WebCQ – Detecting and Delivering Information Changes on the Web

Ling Liu
Georgia Institute of Technology
College of Computing
Atlanta, GA 30332, USA
lingliu@cc.gatech.edu

Calton Pu
Georgia Institute of Technology
College of Computing
Atlanta, GA 30332, USA
calton@cc.gatech.edu

Wei Tang
Georgia Institute of Technology
College of Computing
Atlanta, GA 30332, USA
wtang@cc.gatech.edu

ABSTRACT

WebCQ is a prototype system for large-scale Web information monitoring and delivery. It makes heavy use of the structure present in hypertext and the concept of continual queries. In this paper we discuss both mechanisms that WebCQ uses to discover and detect changes to the World Wide Web (the Web) pages efficiently, and the methods to notify users of interesting changes with a personalized customization. The WebCQ system consists of four main components: a change detection robot that discovers and detects changes, a proxy cache service that reduces communication traffics to the original information servers, a personalized presentation tool that highlights changes detected by WebCQ sentinels, and a change notification service that delivers fresh information to the right users at the right time. A salient feature of our change detection robot is its ability to support various types of web page sentinels for detecting, presenting, and delivering interesting changes to web pages. This paper describes the WebCQ system with an emphasis on general issues in designing and engineering a large-scale information change monitoring system on the Web.

1. INTRODUCTION

The World Wide Web (the Web), as one of the most popular applications on the Internet, continues to grow at an astounding speed. Not only the size of static web pages increases approximately 15% per month, the number of dynamic pages generated by programs has been growing exponentially. The rapid growth of the Web has affected the ways in which fresh information is delivered and disseminated [1]. Instead of having users tracking when to visit web pages of interest and identifying what and how the page of interest has been changed, the information change monitoring service is becoming increasingly popular, which enables information to be delivered while they are still fresh.

Several tools are available to assist users in tracking of *when* web pages of interests have changed [2]. Most of these tools are on .com domain and offer tracking service either from a centralized server or a client's machine. Netmind [11] and TracerLock [12] are examples of server-based tools. WebWhacker^{TM} [14] is a client-based tool. We can also classify these tools based on the coverage of the service. Some tools offer tracking service on a specific or a constrained set of URLs instead of on any registered URLs; whereas other tools either restrict the number of URLs to be monitored for a user such as WWWFetch [16] or monitor changes of selected topics such as WebSprite [13] and amazon's new book alert.

Up till now most tracking tool development has gone on at companies with little exposure of technical details, especially the efficiency, the scalability, and the tracking quality of such systems. Furthermore, from individual users' perspective, we observe three common problems with these tools. First, with the exception of netmind [11] and AIDE [5], most of the tools only address the problem of when to re-visit a fresh copy of the pages of interest but not the problem of what and how the pages have changed. Second, all these tools handle the *when* problem with a very limited set of capabilities, even the well-known change tracking service. For instance, Netmind can only track change on a selected text region, a registered link or image, a keyword, or the timestamp of the page. The third problem with all these tools is the scalability of their notification service with respect to individual users. Typically, these tools treat each web page tracking request as a unit of notification. Users who register a large number of pages with the tracking service are easily overwhelmed with the large number of frequent email notification messages.

In this paper we present an in-depth description of WebCQ, a prototype of a large-scale Web information monitoring system, which makes use of the structure present in hypertext and the concept of continual queries [8, 9, 8]. WebCQ is designed to discover and detect changes to the Web pages efficiently, and to provide a personalized notification of what and how web pages of interest have been changed. Users' update monitoring requests are modeled as continual queries on the Web. A salient feature of the WebCQ change detection robot is its ability to support various types of web page sentinels for finding and displaying interesting changes to web pages. This paper describes the WebCQ system with an emphasis on the general issues in designing and engineering a large-scale information change monitoring system on the Web. One of the key challenges

