XML Data Allocation in Hard Disks Based on Query Patterns

Xuehua JIANG† Haruo YOKOTA‡

†, ‡Department of Computer Science, Graduate School of Information Science and Engineering

Tokyo Institute of Technology 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan

E-mail: †jiang@de.cs.titech.ac.jp, ‡yokota@cs.titech.ac.jp

Abstract Applications of XML data are becoming more and more active and their volume is increasing day by day. To save high capacity of XML data, more than one hard disks are used. Decreasing power cost consumed by multiple hard disks and increasing XML data retrieving performance are important problems. This paper proposes an allocation algorithm for XML data in multiple hard disks by finding out XML tree/subtrees which are often issued together during the same transaction and put them into the same disk in order to reduce power consumption and enhance retrieve performance.

Keyword XML, Query Patterns, Multiple Disks, Data Allocation

1. Introduction

XML has emerged as a standard for data presentation and exchange on W3C. Data exchange format XML has been penetrating virtually all areas of internet application programming. As a result, volume of XML data is increasing day by day. Due to high capacity of data, multiple hard disks are used to save data. When more than one disks are applied, there is a problem for power consumption.

Several recent studies have pointed out that data centers can consume several Mega-watts of power [1]. It has been observed that power densities of data centers was grow to over 100 watts per square foot and that the capacity of new data centers for 2005 could require nearly 40 TWh per year. Recently, decreasing power consumption is one of hot issues.

In this paper, we propose a data allocation algorithm for multiple hard disks to save energy consumed by multiple hard disks by mining patterns from XML query log. Correct mining result offers good idea for data allocation in hard disks and suitable data allocation can lead to reduce consumption by putting disk which is not accessed frequently to a lower power consumption mode.

The reminder of this paper is organized as follows. Section 2 introduces existing common approach of power saving. Section 3 introduces related work about XML data caching algorithm and Section 4 provides XML data mining algorithm. Section 5 introduces data type and their allocation in disks. Finally, explanation about simulation environment and conclusion about the paper and future work is remarked in section 6 and section 7.

2. Common Approach of Power Saving

There is a common approach to reduce power consumption in hard disks which is shown in figure 1. The approach tries to put disk which is not accessed frequently to a low power consumption mode. There are 3 types of disk mode due to its disk rotation speed; active, idle and standby. When there is no read/write request to a certain disk, the disk enters to idle mode from active mode; if there is no access during a period of time, the disk spins down and goes continue to standby mode. In active mode, disk rotation speed is highest; as a result, power consumption is highest. In the Idle mode, power consumption is lower than in active mode; response cost is lower than in standby mode. In the standby mode, disk stops rotating. Therefore, power consumption is lowest. However, the response cost is highest because the disk should spin up to idle mode first, then go to active mode to process requisition. Both spin down and spin up process during mode transition consume some cost.

Data allocation in multiple disks tries to put data which are not issued frequently to a disk with low power consumption to stay disk in the low power consumption mode longer. In oppositely, data which are issued frequently should be put into a disk with active mode or idle mode in order to reduce spin down and spin up processes and offer high performance to the requisition.

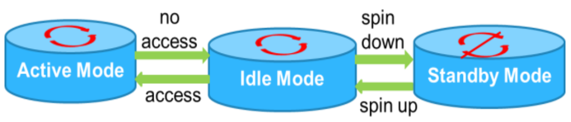


Figure 1: Disk Mode

3. Related Work

We should find out which data are issued frequently to propose a data allocation algorithm based on query patterns for multiple hard disks. Here we introduce related work about XML caching algorithm.

There are some works have been done about XML data mining algorithm so far. [2] and [3] proposed a XML caching system called XCache which processes XPath queries using cached XML data and combine result from cached data and from server in case that not all result can be retrieved from cache memory. [4], [5] and [6] proposed XML data mining algorithms based on query patterns to save data most retrieved frequently in order to enhance query performance. In paper [7], the authors proposed a XML cache management taking ancestor and descendant relationship into account. In many cases, a parent node contains more than one child node. When a child node is issued in a query, its parent nodes are issued together because of characteristic of XML tree structure. As a result, ancestors are issued more frequently and they are less likely to be replaced than descendants in cache memory.

