CHANGE DETECTION AND MAINTENANCE OF AN XML WEB WAREHOUSE

Ching-Ming Chao
Department of Computer and Information Science, Soochow University, Taipei, Taiwan, R.O.C.

Keywords: Web warehouse, XML, Change detection, Warehouse maintenance, Mobile agent

Abstract: The World Wide Web is a popular broadcast medium that contains a huge amount of information. The web warehouse is an efficient and effective means to facilitate utilization of information on the Web. XML has become the new standard for semi-structured data exchange over the Web. In this paper, therefore, we study the XML web warehouse and propose an approach to the problems of change detection and warehouse maintenance in an XML web warehouse system. This paper has three major contributions. First, we propose an object-oriented data model for XML web pages in the web warehouse as well as system architecture for change detection and warehouse maintenance. Second, we propose a change detection method based on mobile agent technology to actively detect changes of data sources of the web warehouse. Third, we propose an incremental and deferred maintenance method to maintain XML web pages in the web warehouse. We compared our approach with a rewriting approach to storage and maintenance of the XML web warehouse by experiments. Performance evaluation shows that our approach is more efficient than the rewriting approach in terms of the response time and storage space of the web warehouse.

1 INTRODUCTION

The World Wide Web is a popular broadcast medium that contains a huge amount of information. Information on the Web is important not only to individual users but also to business organizations, especially for decision-making purposes. Therefore, how to efficiently and effectively utilize web information is an important issue. Recently, the concept of web warehousing was proposed to address this issue (Ng, 1998; Xyleme, 2001). The idea of web warehousing is to build a web warehouse, which materializes and manages useful information from the Web, so as to facilitate utilization of web information. A web warehouse is a repository of web pages extracted from remote web sites. It has specific data sources and is built with a specific theme or purpose that is of interest to a user community. Because XML has become the new standard for semi-structured data exchange over the Web, we study in this paper the web warehouse of XML data, which will be called the XML web warehouse thereafter.

If changes occur at data sources, a web warehouse should be aware of those changes that may affect the data in the web warehouse. How does a web warehouse detect such changes is called the change detection problem. In the traditional data warehouse environment, data sources are cooperated with the data warehouse and hence will send messages about their changes to the data warehouse (Labio, 1995). However, data sources of a web warehouse are autonomous web sites and will not actively notify the web warehouse of their changes. Previous work on change detection of web data queries data sources for their changes (Bhowmick, 2000; Lim, 2001), which requires more message transmission and is inefficient. Therefore, how to efficiently detect changes of data sources is an important problem to a web warehouse.

If changes occur at data sources, a web warehouse may need to be maintained. Deciding whether and how to maintain a web warehouse is called the warehouse maintenance problem. The problem of warehouse maintenance is not a brand new problem and has been studied for traditional data warehouses. However, most of the previous work on this problem was focused on the relational data model and the traditional data warehouse environment (Agrawal, 1997; Zhuge, 1995; Zhuge, 1996). A web warehouse can be differentiated from a traditional data warehouse in terms of its data model and operating environment, and hence requires a different maintenance method. Therefore, how to efficiently maintain a web warehouse is also an important problem to

Chao C. (2005).
CHANGE DETECTION AND MAINTENANCE OF AN XML WEB WAREHOUSE.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 52-59
DOI: 10.5220/0002555700520059

study.

In this paper, we propose an approach to the problems of change detection and warehouse maintenance for an XML web warehouse system. First, we propose an object-oriented data model for XML web pages in the web warehouse as well as system architecture for change detection and warehouse maintenance. Then, we propose a change detection method based on mobile agent technology to actively detect changes of data sources of the web warehouse. Finally, we propose an incremental and deferred maintenance method to maintain XML web pages in the web warehouse. We have implemented an experimental prototype system for change detection and maintenance of an XML web warehouse and have compared our approach with a rewriting approach by experiments. Performance evaluation shows that our approach is efficient in terms of the response time and storage space of the web warehouse.

The remainder of this paper is organized as follows. In Section 2 we illustrate the data model of the web warehouse and the system architecture for change detection and warehouse maintenance. In Section 3 we present the change detection method and algorithm. In Section 4 we present the warehouse maintenance method and algorithm. Section 5 illustrates the experimental results and states our observations from the experimental results. Section 6 concludes this paper and gives some directions for future research.