CIKM 2000, McLean, VA USA

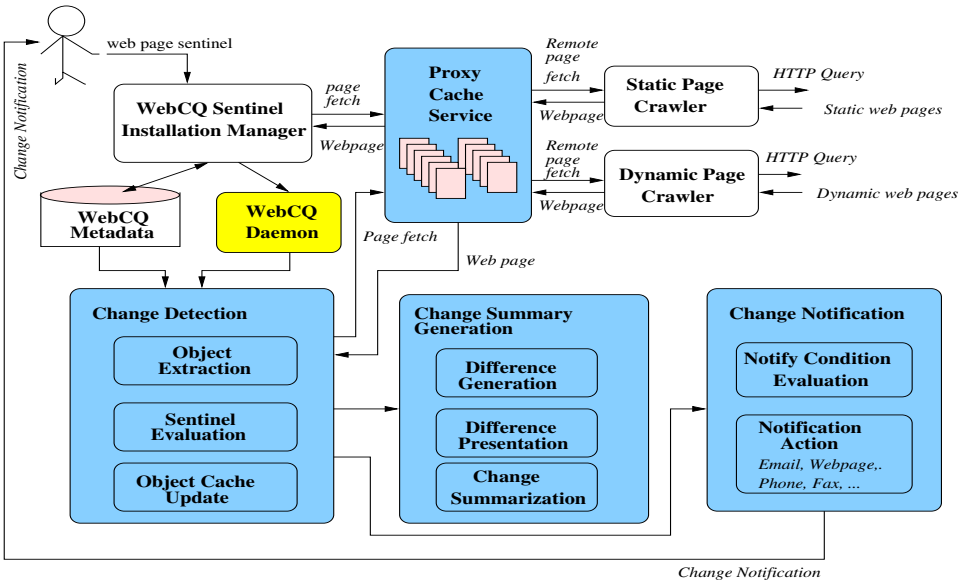

Figure 1: WebCQ System Architecture

for building a Web-based change monitoring system is that it must scale to large number of sources and users, as well as the number of notifications per user.

The rest of the paper proceeds as follows. We first describe the system architecture, the various WebCQ sentinel types, and the installation process of WebCQ sentinels. Then we discuss the main components of WebCQ, including the strategies for detecting changes to arbitrary web pages, the methods for representing changes to web pages, and the WebCQ notification service. We conclude the paper with an overview of related work and a summary.

2. SYSTEM ARCHITECTURE

WebCQ is a server-based change detection and notification system for monitoring changes in arbitrary web pages. The system consists of four main components: a change detection robot that discovers and detects changes to arbitrary web pages, a proxy cache service that reduces the communication traffics to the original information provider on the remote server, a personalized change presentation tool that highlights changes between the web page last seen and the new version of the page, and a centralized notification service that not only notifies users of changes to the web pages of interest but also provides a personalized view of how web pages have changed or what are fresh information. Figure 1 presents a sketch of the system architecture.

Users register their web page monitoring requests with WebCQ using HTML forms. Typically a user enters a URL of interest (say the Internet movie database). The WebCQ installation manager will pass this URL to the proxy server. The proxy server will display the page in the lower frame of the WebCQ window as shown in Figure 4. The user can click on any link of interest on the page until she reaches the exact page that she wishes to monitor. Now the user may select the type of information content she wants the WebCQ to track changes for her from the pull-down menu shown in Figure 4. We call each registered request a web page sentinel. Sentinels in WebCQ are persistent objects. The sentinel installation is submitted to the sentinel installation manager once the user has entered her email address and her preferred monitoring frequency, notification frequency, and notification method. The WebCQ daemon fires the change detection robot periodically to evaluate all installed sentinels. Once interesting changes are detected, the difference generation and summarization module will be fired to generate a detailed change notification report for each sentinel and a change summarization for each client. A notification email will be fired to notify users of the interesting changes.

Users may monitor a web page for any change or specific changes in images, links, words, a chosen phrase, a chosen table, or a chosen list in the page. Furthermore, WebCQ allows users to monitor any fragment in the page specified by a regular expression. Figure 2 shows a list of basic sentinel types supported in WebCQ. Users can register their update monitoring requests using basic sentinels or composite sentinels. A composite sentinel type is a composition of two or more sentinel types. A sentinel is either an instance of a basic sentinel type or an instance of a composite sentinel type. The operators used for sentinel composition are omitted here due to the space restriction. The syntax of a WebCQ sentinel is partially described as follows:

```
<WebCQ Sentinel> ::=
   CREATE Sentinel <sentinel-name> AS
     Sentinel Type <sentinel-type>
     Sentinel Target <sentinel target>
     Sentinel Object <object desc>
     [Trigger Condition <time interval>]
     [Notification Condition <time interval>]
     [Notification Method <method-desc>]
     [Start Condition <time point>]
     Stop Condition <time point>
<sentinel-name>::=<text string>
<sentinel-type>::= <any change> | <all links>
  | <all images> | <all words> | <table> | <list>
  | < phrase> | <regular expression> | keyword
<sentinel target>::=<url>
<object desc>::= <text string>
<time interval>::=<integer> {<minute>
              | <hour> | <day> | <week>}
```

```
<method-desc>::= <email-address> | <fax>
    | <personalized web-bulletin> | <phone>
    | <email-address> <personalized web-bulletin>
<time point>::=<month>``-"<day>``-"<year>``-"
              <hour>``-"<min>``-"<TimeZone>
```