All above studies focused on how to enhance XPath query performance and proposed mining algorithm for caching. However, these works have some defects. Fox example, they did not take attributes of a node, “//” or “*” into account. Sometimes, attributes contain very important information, such as id, therefore, we cannot ignore attributes with important information. Some queries contain “//” node or “*” to search any corresponding nodes or these special nodes are included in queries because not all users understand XML tree structure well. Another problem with existing algorithm is that they cannot be applied to data allocation algorithm directly because size of cache memory is much smaller than size of hard disk. Data allocation algorithm does not need to consider disk space so much because disk size is very big when compared with cache memory.

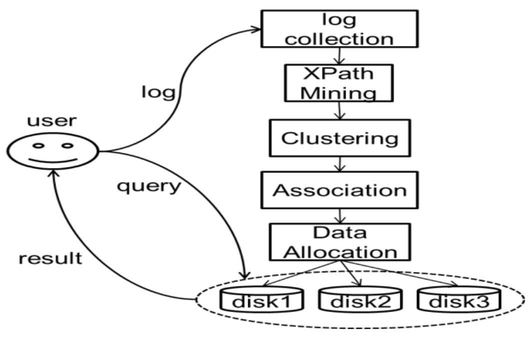


Figure 2: System Architecture

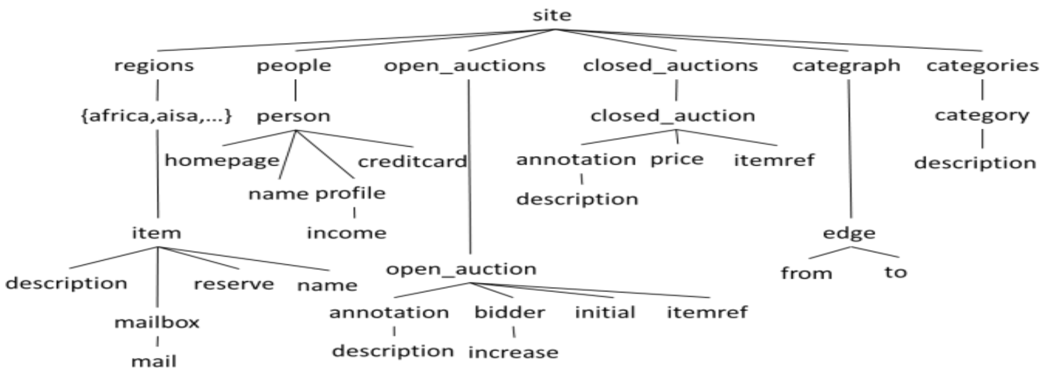


Figure3: XML File DTD

4. Mining Algorithm

Figure 2 shows system architecture for data allocation algorithm we propose. A user issued XPath queries to retrieve some elements from XML documents stored in multiple disks. Our system logs queries from users to apply XML query pattern mining. First, we mine data are put into standby mode; disks with frequent data stay in active or idle mode. When a requisition to a disk with standby mode is issued, the disk has to go back to active more to process the requisition and spin down to standby mode if there is no more requisition during a period of time. Improvement work about existing XML data mining algorithm is introduced in this section.

4-1.XPath Mining

XML data have tree structure and we can use XPath to indicate their ancestor and descendant nodes as well as their relationship. As Example, we apply XML data from Benchmark[8] program and file DTD is described in figure 3.

As special examples which include "//", "*" or attributes, I applied below 3 XPath queries.

Case 1: (with "//" node) We assume that a user issued a a XPath query "*site//name*". When we observe DTD, we can find that there are two different nodes with the same name "*name*"; one is a child node of item node, the other one is a child node of person node. In frequent trees which are frequently issued due to logs collected from users. Then cluster and associate mining result find out which part of XML data are issued frequently in order to propose data allocation algorithm for multiple disks. Data are allocated into different disks due to their access rate. Disks with infrequent this case, we do not know which node does the user want to search, therefore, we assume that both nodes are issued at the same time. XPathes of this query is "*site/regions//item/name*" and "*site/people//name*" respectively.