2 DATA MODEL AND SYSTEM ARCHITECTURE

2.1 Data Model

Our web warehouse stores XML data from the Web. Hence, we propose a data model, called the XML Web Warehouse Data Model (XWWDM), for XML web pages in the web warehouse. Due to the hierarchical structure of an XML web page, we follow the Document Object Model (Apparao, 1998) to decompose an XML web page into a tree structure. Besides, the design of the XWWDM model is based on the OEM-like model (Chawathe, 1999) and considers the characteristics of a web warehouse. First, a web warehouse is like a data warehouse in that it can store historical data. Therefore, the data model includes version information to keep track of the change of data. Second, data in a web warehouse are sourced from remote web sites. Therefore, the data model includes source information to identify the source of data. The XWWDM model is an object-oriented model whose class definition is shown in Figure 1.

class *XML_Page* {*root: XML_Node, version: Version_Info, source: Source_Info*};
class *XML_Node* {*content: Node_Content, version: Version_Info*};
class *Node_Content* {*label: string, value: string, p-node: XML_Node, child#: integer, s-action: char*};
class *Version_Info* {*version#: integer, update-time: time*};
class *Source_Info* {*url: string, title: string*};
class *Update* {*content: Update_Content, source: Source_Info*};
class *Update_Content* {*label: string, value: string, p-node: XML_Node, detect-time: time, action: char*};

Figure 1: The class definition of the XWWDM model

An XML web page is represented as an object of the class *XML_Page*, which has three attributes *root*, *version*, and *source*. The attribute *root* records the root node of the tree structure of the web page. The attributes *version* and *source* record the newest version information and source information of the web page, respectively. Each node of a web page is represented as an object of the class *XML_Node*, which has two attributes *content* and *version*. The attribute *content* records the content, position, and source action of a node. The attribute *version* records the version information of a node. The class *Node_Content* has five attributes *label*, *value*, *p-node*, *child#*, and *s-action*. The attributes *label* and *value* record the tag label and data content of a node, respectively. The attributes *p-node* and *child#* record the parent node and child number under its parent, respectively. The attribute *s-action* records the source action causing the creation of a node, whose value is *I* (for insertion), *D* (for deletion), or *M* (for modification). The class *Version_Info* has two attributes *version#* and *update-time*, which record the version number and time of last update, respectively. The class *Source_Info* has two attributes *url* and *title*, which record the URL and title of the source web page, respectively.

We adopt a change-centric approach to storage of all versions of an XML web page. Only the first version is completely stored. For subsequent versions, only deltas are stored. As shown in Figure 2, all frames represent the same web page, in which each frame represents a specific version at time Ti. The first frame represents the first version, in which all nodes of a web page are stored. Other frames represent subsequent versions, in which only nodes that are changed are stored. The number and letter drawn by a node are the child number and source

action of the node. Different versions of a node have the same parent node and child number but different version numbers. The first version of a node has a value *I* in the attribute *s-action*. A value *D* in the last version of a node indicates that this node has been deleted.

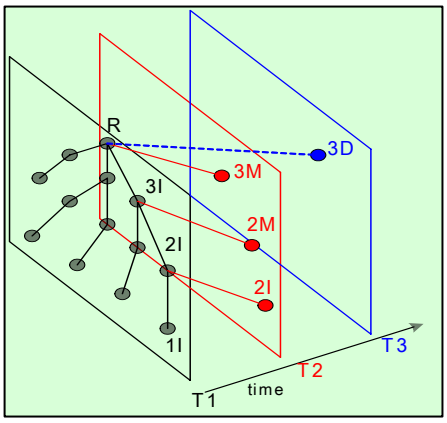


Figure 2: All versions of an XML web page

A source update is represented as an object of the class *Update*, which has two attributes *content* and *source*. The attribute *content* records a variety of information about the update. The attribute *source* records the identification information of the web page in which the update occurs. The class *Update_Content* has five attributes *label*, *value*, *p-node*, *detect-time*, and *action*. The attributes *label*, *value*, and *p-node* have similar meaning as in the class *Node_Content*. The attributes *detect-time* and *action* record the detection time and type of the update, respectively. The update type can be *I* (for insertion), *D* (for deletion) or *M* (for modification). Source updates are detected by mobile agents and transmitted from data sources to the web warehouse, and are used for the purpose of maintaining the web warehouse.