The trigger condition of a sentinel specifies how frequent the change detection robot should be fired to check out if any interesting changes have happened. The notification condition is designed to support the situations where the desired notification condition is different from the trigger condition. It can be the same as the trigger condition or n times of the trigger frequency. For instance, one may want WebCQ to track changes daily but receive the notification report weekly. The optional clause `Notification Method` allows users to select a notification means from the set of methods supported by the system. In the first prototype of WebCQ, we only support two notification methods: email and personalized web bulletin.

Consider a WebCQ sentinel "*tracking 'Movie of the day' changes from the Internet Movie Database Web site in November of 2000*". The installation through WebCQ GUI is shown in Figure 4. This sentinel will be captured in the following WebCQ sentinel expression:

```
CREATE Sentinel movie_of_the_day AS
  Sentinel Name ``IMDB movie of the day"
  Sentinel Type regular expression
  Sentinel Target http://www.imdb.com
  Sentinel Object ``IMDB Movie of the Day.*?more.*?\)''
  Trigger Condition 1 day
  Notification Condition 1 day
  Notification Method email
  Start Condition November 1, 2000
  Stop Condition November 30, 2000
```

Another important feature of WebCQ architecture is its server-based proxy cache service. The goal of introducing proxy cache service is to reduce the cost of repeated connections to the remote information servers. The proxy service works as follows: Any remote page request will be sent to the nearest proxy server first. The proxy server only forwards the remote fetch request (such as HTTP Get or HTTP Post) to the corresponding information server if a copy of the requested page is not found in the local proxy cache or any of the relevant proxies. The static page crawler is used to fetch static pages, while the dynamic page crawler is designed for handling the remote fetch of dynamic pages, including search interface extraction and HTTP query composition. An obvious advantage of the proxy service is to enable the WebCQ system to share the cost of remote page fetch among all monitoring requests over the same web page. By reducing the frequency and the overhead of network connections to remote data servers, the unnecessary network connection cost is avoided, and the scalability of the system is enhanced.

3. CHANGE DETECTION ROBOT

The change detection robot consists of three modules: object extraction, sentinel evaluation, and object cache update. For each sentinel, the change detection robot first fetches the web page being monitored by the sentinel through a page fetch call to the proxy server (recall Figure 1). Then it performs the following three tasks:

Object Extraction – it locates and extracts the objects being monitored from the page. In WebCQ, we provide two different methods to extract web page contents of interest. They are the HTML based data extraction and the regular-expression based data extraction as shown in Figure 3.

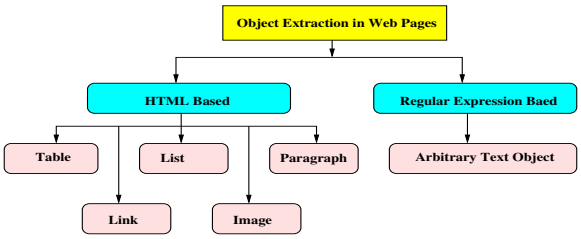

Figure 3: Object Extraction in WebCQ

- For HTML-based data extraction, five different logical structures are extracted, including Table, List, Paragraph, Link, and Image. A small token-based HTML parser is used to identify HTML tags and extract logical structures of interest from a given web page.
- For regular-expression based data extraction, a regular expression package is used to identify and extract any text string specified by an regular expression. Figure 4 shows an example of installing a regular-expression sentinel, which tracks changes to the IMDb Movie of the Day information on the front page of Internet Movie Datanase web site. The small pop-up window shows the result of data extraction using the user-defined regular expression "`IMDb Movie of the Day.*?more.*?\)`" over the IMDb front page. This regular expression says to extract the text starting from "IMDb movie of the Day" till "more)", ignoring any potential tags (text) in between the word "more" and the character ")". It is well known that different regular expressions have different evaluation cost. In the current prototype of WebCQ we do not support the transformation of a user-defined regular expression into an efficient one. This is one of the main reasons we choose to use HTML-based data extraction algorithms to extract those content objects that are well defined by HTML tags.

