Monitoring the Dynamic Web to respond to Continuous Queries: A Demonstration

Sandeep Pandey
Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai-400076, India
pandey@cse.iitb.ac.in

Krithi Ramamritham
Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai-400076, India
krithi@cse.iitb.ac.in

Soumen Chakrabarti
Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai-400076, India
soumen@cse.iitb.ac.in

ABSTRACT

Our **C**ontinuous **A**daptive **M**onitoring (CAM) system provides responses for *continuous queries* by monitoring and extracting information scattered across the web. Continuous queries are the queries for which responses given to users must be continuously updated, as the sources of interest get updated. Such queries occur, for instance, during on-line decision making, e.g., traffic flow control, weather monitoring etc. Whereas push-based techniques may be able to refresh query results meeting user requirements, they do not scale well. With the pull based approach, the problem of keeping the responses current reduces to the problem of deciding how often to visit a source to determine if and how it has been modified so that a user response can be updated accordingly. As should be evident, periodical monitoring is not scalable. Also, it can lead to huge wastage of monitoring resources. Hence CAM employs a multiphase approach. In the *tracking phase*, changes to an initially identified set of relevant pages, are tracked. From the observed change characteristics of these pages, a probabilistic model of their change behaviour is formulated and weights are assigned to pages to denote their importance for the current queries. Based on these statistics, during the next phase, the *Resource Allocation* phase, resources, needed to continuously monitor these pages for changes, are allocated. Given these resource allocations, the *Scheduling* phase produces an optimal achievable schedule of monitoring. An experimental evaluation of our approach compared to prior approaches on synthetic data for crawling dynamic web pages shows the effectiveness of our approach to monitoring dynamic changes. For example, by monitoring just 5% of the possible change instances, CAM is able to return 90% of the changed information to the users. In this demonstration, we show how CAM keeps users up-to-date with respect to a set of ongoing sports related events.

1. MOTIVATION BEHIND CAM

The World Wide Web consists of an ever-increasing collection of decentralized web pages that are modified at unspecified times by their owners. Current search engines try to keep up with the dynamics of web by crawling it periodically, in the process building an index that allows better search for pages relevant to a topic or a set of keywords. Clearly, any good crawling technique needs to consider the change behaviour of web pages. But, the algorithms used for crawling and the typical frequency of crawling are insufficient to handle a class of queries known as Continuous Queries (for example, see[6]) in which the user expects to be continuously updated as and when new information of relevance to his/her query becomes available. For example, consider a user who wants to monitor a hurricane in progress with the view of knowing how his/her town will be affected by the hurricane. Obviously, a system which responds taking into account the continuous updates to the relevant web pages will serve the users better than another which, say, treats the query as a *discrete query*, i.e., returns an answer only when the query is submitted.

Copyright is held by the author/owner(s).
WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

Not surprisingly, the problem of keeping track of the dynamics of the web becomes inherently different for the continuous query case compared to the discrete query case. We use the term *monitoring* to explicitly account for the differences from the classical crawling problem. A *monitoring task* fetches a web page, much like a crawler does, but with the goal of fetching new information relevant to one or more queries while a *crawl* is not done with any specific user request in mind. The work involved in handling continuous queries is portrayed in Figure 1. For continuous queries, since the system should maintain the *currency* of responses to users, the problem translates to one of (a) knowing which pages are relevant, (b) tracking the changes to the pages, to determine the characteristics of changes to these pages, and from these, (c) deciding when to monitor the pages for changes, so that responses are current. The last problem has several subproblems: allocating the resources needed for monitoring the pages, scheduling the actual monitoring tasks, and then monitoring. The algorithm for doing this has been proposed in [9]. Specifically, they address the problem of distributing a given number of monitoring tasks among the pages whose changes need to be tracked so as to respond to a set of continuous queries. In Figure 1, the feedback arcs from the monitoring phase to the earlier phases indicate that observations made during the monitoring phase can be used to adjust subsequent decisions.