2.2 System Architecture

The system architecture for change detection and maintenance of the XML web warehouse is shown in Figure 3, which is divided into three layers: the storage layer, the system kernel layer, and the mobile agent layer. The primary components and their functions in each of the three layers are presented below.

2.2.1 Storage Layer

This layer is where the web warehouse is stored. The web warehouse uses the XWWDM model to store specific and historical XML data from the Web. The web warehouse serves as a knowledge base of decision support systems, providing data for web mining and on-line analytical processing.

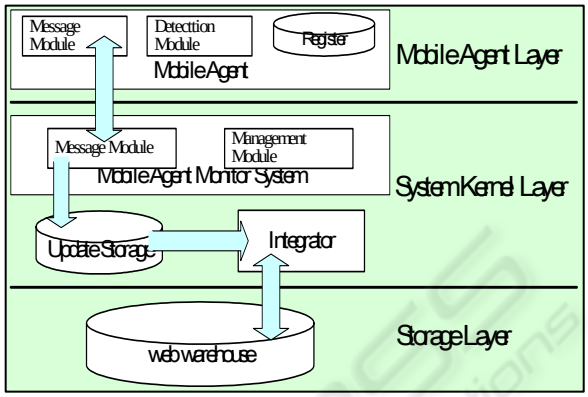


Figure 3: System architecture

2.2.2 System Kernel Layer

This layer is the kernel of the part of the web warehouse system for change detection and warehouse maintenance. It includes the Mobile Agent Monitor System, the Integrator, and the Update Storage. The Mobile Agent Monitor System further consists of two modules: the Management Module and the Message Module. The Management Module is responsible for producing, dispatching, and tracing mobile agents. The Message Module is responsible for sending messages to and receiving messages from mobile agents as well as placing received source updates into the Update Storage. The Integrator is responsible for maintaining the web warehouse according to source updates stored in the Update Storage. The Update Storage is a temporary storage of source updates sent by mobile agents and provides the Integrator with required update information for warehouse maintenance.

2.2.3 Mobile Agent Layer

This layer includes mobile agents that are operating in data sources of the web warehouse and are responsible for detecting and reporting changes of data sources. Each mobile agent includes three components: the Detection Module, the Message Module, and the Register. The Detection Module is responsible for detecting changes of the data source. The Message Module is responsible for sending messages to and receiving messages from the Mobile Agent Monitor System. The Register is the storage for the mobile agent to store the last states of web pages and detected source updates.

3 CHANGE DETECTION

In this section, we address the issue of change detection of data sources of an XML web warehouse. Because a web warehouse system is operating in the Internet environment and its data sources are remote and autonomous web sites, it is better for it to actively detect changes of data sources so as to speed up data refreshment and reduce network traffic. Mobile agents have the feature of cross platform as well as the abilities of active detection and rapid reporting. Therefore, we propose a change detection method based on mobile agent technology in order for the web warehouse system to fully and efficiently obtain changes of data sources. The procedure of our change detection method consists of the following four steps:

1. The Mobile Agent Monitor System produces and dispatches a mobile agent to each of the data sources.
2. When a mobile agent arrives at a data source, it first sends source information of the data source back to the Mobile Agent Monitor System. This source information allows the Mobile Agent Monitor System to trace the mobile agent.
3. Thereafter a mobile agent will actively and periodically detect changes of a data source using the Change Detection Algorithm. If changes are detected, the mobile agent sends detected source updates to the Mobile Agent Monitor System.
4. When the Mobile Agent Monitor System receives source updates from a data source, it stores these updates in the Update Storage for the purpose of warehouse maintenance.