Sentinel Evaluation – it compares the object extracted from the current copy of the page with the previous cached copy of the object. The difference between the two copies of the object is computed.

Object Cache Update – When the difference between two copies of the object is found, and it matches the specified sentinel condition, then both the old copy and new copy of the object will be passed to the difference generation and summarization module, and the old cache copy of the object is updated by the new copy.

For sentinels of type *any change*, *link change*, *image change*, or *word change*, the method for locating and extracting the objects being monitored is straightforward. For detecting any change to a page, we use the HTTP `HEAD` request to obtain the last-modified timestamp for all static pages. Occasionally, we see that a page whose last modification timestamp changes while the body does not is flagged as changed. In such cases, using just a `HEAD` request without actual checksum evaluation will not be good enough. Several methods

sentinel types	Synopsis	Sentinel Monitoring Method
Content Update	Any update on the page object	watch any change to the modification timestamp of the file
Content Insertion	Any new content to the page object	when file size increases
Content Deletion	Any content deletion on the page object	when file size decreases
Link Change	Any change to links of the page	when new links added or old links removed
Image Change	Any change to images of the page	when new images added or old images removed
Words Change	Any change to words of the page	when new words added or existing words removed
Phrase Update	Any change to the selected text phrase	identify and extract the selected phrase; detect any change in the phrase
Phrase Deletion	Removal of the selected text phrase	identify the surrounding text of the selected phrase; detect if the phrase disappears
Table Change	Any change to the content of the table	identify and extract the selected table; detect any change to the table
List Change	Any change to the content of the list	identify and extract the selected list; detect any change to the list
Arbitrary text Change	Any change to the text fragment specified by a regular expression	identify and detect any change ; in the selected fragment
Keywords	The specified keywords appear in or disappear from the page	watch if the selected keywords disappear or appear

Figure 2: A list of basic sentinel types in WebCQ

can be used in addition to the last modified timestamp to double-check if a page has changed. For example, once a `HEAD` request indicates a change, a simple solution is to use the file size in addition to the modification timestamp. This method will avoid those cases where content of a page did not change though the timestamp is changed. However, it will miss to capture the case where a page has changed but the amount of text deleted equals to the amount of changes inserted, thus the file size of the new copy of the page remains the same as the file size for the old copy. A better approach is to use `HTTP GET` to retrieve the page and compute the difference. Obviously, generating the difference will give a more accurate detection result but it is also more expensive. However, when no last modification timestamp is provided such as the case of dynamic pages or a prior attempt that no timestamp is given, then HTTP `GET` is used to retrieve the body of the page via the proxy service and a simple checksum is computed. To detect changes to hypertext links or images in a page, the hypertext reference tag (`<A href=...>`) is used to identify all hypertext links in the page; and the image tag (`<IMG src=...>`) is used to identify all images in the page.

For sentinels of type *phrase change*, *table change*, *list change*, and *regular expression monitoring*, the task of identifying and extracting the correct object being monitored in a page is more sophisticated. We introduce the concept of context bounding box. A content bounding box for an object being monitored in a web page is defined by the surrounding context of the object. This context bounding box is used to identify and extract the object in the subsequent copies of the page. There are several ways to define the bounding box context for an object. For example, we may define the bounding box context of an object `O` by the ten words before the boundary of the object `O` and the ten words after the boundary of the object `O`. The context bounding box approach assumes that the selected text surrounding the object being monitored is relatively stable. Our experience has shown that for static web pages, using words instead of tags to define the bounding box context of an object often gives us more accurate results. However, for dynamic pages, using tags instead of words tends to provide results of higher accuracy. For a table or a list object, users may select a sentence or a keyword within the table or the list as the unique identifying text of the table or list. This mechanism also applies to phrase objects. Whenever users provide such identifying text, instead of using the bounding box context method, WebCQ will then use such identifying text to identify the object.