Case 2: (with "*", "//" node) In this case, we assume that user issued an XPath query to search a node with different search condition. The first query is "*site/regions//item[name = "John"]*". Node contained in [ ] mark is filtering condition. This query searches an item whose name is John. As a result, the search traverses only one node "*name*". The second query is "*site/regions//item[*="John"]*" to search child node of "*item*" node whose child is John. In this case, "*" can be matched with all child nodes of "*item*", including "*description*", "*mailbox*", "*reserve*" and "*name*". The last query is "*site/regions//item[// = "John"]*", searching an item whose child node or itself matches with "John". In this case, matching nodes are "*item*", "*description*", "*mailbox*", "*reserve*" and "*name*". When

Transaction 1	Transaction 2
Q1:site/people/person[@id=”person0”]/name	Q6:site/regions//item[name= “John”]
Q2:site/open_auctions/open_auction/bidder[1]/increase	Q7:site/regions//item[*=”John”]
Q3:site/people/person/profile/income	Q8:site/closed_auctions/closed_auciton/annotation/description
Q4:site/people/person[homepage]/name	Q9:site/closed_auctions/closed_auciton/annotation[description]
Q5:site/people/person/profile[income]	Q10:site/regions//item[//=”John”]

Table 1: XPathes

we consider all 3 kinds of search condition, we can conclude that “//” contain more information than a certain node or a “*” node. Therefore, we should mine the query with “//” node when other kind of search condition appears together with this kind of search condition.

Case 3: (with attribute of a node) In this case, we assume that there is a query “site/people/person[@id = “01”]” issued. Here, filtering condition is id, which is an attribute of the “person” node and contains unique information about “person” node. In this case, attribute value should be mined as well.

As example, we apply 10 queries and 2 transactions, 5 XPathes in each transaction. Detail of queries and transactions are described in table 1 and XPathes explained in tree format are shown in figure 4.

At first, we calculate frequency of each XPath. Every XPath is expressed as tree structure in figure 3 from which we can calculate frequency of each XPath. We use ***freq(XPath)*** to indicate frequency of a certain XPath. For example, *freq*(*site/regions*) indicates frequency of site/regions issued in a certain transaction. *freq*(*site/closed_auctions/closed_auction/annotation/description*) = 2, issued in Q7 and Q8 respectively.

freq(*site/people/person/”? ”*) indicates an XPath whose child node of “*person*” is not always the same one, or sometimes, there is even no child node for “person” node at all. Furthermore, we calculate support of each XPath, which is defined as

$$\mathrm{supp(XPath)} = \frac{\mathrm{freq(XPath)}}{\text{number of XPathes}}$$

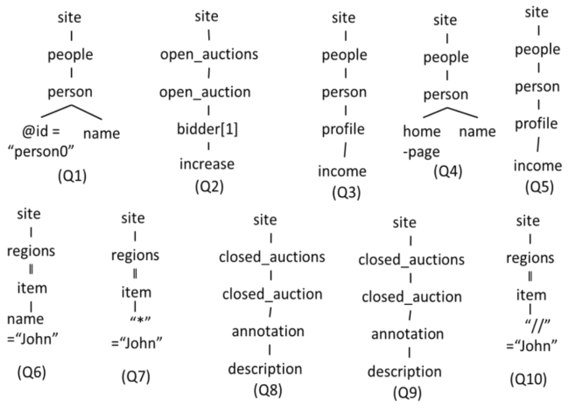


Figure 4: XTree

Support of each XPath can be simply calculated by definition above.

supp(*site/people/person/”?”*) = 0.4

supp(*site/open_auctions/open_auction/bidder[1]/increase*) = 0.1

supp(*site/regions//item/”?”*) = 0.3

supp(*site/closed_auctions/closed_auction/annotation/description*) = 0.2

Here “?” indicates actual data appears in queries. For the XPath *site/people/person/”?”*, “?” is “*name*” and “*id*” in Q1; “*profile/income*” in Q3 and Q5; “*homepage*” and “*name*” in Q4. To pick out XPathes which are issued frequently, we propose a minimum support (we use *min_supp* to indicate minimum support below.) to decide whether an XPath is a frequently issued one or not. XPath whose support is greater or equal to min_supp is considered as frequently issued XPath; others are not. In this paper, we apply min_supp as 0.2. Then all XPath except Q2 are considered as frequently issued XPathes.

4-2. XTree Clustering

Subsequently, we explain how to cluster XPath mined in above section. Clustering is process of combining XPathes with the same ancestor/parent nodes.

Clustering Rule: cluster tree should contain all nodes appeared in mined trees.