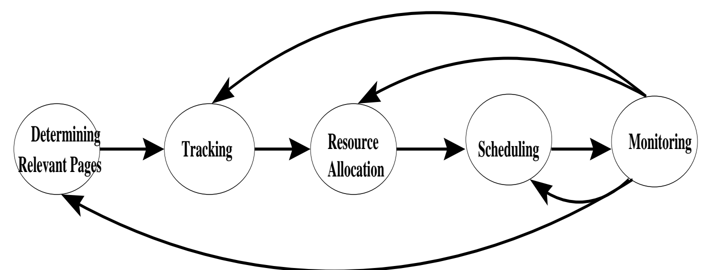

Figure 1: Different phases of our approach

It could be argued that discrete queries posed every so often can be considered to be equivalent to continuous queries but the following reasons should help dispel this misconception: First, de-

termining the next time when the discrete query should be posed by the user is highly non-trivial. If the time-interval is kept small then it may induce unnecessary load on the system, particularly when the updates are not frequent. If we set the time-interval to be large, it may lead to loss of information if updates are more frequent than expected. Second, continuous queries have a non-zero lifetime and so a query system can study a query's characteristics carefully and can answer it more efficiently than in the case where discrete queries, which have zero lifetime, are continuously posed. Furthermore, unlike in the case of discrete queries, the time taken to provide the system's first response to a continuous query may not be as important as the maintenance of currency during all the responses. This discussion makes it clear that not only the nature of the crawling problem but optimization goals also become different when we move from discrete to continuous query case.

2. OVERVIEW OF THE CAM APPROACH

Consider a user who is worried about a hurricane in progress and wants to keep abreast of the hurricane-related updates. To achieve this, he poses a continuous m-keyword query $q = \{w_1, w_2, \ldots w_m\}$. In this section, we present an overview of the major ingredients of the CAM approach.

Identifying Pages Relevant to a Set of Queries: Based on the keywords specified by a user, we first identify pages relevant to this query. The query is fed to an inverted index which in turn returns a set of pages relevant to the queries. We find, say, that *the National Hurricane Center*, *National Weather Organization*, and other tropical cyclone sites as well as news sites are relevant. The relevance of a page to a query can be measured by standard IR techniques based on the *Vector-Space* model (see Appendix B for details).

Tracking the Changes to Relevant Pages to Characterize Changes: Once relevant pages have been identified, by visiting each page at frequent intervals during a tracking period, changes to these pages are tracked, update statistics collected, and the relevance of the changes, vis a vis the queries, is assessed. This is used to build a statistical model of the changes to the pages relevant to a set of queries. These statistics include page update instances, page change frequency, and relevance of the changes to the pages for current queries.

Let Q denote the set of all queries submitted in the system and ω_i denote the importance of ith query. These are input to the system. Let P denote the set of web pages relevant to the continuous queries, Q_{p_i} be the set of queries for which i^{th} page is found to be relevant, and $r_{i,j}$ be the estimated relevance of ith page for j^{th} query. It is positive for all queries q$\in Q_{p_i}$ and zero for all $q \in Q$ - Q_{p_i}. These relevance measures are initially calculated during the tracking period (and get updated, as explained later, after every monitoring epoch).

It is clear that not all pages will be equally important for each query in the system. So we rank the pages by assigning a *weight* to each page using its relevance for queries. The *weight* of a page, computed as $\sum_{j \in Q}(\omega_j r_{i,j})$, denotes the value of current version of the page. If the page gets updated before its current version is *monitored*, we assume that we incur a loss of W_i.

Considerations underlying the Monitoring of Changes:

CAM does its monitoring in epochs, each epoch is of duration T time units. The purpose of the resource allocation phase is to decide how to allocate monitoring resources for an epoch and the goal of scheduling is to decide when a monitoring task should execute, given the resource allocation decisions. Monitoring is done by a monitoring task where the task includes fetching a specified page from its source and determining if it has changed and if so applying the changes to return new results for those queries for which that page is relevant.

Let C denote the total number of *monitoring tasks* that can be employed in a single monitoring epoch. C is derived as an aggregation of the resources needed for monitoring, including CPU cycles, communication bandwidth, and memory[1].