4. DIFFERENCE GENERATION AND SUMMARIZATION

Most of the tools that monitor changes to web pages have the capability of notifying users that something on a page has changed and here is the link to the new copy of the page. But few are able to include what and how the page has changed in the notification report. When the number of pages that a user is interested in tracking changes is large and the changes on the page are subtle, it is likely that the user will not know what has changed simply by viewing the new copy of the page. Therefore, computing and showing the difference to a web page is critical from the usability perspective. In this section we address three issues related to representing and viewing changes: difference generation, difference representation, and change summarization.

4.1 Difference Generation

Computing changes to a web page consists of two tasks: obtaining both the old and the new version of the page and comparing the two versions to display the difference.

The first task is related to the archival of web pages. In WebCQ, we deliberately choose not to archive every page we access. Instead, for every web page accessed by WebCQ we at most keep one past copy which will be used to compare with the current version of the page and compute how changes have been made. The object cache update module

Figure 4: Object extraction during a sentinel installation

is in charge of maintaining and refreshing the object cache used by the sentinel evaluation module and the difference generation module (recall Figure 1).

The second task is related to the difference generation from two versions of a web page. One way to compute the difference between two versions of an HTML page is to use *HTMLDiff* developed by AT&T [5, 4]. Similar to the UNIX *diff* utility [6], *HTMLDiff* takes two versions of a page as an input and produces a new and merged HTML document that highlights the differences between the two versions by flagging the inserted text with bold or colored face and deleted text with a horizontal line cross over. Changes to existing text are treated as deletions followed by insertions. As pointed out by Fred Douglis and his colleagues [5], every invocation of the *HTMLdiff* may potentially consume significant computation and memory resources. Such resource overheads will restrain the number of difference operations a server can perform at a time, limiting the scalability of the system. In WebCQ, most of the sentinels are targeted at tracking changes to a page fragment rather than the entire page. A page fragment can be a specific HTML object (such as phrase, list, and table), an arbitrary text fragment, or a specific component of the page (such as links, images, and words). In such cases, a simplified difference generation algorithm should be used to reduce the overhead of the general *HTMLDiff*.

There are three basic mechanisms for computing the difference between two versions of a page fragment.

- Difference is flagged only when content (raw text in between a pair of tags) changes.
- Difference is flagged when either content or structure changes.
- Different flags are used for both content changes and structure changes.

One problem with the second and the third method is that any small and trivial change to formatting, such as adding or removing a line-break tag `<br>` in the middle of a sentence or adding a paragraph begin tag `<P>` in between two sentences, will be flagged. After careful consideration, the first mechanism is used, in the current prototype of WebCQ, for computing the difference from two versions of an HTML page fragment.

4.2 Difference Presentation

There are three popular ways to present the difference between two web documents [5, 11, 10]. The first approach is to *merge the two documents* by summarizing all of the common, new, and deleted materials in one document, as it is done in *HTMLDiff* [5] and UNIX *diff*. The advantage of this approach is to display the common parts of the two pages just once. The disadvantage of this approach is also obvious. By incorporating two pages into one may cause the creation of a syntactically or semantically incorrect HTML, making the merged page difficult to read especially when the amount of changes is not small.

The second approach is to *display only the differences* but omit the common parts of the two pages. This approach is especially beneficial for the documents of large size and for the case where the two copies of a document have a lot in common. An obvious drawback of this approach is the lost of surrounding common context, resulting in sometimes confusing presentation of the difference.