The Detection Module of a mobile agent uses the Change Detection Algorithm, the CD algorithm for short, shown in Figure 4 to detect the difference between the current state and the last detected state of an XML web page and send a collection of updates to the Mobile Agent Monitor System through the Message Module. Here we first give a brief overview of the CD algorithm. First, the algorithm fetches an XML web page from the web site and finds the last recorded state of the web page from the Register. Both states of the web page are decomposed into tree structures using an XML parser. Then, these two tree structures are compared level-by-level in a top-down fashion to detect their difference. During this process, a collection of updates representing the difference is accumulated. Finally, the current state of the web page is recorded in the Register and the collection of updates is sent to the Mobile Agent Monitor System. Now we explain the CD algorithm in detail. The algorithm executes the following steps in sequence:

Algorithm Change Detection Algorithm
- Fetch an XML web page WP.
- Find the last recorded state LP of WP from the Register.
- Initialize a collection of updates UC to be empty.
- Use an XML parser to decompose WP and LP into tree structures.
- **For** each level L of WP and LP **do**

① **For** each node WPN of level L of WP **do**
Find the node LPN of level L of LP such that
LPN.content.label = WPN.content.label
If LPN is found **then** Mark WPN and LPN.
If WPN.*content.value* ≠ LPN.*content.value* **then**
⇒ Create an update object U whose attributes are as follows:
content.label ← the tag label of WPN
content.value ← the data content of WPN
content.p-node ← the parent node of WPN
content.detect-time ← current system time
content.action ← 'M'
source.url ← the URL of WP
source.title ← the title of WP
⇒ Put U into UC.

② **For** each node WPN of level L of WP **do**
If WPN is not marked **then**
⇒ Create an update object U whose attributes are as follows:
content.label ← the tag label of WPN
content.value ← the content of WPN
content.p-node ← the parent node of WPN
content.detect-time ← current system time
content.action ← 'I'
source.url ← the URL of WP
source.title ← the title of WP
⇒ Put U into UC.

③ **For** each node LPN of level L of LP **do**
If LPN is not marked **then**
⇒ Create an update object U whose attributes are as follows:
content.label ← the tag label of LPN
content.value ← the data content of LPN
content.p-node ← the parent node of LPN
content.detect-time ← current system time
content.action ← 'D'
source.url ← the URL of LP
source.title ← the title of LP
⇒ Put U into UC.

- Record the state of WP in the Register.
- Send UC to the Mobile Agent Monitor System.

End Algorithm

Figure 4: The change detection algorithm

1. Fetch an XML web page WP from the web site. WP is the current state of the web page whose difference from its last recorded state is to be detected.
2. Find the last recorded state LP of WP from the

Register. Because the detected web page is assumed to have been stored in the web warehouse, it must have the last recorded state.
3. Initialize a collection of updates UC to be empty. UC represents the difference between WP and LP.
4. Decompose WP and LP into tree structures using an XML parser.
5. Compare each level of the tree structures of WP and LP in a top-down fashion in a loop. For each level L of WP and LP, there are three inner loops. In the first inner loop, for each node WPN of level L of WP, find the node LPN of level L of LP such that the tag label of LPN is equal to that of WPN. If such a node is found, mark WPN and LPN to indicate that this node exists in both states of the web page. If the data content of WPN is not equal to that of LPN, which indicates that WPN has been modified, create a modification update and put it into UC. In the second inner loop, for each node WPN of level L of WP, check if it is marked. If it is not marked, which indicates that this node has been inserted to the current state, create an insertion update and put it into UC. In the third inner loop, for each node LPN of level L of LP, check if it is marked. If it is not marked, which indicates that this node has been deleted from the last state, create a deletion update and put it into UC.
6. Record the state of WP in the Register. This state will become the last recorded state next time this web page is detected.
7. Send the accumulated collection of updates UC to the Mobile Agent Monitor System.

4 WAREHOUSE MAINTENANCE

In this section, we address the issue of maintaining an XML web warehouse in response to changes of data sources. The way to warehouse maintenance somewhat depends on the way to storage of the warehouse as well as the way to change detection of data sources. Our web warehouse adopts a change-centric approach to storage of historical data of web pages. Our change detection method utilizes mobile agents to send actively and periodically source updates back to the web warehouse. Accordingly, we propose an incremental and deferred maintenance method to maintain XML web pages in the web warehouse. Incremental maintenance of a web page means that only parts of the web page are updated according to its source updates. It is generally more efficient in terms of time and space than rewriting the whole web page. Deferred maintenance of a web page means that the web page is maintained only when it is accessed. It is generally more time efficient than immediate maintenance in which a web page is maintained immediately after a source update occurs. Besides, the user is still able to access reasonably up-to-date web pages in a relatively short time. The procedure of our warehouse maintenance method consists of the following four steps:

1. When the web warehouse receives a request from a user to access a web page, it notifies the Integrator to maintain the web page.
2. The Integrator checks the Update Storage to see if there are source updates relevant to the web page. The relevant source updates are those that have the same source location information (i.e., URL) as the web page. All updates made to the source web page of this web page are considered as relevant source updates.
3. If the web page has relevant source updates, which indicates that it is out of date, the Integrator maintains it using the Web Warehouse Maintenance Algorithm.
4. The web warehouse returns the requested web page to the user.

The Integrator uses the Web Warehouse Maintenance Algorithm, the WWM algorithm for short, shown in Figure 5 to maintain an XML web page in the web warehouse according to the relevant source updates of the web page in the Update Storage. Here we first give a brief overview of the WWM algorithm. First, the algorithm finds the XML web page to be maintained in the web warehouse. Then, it gets and removes the relevant source updates of the web page from the Update Storage. These relevant updates are sorted by their detection time to reflect their order of occurrence. Finally, it maintains the web page for each relevant update in sequence. A new version of the web page is obtained. Now we explain the WWM algorithm in detail. The algorithm executes the following steps in sequence:

1. Find the XML web page to be maintained WP in the web warehouse.
2. Get and remove a collection of relevant source updates RU from the Update Storage. RU is the collection of relevant source updates of WP.
3. Sort the updates in RU by their detection time recorded in the attribute *detect-time* to reflect their order of occurrence.
4. Create a version object V whose attribute *version#* is the current version number of WP plus one and whose attribute *update-time* is the current system time. V records the newest version information and will be stored in the next version of WP.
5. Change the newest version information of WP to V.

Algorithm Web Warehouse Maintenance Algorithm
- Find the XML web page to be maintained WP in the web warehouse.
- Get and remove a collection of relevant source updates RU from the Update Storage.
- Sort the updates in RU by the attribute *detect-time*.
- Create a version object V whose attributes are as follows:
  version# ← WP.*version.version#* + 1
  update-time ← current system time
- WP.*version* ← V
- **For** each update U of RU **do**

① **If** U.*content.action* = 'I' **then**
⇒ Find the maximum *child#* MN among all nodes whose parent is U.*content.p-node*
⇒ Insert a node whose attributes are as follows:
content.label ← U.*content.label*
content.value ← U.*content.value*
content.p-node ← U.*content.p-node*
content.child# ← MN + 1
content.s-action ← 'I'
version ← V

② **If** U.*content.action* = 'D' **then**
⇒ Find the newest version of node N such that N.*content.p-node* = U.*content.p-node* and N.*content.label* = U.*content.label*
⇒ Insert a node whose attributes are as follows:
content.label ← N.*content.label*
content.value ← N.*content.value*
content.p-node ← N.*content.p-node*
content.child# ← N.*content.child#*
content.s-action ← 'D'
version ← V

③ **If** U.*content.action* = M **then**
⇒ Find the newest version of node N such that N.*content.p-node* = U.*content.p-node* and N.*content.label* = U.*content.label*
⇒ Insert a node whose attributes are as follows:
content.label ← N.*content.label*
content.value ← U.*content.value*
content.p-node ← N.*content.p-node*
content.child# ← N.*content.child#*
content.s-action ← 'M'
version ← V

End Algorithm

Figure 5: The warehouse maintenance algorithm

6. Maintain WP for each update U of RU in sequence in a loop. For each update U, check to see if U is an insertion, a deletion, or a modification. If U is an insertion, insert a node whose parent node is U.*content.p-node* to WP. The attribute *child#* of the inserted node is the current maximum child number among its siblings plus one. This inserted node represents that a node was inserted to the source web page of WP. If U is a deletion, find the newest version of the node in WP that corresponds to the node that was deleted from the source web page of WP and insert a node to WP. This inserted node represents that a node was deleted from the source web page of WP. If U is a modification, find the newest version of the node in WP that corresponds to the node that was modified in the source web page of WP and insert a node to WP. This inserted node represents that a node was modified in the source web page of WP and becomes the newest version of the node in WP. After every update of RU is processed, a new version of WP is obtained.