λ_i is the estimated number of changes that occur in page i in T time units. Henceforth we will call it the *change frequency* for page i. Suppose U_i denotes the sequence of time instances $u_{i,1}$, $u_{i,2}......u_{i,p_i}$ at which the tracking phase determines that possible updates occur to page i. We assume $0 \leq u_{i,1} \leq u_{i,2}.....u_{i,p_i} \leq T$ and $u_{i,0} = 0$ and $u_{i,p_i} = T$. p_i is the total number of update instances for i^{th} page during T, i.e., cardinality of sequence U_i (p_i=$|U_i|$). Note that a page may not be updated at these time instances and so there is a probability $\rho_{i,j}$ associated with each time instance $u_{i,j}$ that denotes the chances of ith page being updated at the jth instance. The overall goal of the resource allocation and scheduling phases is to monitor in such a way that the monitoring events occur just after updates are expected to take place. The number of missed updates is an indication of the amount of lost information and minimizing this is the goal of the system.

With these considerations in mind, decisions are made about the allocation of a given number of monitoring tasks among a set of relevant pages while also deciding when these allocated monitoring tasks should ideally occur within an epoch. The basic idea is that these monitoring epochs of length T repeat every T units of time and we will make decisions pertaining to the monitoring tasks to be carried out in one monitoring epoch using both new data and the

3. ARCHITECTURE

We first describe the architecture of CAM by briefly discussing all the components of CAM and their functionality Next we take a walk through the system elucidating the sequence of actions CAM employs for answering continuous queries.

3.1 Nuts and Bolts of CAM

We have three main independent components in our architecture: client, server, and fetching unit. Each of these components can reside on a seperate machine or multiple components on multiple machines or all on the same machine. A sketch of the system architecture is given in Figure 1.

The client unit currently has five subcomponents: (1) The registration manager which allows clients to register with the system using a valid user id and password, and returns to the clients a confirmation on their registration, (2) The query interface through which a user can submit a new continuous query in the system (3) The query parser, (4) The query installer assigns each query a unique idenifier, storing it in the query database in a pre-decided format and initiating a thread for monitoring, (5) The client services which provide utilities for browsing or updating installed continual queries and for tracing the performance of update monitoring of source data.

The second unit is the system server which consists of three main subcomponents: (1) continual query thread (CQT) with continual query manager, aggregrator, and the query evaluator, (2) Resource Manager, and (3) the Index Rebuilder. Their functionalities are as

[1]For example, the authors of [5] report that with two 533 MHz Alpha processors, 2 GB of RAM, 118 GB of local disk, a 100 Mbit/sec FDDI connection to the Internet, and *Mercator* under srcjava, their crawler crawled at an average download rate of 112 documents/sec and 1,682 KB/sec. Similarly the capabilities of a given infrastructure can be mapped to the number of *monitoring tasks* that it is capable of on average.

described below:

- Once a query gets installed, a CQ Thread is initiated for it which interacts with other units and takes care of all the processings required for the query. It consists of a continual query manager, aggregrator and the query evaluator.

  – Continual query manager is responsible to coordinate with the aggregrator and query evaluator to monitor updates of interest, and coordinate with crawlers to track the new updates of the source data.

  – Query evaluator is the unit which determines monitoring sources, tracks them and then performs efficient monitoring with the help of Resource Manager and Inverted Index.

    * When the user poses a query, the source identifer examines the query and determines which sites contain information that is relevant to the user's request. Consequently, instead of contacting all the available data sources, monitoring is only done for the selected sites that can actually contribute to the query. Determining of relevant sites is achieved using inverted index in CAM system. But to remain in synchronization with the web, inverted index is required to be updated frequently following changes of web pages which becomes a intimidating task, especially because system has to deal with size and dynamics of web as well as its randomness.

    * Once relevant pages have been identified, by visiting each page at frequent intervals during a tracking period, changes to these pages are tracked, update statistics are collected, and the relevance of the changes, vis a vis the queries, is assessed. This is used to build a statistical model of the changes to the pages relevant to a set of queries. These statistics include page update instances, page change frequency, and relevance of the changes to the pages for current queries.