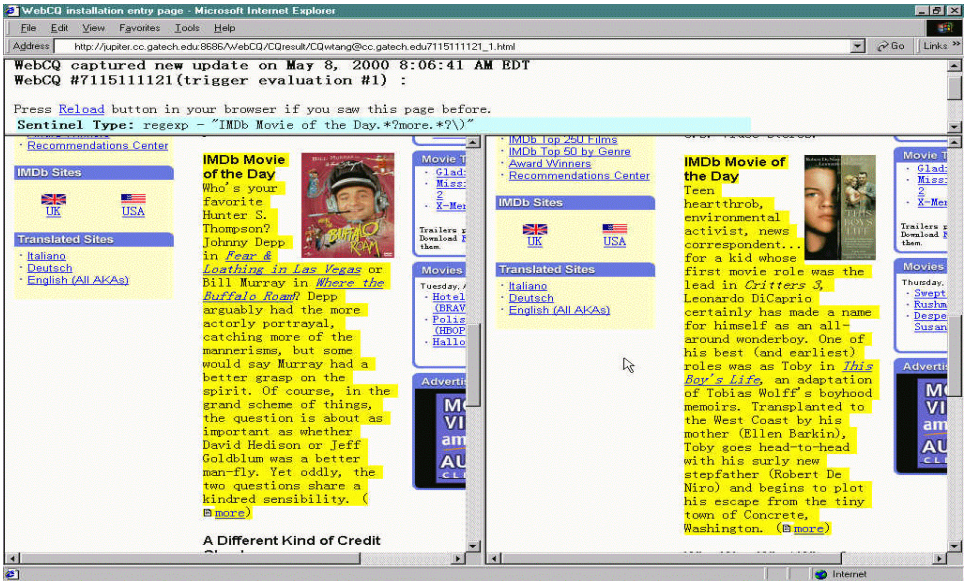

Figure 5: A difference presentation for a regular expression sentinel

The third approach is *side-by-side presentation* of the differences between two documents. A value-added feature to this approach is to highlight the fragments that are changed in both the old and the new copy of the page. This approach is considered the most pleasing way of displaying the differences between two documents. However, it is not an optimized approach to present the differences for very large documents, especially when the amount of differences is large as well.

In WebCQ we use a hybrid approach that utilizes all three of the presentation schemes in selected combinations. For instance, we use the *Side-by-Side* for sentinels of phrase or regular expression type, and the combination of *Side-by-Side* with *Only Difference* to display differences for all-links, all-words, table, and list sentinels. We use the merge-two-documents approach for sentinels of image type. Figure 5 shows an example where a regular expression sentinel is used to track changes in the text region specified by the regular expression given in Figure 4. The difference presentation module displays the new web page with the new text highlighted on the right window and the old copy of the web page with the deleted text highlighted in a different color on the left window.

4.3 Change Summarization

Our experience with the change monitoring systems [9, 7] shows that one of the important usage improvement for change tracking and notification services on the Web is to provide at least one change summarization for each user, reporting the status of all sentinels installed by the user. Users will be given a choice of receiving a summarization notification for all sentinels she has registered with the system or one notification per sentinel. A change summarization report contains a list of summarization records. Each of such records presents a brief summary of the recent sentinel evaluation result, including

- the URL of the web page being monitored,
- the type of change, such as insertion, deletion, update to the page, a new page, or a deleted page,
- the sentinel type,
- the last modification timestamp, showing when the page has changed,
- the last evaluation timestamp, displaying when the change is detected,
- the last notification timestamp, showing when the last notification was sent out,
- the timestamp when the page was last seen,
- the installation timestamp, showing when the sentinel was registered initially,
- the termination timestamp, showing when the monitoring sentinel will be terminated,
- the number of notifications since the installation,
- the number of evaluations since the installation, and
- the hypertext link to the detailed difference presentations of the sentinel.

Figure 6 shows an example change summarization.

5. CHANGE NOTIFICATION SERVICE

Change notification is a software facility that provides mechanisms for notification of information changes or occurrence of events. The common characteristics of the notification services are the suite of mechanisms that identify when a notification should be sent, how a notification is sent, and what should be included in a notification.

When to Notify – Eager notification, periodic notification, and on-demand notification are the three mechanisms we have investigated.

- *Eager notification* advocates that any interesting change once detected should be delivered to the client immediately. An obvious advantage of eager notification is