For the XPath *site/people/person/"?"*, *site/people/person* issued for four times over all, therefore Q1, Q3, Q4 and Q5 should be combined to produce a cluster tree. Similarly, clustering result for XPath *site/regions//item/"?"* and *site/closed_auctions/closed_auction/annotation/description* are shown in figure 5. In other word, {Q1 Q3, Q4, Q5} can retrieve result from cluster tree 1, {Q6, Q7, Q10} and {Q8, Q9} from cluster tree 2 and cluster tree 3 respectively, as well.

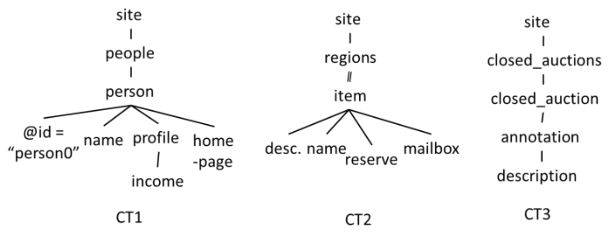


Figure 5: Clustering Tree

4-3. Association Rule

Association is process of making data sets from cluster trees in transaction unit. Association process should be done within a transaction because we try to find out which nodes are retrieved together frequently. Table 2 shows query information for per transaction in cluster tree unit.

We apply an association rate to calculate their relationship. In the definition, we only consider

Transaction 1	Transaction 2
CT1	CT2
NULL	CT2
CT1	CT3
CT1	CT3
CT1	CT2

* "NULL" means corresponded query is not contained in cluster tree.

Table 2: Transaction with Cluster Trees

combination of cluster trees, while existing caching algorithm consider correlation of cluster trees because we have enough space to save some duplicated nodes in disks. In the definition, $\boldsymbol{CT_n}$ is a cluster tree from clustering result, m indicates number of cluster trees in the same transaction.

$$\boldsymbol{asso(CT_1, CT_2, \dots CT_n) = \frac{freq(CT_1, CT_2, \dots CT_n)}{C_m^n}}$$

In transaction 1, $\text{asso}(C_1) = 4/C_5^1 = 0.8$. Similarly in transaction 2, association rate for CT2 is 0.6, for CT3 is 0.4, for CT2 and CT3 is 0.6. When we apply minimum association rate as 0.4, all cluster trees can be selected and we get 2 association trees, AT1 and AT2 respectively, which are shown in figure 6. There is an association rule which indicates that association tree should contain all nodes regardless keywords, "*" or "//" nodes. AT1 just equals to CT1 because transaction 1 contains only one cluster tree; AT2 is combination of CT2 and CT3 because data contained in both CT2 and CT3 are issued during transaction 2.

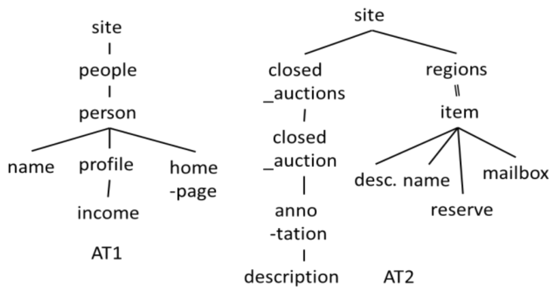


Figure 6: Association Tree

5. Data Allocation

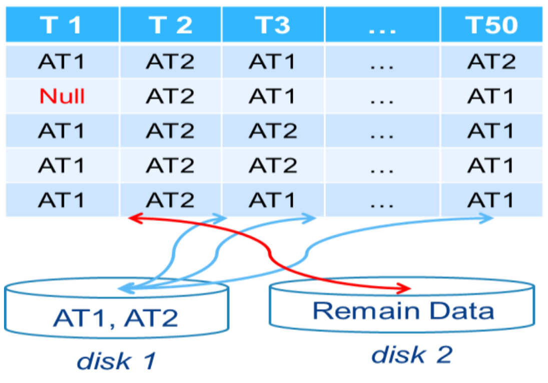
We classify data into three types: cache data, association data and remain data. Cache data is cluster trees, which are most like to be reissued. As a result, cache data is saved in cache memory to enhance query performance. Association data is aggregation of cluster trees due to association rule. In other word, it is combination of cluster trees which appear together frequently. Remain data is the data not contained in association tree. Therefore, original data should be aggregation of association tree and remain data.

