUSION

In this paper, I propose the technique of caching in Distributed Information Retrieval systems. By applying this technique to the Carrot-II system, I intend to improve query throughput by providing higher availability of query results through replication via caching in addition to improving the performance of the system by caching documents together with the queries.

REFERENCES