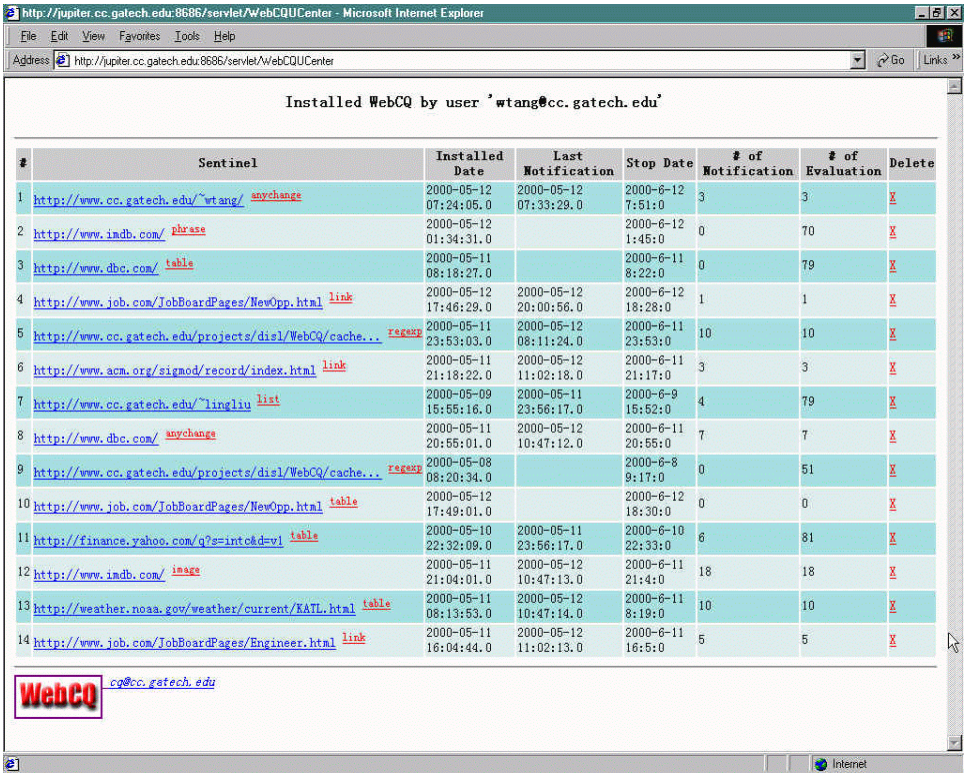

Figure 6: A change summarization example

to guarantee the minimum latency from the time when changes were detected to the time when the notification was sent out. Periodic notification and on-demand notification, on the contrary, are based on a deferred notification scheme.

- *Periodic notification* for changes have the advantage of ensuring that the user will have relatively up-to-date information about the pages being tracked, but require Internet connectivity at the time specific and for a duration in proportion to the number of pages to be monitored.
- *On-demand notification* promotes the idea of having users fire the notification request when they want to review the change tracking results. On-demand notification will not be effective if the number of web pages being monitored is large.

In WebCQ, the notification service runs on the server side, thus on-demand approach is not considered. We choose the periodic notification over the eager notification because immediate notification may not scale well when a user registers a large number of sentinels for tracking changes to the web pages. The periodic interval is defined by the notification condition given by the user at the sentinel installation time.

How to Notify – The how to notify problem involved three issues: how to initiate a notification, how carry out a notification, and how to present changes to the users without making them overwhelmed with notification.

- In WebCQ, the notification can be provided by either a server-initiated push delivery or a client initiated pull delivery.
- Both email and web pages are effective mechanisms for server-pushed delivery. However, for client-pulled delivery, WebCQ only provides web pages with a search interface to allow users to query and view the change reports from anywhere in an ad-hoc style.

What to Notify – We observed that many users prefer to have the WebCQ tracking changes to the web pages of interest and notify them of the changes with some summarization in addition to individual change report per sentinel. In WebCQ, we provide support for the following four types of notification reports:

- Detailed sentinel evaluation report, displaying the differences side by side or in a merged document;
- A change summary report per user, displaying the list of sentinels installed by a user, each with a brief summary of the most recent change detection status. This approach is especially when the number of web pages being monitored per use is not small.
- A change summary report per user grouped by topic of the web pages being monitored. This approach scales much better than the previous one when the number of web pages monitored per user is getting large.
- A dedicated web notification side where a conditional query interface is provided to allow users to search and find the change summary reports or the detailed change reports which match the given condition. This approach is particularly useful for busy users or users with Wireless devices and intermittent Internet connectivity.

6. RELATED WORK

There has been considerable research done on data update monitoring in databases. Powerful database techniques such as active databases and materialized views have been studied extensively. These techniques have been proposed primarily for "data-centric" environments, where data is well organized and centrally controlled in a database with a close-world assumption. When applied to an open information universe as the Internet, these assumptions no longer hold, and some of the techniques do not easily extend to scale up to the distributed interoperable environment.