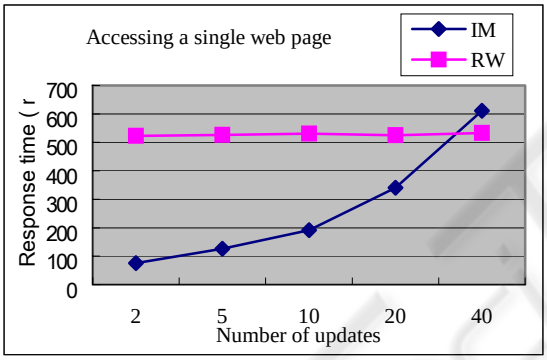
5 PERFORMANCE EVALUATION

We have implemented an experimental prototype system for change detection and maintenance of an XML web warehouse. The web warehouse is built on the ObjectStore object-oriented database management system and programs are written in the Java object-oriented programming language. Besides, we adopt the Aglets Software Development Kit (ASDK) and the Aglets Workbench both from IBM as the development tool and operating environment of mobile agents, respectively. The hardware platform consists of several personal computers that communicate with the Internet.

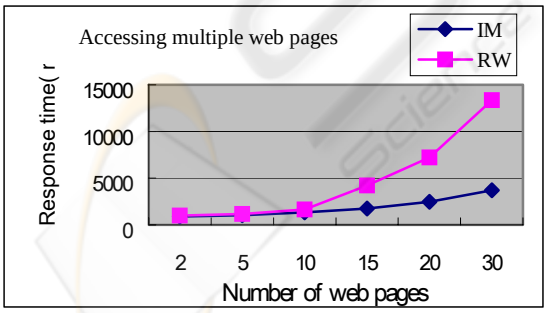
In the performance evaluation, we compare our approach with a rewriting approach to storage and maintenance of an XML web warehouse. In the rewriting approach, every version of a web page is completely stored. In our approach, on the other hand, every version except the first one stores only its difference from the previous version. As in our approach, the rewriting approach uses the deferred strategy to maintain a web page. However, it adopts a different method for change detection and warehouse maintenance. While receiving an access request for a web page, the web warehouse sends a request to the source web site of the web page for the current state of the web page. After maintaining the web page by creating a complete version of the current state, the web warehouse returns the requested web page to the user. In our approach, on the other hand, the requested web page is maintained using updates that have already been sent back by the mobile agent and stored in the local Update Storage. We compare our approach with the rewriting approach in terms of two important performance criteria: the response time and the storage space.

The response time is the elapsed time starting from the web warehouse receives an access request until it returns the requested web pages. From the point of view of the user of a web warehouse, the

response time is the most important criterion to judge the performance of the system. We separately consider two factors that may affect the response time, the number of updates to a web page and the number of accessed web pages. We first compare the response time of two approaches for accessing a single web page with different numbers of updates to the web page. An experimental result of such a comparison is illustrated in Figure 6(a). In Figure 6 and Figure 7, the abbreviations IM (standing for incremental maintenance) and RW (standing for re-writing) represent our approach and the rewriting approach, respectively. From the experimental results of this comparison, we observe two phenomena. First, the response time of our approach increases with the number of updates to a web page. However, the response time of the rewriting approach does not depend on the number of updates. Second, the response time of our approach is shorter than that of the rewriting approach in general, especially when the number of updates in smaller.



(a)



(b)

Figure 6: Comparison of the response time

We also compare the response time of two approaches for accessing multiple web pages. In this comparison, the number of updates to each web page is fixed and the numbers of updates to these web pages are small. An experimental result of such a comparison is illustrated in Figure 6(b). From the experimental results of this comparison, we observe two phenomena. First, the response time of both approaches increases with the number of accessed web pages. Second, no matter how many web pages are accessed, the response time of our approach is always shorter than that of the rewriting approach as long as the numbers of updates to web pages are small.