    * CAM does its monitoring in epochs, each epoch is of duration T time units. The purpose of resource manager is to allocate monitoring resources among web pages for an epoch. It also decides the time instances when a monitoring task should execute, given the resource allocation decisions. Monitoring is done by performing a monitoring task which includes fetching of a specified page from its source, determining if it has changed and if so, then propagating those changes to the aggregrator to prepare new results for the queries for which that page is relevant.

  – Aggregrator is the in-charge of compiling the information collected from various monitoring sources. It removes the junk from downloaded pages, looks for changes and on founding them to be interesting, propagates to the database storing the results of query.

- Resource Manager allocates the crawling resources among sites required to be monitored. It gets page information (pattern of changes, relevance etc.) from query evaluator (Tracker) and then accordingly allocates crawling resources to sites. It also prepares an approximate optimal schedule for monitoring which query evaluator (monitor) uses for the purpose of probing at the source sites.

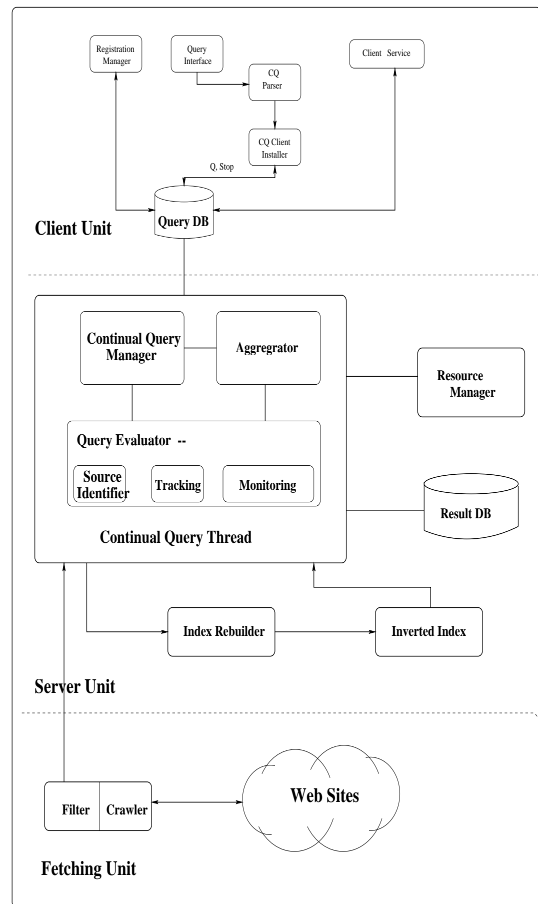

Figure 2: **System Architecture**

- The third unit is the fetching component. It consists of filters and crawlers. The cralwer, on behalf of the query evaluator, connects to information sources and fetches pages. The Filter checks if the newly fetched copy is more recent than the one already kept in the repository and packages (translates) the response from the corresponding data source site into the object format used by the query system. Instead of periodically polling the sites, which will consume a lot of resources, we use a novel scheme for monitoring which exploits the prior knowledge of change patterns captured during the tracking phase.

3.2 Query evaluation in CAM

Here we discuss how a client can use CAM for obtaining continuoisly updated answers to his/her queries. We walk through the process CAM uses for providing answers.

1. Client connects to the registration Manager and registers himself.

2. The registration manager provides the clients with valid user id and password, and returns the client a confirmation on his registration.

3. The client can submit a new query using query interface. The client services also provides the clients facility of deleting/updating any of his previously submitted queries and viewing results of installed queries.

4. The query parser parses the form request and construct the key components of a continuous query (*Q, Stop*).

5. The query installer installs the query by giving it a unique identifier which serves as the primary key for all the information stored about this query in the repository. Also a process/thread is started for further processing of the CQ.

6. The Continuous query manager coordinates with the aggregrator and query evaluator to monitor updates of interest, and coordinates with crawlers to track the new updates to the source data.

7. The Source Identifier determines the sources relevant to the query using the inverted index. Inverted Index maps word to web pages/sites.

8. Maintaining Inverted Index: As mentioned above, it becomes essential to keep inverted index up-to-date with the web pages. The naive way of doing this will be to periodically visit each and every page of repository and refresh it. But this method is not only expensive but also very ineffective primarily for two reasons.