Association data and remain data are saved in different hard disks respectively in order to reduce unnecessary actuation of disk for only a small part of its data. We assume that there are 50 transactions, each transaction contains 5 queries. XML data are separated into2 disks, disk 1 and disk 2, each holds frequent data and infrequent data respectively. Figure 7 shows assumption in our paper. In other words, association tree are saved in disk 1, and remain data are saved in disk 2.

In the first transaction, there is one query which is not contained in association tree, so result of this query should be retrieved from disk 2 which save remain data. Other queries are all contained in association trees; therefore, they can retrieve result from disk 1. Now we can put disk2 to a low power consumption mode after it is accessed in transaction 1, in other words, disk2 can be put into standby mode during transaction 2 ~ transaction 50.

6. Simulation

″auction.xml” from XMarkBench program is a file which will be used to do simulation. Size of this file is about 25M. Test data(query) are the same data set applied in a paper named “”which were applied to evaluate cache memory efficiency for XML. The data



T 1	T 2	T3	...	T50
AT1	AT2	AT1	...	AT2
Null	AT2	AT1	...	AT1
AT1	AT2	AT2	...	AT1
AT1	AT2	AT2	...	AT1
AT1	AT2	AT1	...	AT1

Figure 7: Data Allocation

set contain 100 queries, 96% of queries are frequent queries, and else 4 % are infrequent queries. Among frequent queries, there are 6 different queries, each appears 16% in the log.

Among the data set, we know which queries are issued frequently and which queries are not. There are only 6 different frequent queries, mining result are just the same 6 queries in the data set.

Clustering process is the process of combining queries with same ancestors. Association is a further process of combining cluster trees in transaction unit. Nodes contained in the association trees are marked as 1, others are marked as 2. Here, we assume that apply 2 disks, disk1 holds frequent data, disk2 holds infrequent data.

Data allocation affectivity is evaluated by disk_id and count value. If nodes with high count value are allocated in the disk1, in other word, the disk_id is 1, we can conclude that the algorithm is good.

When we evaluate power consumption, we should compare two cases; first case applies our data allocation algorithm, the other case does not apply our algorithm. In the first case, when a node in the disk1 is issued, the power cost is parameter1; when a node in the disk2 is issued, the power cost is sum of spin up + process request.

Total power cost during a period of time

= power cost in disk1 + power cost in disk2

In the case2, power cost can be calculated as below:

Total power cost = nodes issued during in a period of time * power for each node search + power consumed in active mode * period of time

If power cost in the first case is little than power cost in the second case, we can conclude that our algorithm can save power consumption.

7. Conclusion and Future Work

In this paper, we improved an existing XML data caching algorithm to apply it in multiple hard disks case and proposed data allocation algorithm due to association result in order to reduce power consumption.

In the future, we plan to do a simulation to evaluate data allocation efficiency and power consumption efficiency. We also try to propose suitable parameters due to evaluation result, such as minimum support for mining algorithm and minimum association rate for association process.

Acknowledgment

This research was supported in part by MEXT Grant-in-Aid for Scientific Research on Priority Areas (#201013017) and JSPS Grant-in-Aid for Scientific Research (A) (#22240005).

Reference

[1] J.Chase and R. Doyle. Balance of Power: Energy Management for Server Clusters. Proc. 8th HotOS, 2001.

[2] L. Chen, E.A.Rundensteiner and S. Wang, “XCache – A Semantic Caching System for XML Queries”, Proc. ACM SIGMOD, 2002.

[3] L.Chen and E.A.Rundensteiner, “ACE-XQ: A CacheE-aware XQuery Answering System”, Proc. WebDB , pp 31-36, 2002

[4] L.Chen, S.S.Bhowmick and L.T.Chia, “Mining Positive and Negative Association Rules from XML Query Patterns for Caching”, Proc.DASFAA, pp736-747, 2005

[5] L.H.Yang, M.L.Lww and W.Hsu, “Efficient Mining of XML Query Patterns for Caching”, Proc.VLDB, 2003

[6] C.Hua, “Frequent Query Patterns Guided XML Caching and Materialization”, Proc.WiCom, 2007

[7] D.Park and M.Toyama, “XML Cache Management Based On XPath Containment Relationship”, Proc. ICDE, 2005

[8] A.Schmidt, F.Waas, M.Kersten, M.J.Carey, I.Manolescu and R.Busse, “XMark: A Benchmark for XML Data Management”, Proc. VLDB, 2002