The storage space is the size of the secondary storage required for storing the historical data of a web page in the web warehouse. We consider two factors that may affect the storage space, the size of a web page and the update percentage of a web page. We compare the storage space of two approaches in terms of three different sizes (large, median, and small) and three different update percentages (30%, 50%, and 80%). An experimental result of such a comparison is illustrated in Figure 7, which shows the ratio of the storage space of our approach to the storage space of the rewriting approach. From the experimental results of this comparison, we observe three phenomena. First, the storage space of our approach is smaller than that of the rewriting approach in most of the situations. Second, the storage space of our approach increases with the update percentage of the web page. Third, our approach is more advantageous to web pages of larger size. It tends to require more storage space than the rewriting approach as the size of the web page gets smaller and the update percentage of the web page gets higher.

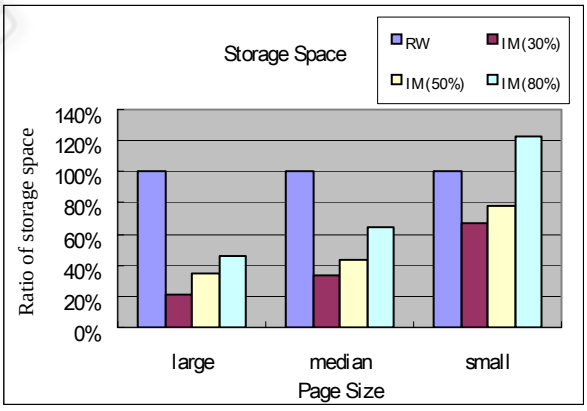


Figure 7: Comparison of the storage space

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed an approach to change

detection and warehouse maintenance for an XML web warehouse system. We compared our approach with a rewriting approach in terms of the response time and storage space of the web warehouse. In our approach, mobile agents are dispatched by the web warehouse to data sources and will actively and periodically detect and report changes of data sources back to the web warehouse. Beside, the web warehouse is incrementally maintained in response to source updates that have already been stored in the local storage. These can dramatically reduce the number of messages transmitted between the web warehouse and data sources. Therefore, our approach is more efficient than the rewriting approach in terms of the response time. With regard to the storage space, our approach adopts a change-centric approach in which every version of a web page except the first version stores only its difference from the previous version. This can dramatically reduce the size of the storage required for historical data of web pages in the web warehouse. Therefore, our approach is more efficient than the rewriting approach in terms of the storage space.

In the future, we will improve our approach in an attempt to further increase its efficiency. First, with regard to the storage of all versions of an XML web page, we will consider storing the current version completely and only deltas for historical data. Second, we will try to improve our warehouse maintenance algorithm so as to handle large number of updates more efficiently. Besides, more comprehensive experimentation will be performed to extract conclusive results.

REFERENCES

Agrawal, D., El Abbadi, A., Singh, A., Yurek, T., 1997. Efficient view maintenance at data warehouses. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pp. 417-427.

Apparao, V., 1998. Document Object Model (DOM) Level 1 Specification (Version 1.0).

Bhowmick, S. S., Ng, W. K., Madria, S. K., Lim, E. P., 2000. Detecting and representing relevant web deltas using web join. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pp. 255-262.

Chawathe, S. S., Abiteboul, S., Widom, J., 1999. Managing historical semistructured data. *Theory and Practice of Object Systems*, Vol. 5, No. 3, pp. 143-162.

Labio, W., Garcia-Molina, H., 1995. Efficient snapshot differential algorithm for data warehousing. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pp. 63-74.

Lim, S. J., Ng, Y. K., 2001. An automated change detection algorithm for HTML documents based on semantic hierarchies. In *Proceedings of the 17th IEEE International Conference on Data Engineering*, pp. 303-312.

Ng, W. K., Lin, E. P., Huang, C. T., Bhowmick, S., Qin, F. Q., 1998. Web warehousing: an algebra for web information. In *Proceedings of the 1998 IEEE Forum on Research and Technology Advances in Digital Libraries*, pp. 228-237.

Xyleme, L., 2001. A dynamic warehouse for XML data of the web. *IEEE Data Engineering Bulletin*, Vol. 24, No. 2, pp. 40-47.

Zhuge, Y., Garcia-Molina, H., Hammer, J., Widom, J., 1995. View maintenance in a warehousing environment. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 316-327.

Zhuge, Y., Garcia-Molina, H., Wiener, J. L., 1996. The Strobe algorithms for multi-source warehouse consistency. In *Proceedings of the 4th IEEE International Conference on Parallel and Distributed Information Systems*, 146-157.