Comparing with the research in active databases [15], the WebCQ system differs in three ways: First, the WebCQ system targets at monitoring and tracking changes to arbitrary web pages. Second, WebCQ monitors data provided by the content providers on remote servers, and WebCQ monitoring and tracking service requires neither the control over the data it monitors nor the structural information about the data it is tracking. Whereas active databases can only monitor structured data that reside in a database. Third, WebCQ provides efficient and scalable proxy service as well as grouping techniques for trigger processing, and it can handle large numbers of concurrently running sentinels on a large number of web pages.

There are several systems developed towards monitoring web source changes. One type of systems is the extension of Web search software by monitoring URL changes and notifying users of the URLs that have changed. A representative system is the NetMind, formerly known as URL-minder, which provides keyword-based and text-based change detection and notification service over Web pages. Compared with NetMind WebCQ provides a richer set of sentinel types for monitoring changes to arbitrary web pages and offers variations of personalized difference presentations and change summarizations. Furthermore, WebCQ allows users to specify a trigger condition for detecting changes in addition to a notification condition for delivering changes. The second type of monitoring systems is the application-specific change notification systems such as E*Trade alert facility, Amazon.com new book notification service. The most representative one of this type is the Stanford news service SIFT [17], which filters and notifies Internet news of interest through user preferences on news items and news groups. The problem with this type of systems is that they are tailored to a particular (type of) data source or application, which consequently do not have the generality and extensibility as WebCQ. The third type of projects is the change detection over wrapped Web pages, such as the $C3$ [3] project at Stanford and OpenCQ [9]. In contrast, WebCQ can monitor arbitrary web pages and track and deliver interesting changes to the users at the right time.

7. CONCLUSION

We have presented WebCQ, a prototype of a large-scale Web information monitoring system, with an emphasis on general issues in designing and engineering a large-scale information change monitoring system on the Web. Features of WebCQ include the capabilities for monitoring and tracking various types of changes to static and dynamic web page changes, personalized delivery of page change notifications, and personalized summarization of web page changes.

Our work on the WebCQ system continues. Several issues are being explored, including experiments to evaluate the performance of alternative architectures and algorithms, and the performance improvements by incorporating research results in indexing techniques into the WebCQ change detection and change notification. We are also investigating the scalability enhancement of the WebCQ system along three dimensions: replication, distribution, and caching [7].

8. ACKNOWLEDGEMENTS

The authors are partially supported by NSF CCR-9988452, DARPA/ITO Information Expedition, Ubiquitous Computing, Quorum and PCES program, and Yamacraw mission, State of Georgia.

9. REFERENCES

[1] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD Conference*, 1997.

[2] BotSpot. http://bots.internet.com. 1999.

[3] S. Chawathe, S. Abiteboul, and J. Widom. Managing and querying changes in semi-structured data. In *Proceedings of ACM SIGMOD Conference*, 1997.

[4] F. Douglis, T. Ball, Y.Chen, and E. Koutsofios. WebGuide: Querying and Navigating Changes in Web Repositories. *Proceedings of 1996 USENIX.*

[5] F. Douglis, T. Ball, Y.Chen, and E. Koutsofios. The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web. *World Wide Web, 1 (1), pp.27-44*, January 1998.

[6] J.W.Hunt and M.D.Mcllroy. An algorithm for efficient file comparison. *Technical Report Computer Science TR#41, Bell Laboratories, Murray Hill, N.J.*, 1995.

[7] L. Liu and C. Pu. Issues in web information change monitoring and notification systems. *Technical Report, College of Computing, Georgia Tech*, May, 2000.

[8] L. Liu, C. Pu, R. Barga, and T. Zhou. Differential evaluation of continual queries. In *IEEE Proceedings of the 16th ICDCS*, May 27-30 1996.

[9] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 1999. Special Issue on Web Technology.

[10] Mortice Kern Systems (MKS). Web integrity. http://www.mks.com/solutions/ebms. 1997.

[11] NetMind. http://www.netmind.com.

[12] TracerLock. http://www.peacefire.org/tracerlock.

[13] WebSprite. http:/www.websprite.com.

[14] Webwhacker. http://www.webwhacker.com.

[15] J. Widom and S. Ceri. *Active Datanase Systems.* Morgan Kaufmann, 1996.

[16] WWWFtech. http://www.wwwftech.com.

[17] T. W. Yan and H. Garcia-Molina. SIFT - a tool for wide area information dissemination. In *Proceedings of the 1995 USENIX*, pp177–186.