   - Periodically visiting each page will mean massive resources and so a large period of refreshing too.
   - Doesn't take into account the importance and dynamics of individual pages.

   We solve the problem efficiently by mapping it into a linear optimization problem.

9. These sources of information are tracked in tracking phase using crawlers. On detecting any update, these crawlers alert the index rebuilder and aggregrator which in turn updates inverted index and results database respectively. Also different parameters of page information are estimated and are supplied to the resource manager. We use standard OR techniques for predicting change pattern of page (ex. moving average/exponential smoothing/trend/ seasonal/fourier/cyclic) and our experiments show the benefit of using these forecasting models for monitoring rather than naively probing it in periodical fashion.

10. Resource Manager in turn prepares a monitoring schedule using the parameters supplied during tracking.

11. According to the above prepared schedule, crawlers probe source sites during monitoring phase. Relevant information is supplied to interested aggregrators.

12. Aggregrators prune the junk data and select out the information relevant to query which is also more recent than the information already stored in result database. This requires aggregrator to find out how is newly recevied data related to the previously stored one and even if it is related, then which part of it is actually new and is not only a sole repetition of the previous one.

4. THE DEMONSTRATED SCENARIO

5. CONCLUSIONS AND RELATED WORK

In this paper, we described the achitecture of CAM, a continuous query answering system. There are several systems developed for monitoring sources on the web. CONQUER[8], WebCQ[7] and C3 are the most related project to our work. The most evident differences between CAM and any of these related works is the approach of monitoring. The way source site are monitored in CAM is more efficient and optimal. CAM keeps responses to continuous queries current by focusing on the problem of dynamically monitoring the sources of information relevant to the queries. From the change characteristics of these pages—observed in a tracking phase, a probabilistic model of their change behaviour is formulated and weights are assigned to pages to denote their importance for the current queries. During the Resource Allocation phase, based on these statistics, resources, needed to continuously monitor these pages for changes, are allocated. Given these resource allocations, the Scheduling phase produces an optimal achievable schedule for monitoring.

Also there have been several studies of web crawling in an attempt of capturing web dynamics. The earliest study to our knowledge is by Brewington and Cybenko. In [1], they not only studied the dynamics of web but also raise some very interesting issues for developing better crawling techniques. They showed that page change behaviour varies significantly from page to page and so crawling them equal number of times is a fallacious technique. [3] and [2] address a number of issues relating to the design of effective crawlers. In [4][10], authors approached the problem formally and devised an optimal crawling technique. (Some aspects of our formal are adopted from [10] and modified to suit our problem definition.) A common assumption made in most of these studies is that page changes are a *Poisson* or *memoryless* process. In fact it has shown to hold true for a large set of pages but it is also found in [1] that most of web pages are modified during US working hours, *i.e.,* 5 a.m. to 5 p.m. In CAM, we go beyond these assumptions and present an optimal monitoring technique for answering continuous queries independent of any assumption about page change behaviour. Instead, we collect and build page change statistics during a tracking period and only on the basis of this collected information, we do resource allocation. Then we keep on updating this information after every T time units based on the result of the monitoring done. This makes our solution robust and adaptable in any web scenario.

6. REFERENCES

[1] B. E. Brewington and G. Cybenko. How dynamic is the Web? *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):257–276, 2000.

[2] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.

[3] J. Cho and H. García-Molina. Synchronizing a database to improve freshness. *In Proceedings of 2000 ACM International Conference on Management of Data(SIGMOD)*, 30(1–7):161–172, 2000.

[4] E. Coffman, J. Z. Liu, and R. R. Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1998.

[5] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[6] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *Knowledge and Data Engineering*, 11(4):610–628, 1999.

[7] L. Liu, C. Pu, and W. Tang. Webcq: Detecting and delivering information changes on the web. *In Proc. Int. Conf. on Information and Knowledge Management (CIKM)*, 2000.

[8] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A continual query system for update monitoring in the www. *International Journal of Computer Systems, Science and Engineering*, 1999.

[9] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. *World Wide Web*, 2003.

[10] J. Wolf, M. Squillante, P.S.Yu, J.Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. *In WWW*, 2002.