Vol.2011-DBS-152 No.5
Vol.2011-IFAT-103 No.5
2011/8/2

Evaluation of a Power Saving Method Storing XML Data

†Xuehua Jiang †† Yousuke Watanabe
†Haruo Yokoa

Abstract A growing number of applications are choosing to use XML format to transfer data on internet or save data. As a result, volume of XML data is increasing speedy so that data centers use several to some dozens of servers to store XML data. In this case, there is a problem for power consumption in hard disks. This study proposes an XML data allocation algorithm for power consumption(XAPC). We analyze query patterns to find out which part of file are not accessed frequently and separate this part to different disks to cut down power consumption by putting the disk into a low power consumption mode. Finally, we evaluate power consumption of proposed algorithm using a simulator.

Keyword: XML, Query Patterns, Data Allocation, Power Consumption, Evaluation

1. Introduction

XML has been 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. In this case, there is a problem that data centers consume so much electric power. Several recent studies have pointed out that data centers 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. Thus decreasing the power consumption consumed by data centers is one of urgent problems should be solved in near future.

In order to cut down electric energy in hard disks, we should put the disks which are not accessed frequently to a low power consumption mode. A disk should be spun up whenever it is accessed if it is in a low power consumption mode no matter how much of data are accessed in a request. Therefore, like in figure 1 which shows research overview, we should separate the data into frequent and infrequent part and try to put disks which store infrequent data into the low power consumption mode as much as we can.

There is our pervious study[2] which proposed XML data allocation algorithm. However, that study considers only count-based transaction which consists of the fixed number of queries. But that study did not consider time-based transaction. In this study, we consider time-based transaction and improve association algorithm proposed in earlier study and propose XML data allocation algorithm for power consumption(XAPC). Improved algorithm still tries to decrease power consumption in hard disks with slight influence of retrieve performance by analyzing query patterns. Correct mining result can offer good idea for proposing data allocation algorithm in hard disks and suitable data allocation algorithm could lead to cut down energy consumption by putting disk which is not accessed frequently to a low power consumption mode. Try to get correct mining result, we should retrieve all nodes including those issued by special symbols, such as "//" and "*". In addition, a query does not retrieve nodes only, but also retrieves attributes of the nodes as well. Therefore, we consider attributes as well as node in the mining algorithm. Data with high

† 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
†† Global Scientific Information and Computing Center, Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-Ku, Tokyo, 152-8552 JAPAN

frequency and that with low frequency are stored in the different disks respectively. The disk with data which are not accessed frequently is able to be in low power consumption mode, by which we can cut down energy consumption. Finally, we use a simulator to evaluate power consumption of our algorithm and compare the result with other approach.

The reminder of this paper is organized as follows. Section 2 introduces 3 types of disk mode. Section 3 introduces related work about XML data caching algorithm. The caching algorithm could not be applied to power saving in hard disks directly . However, it can offer preliminary idea about data allocation algorithm. Next in Section 4 and section 5, we introduce detail mining algorithm and data allocation algorithm respectively. We have done simulation to evaluate the algorithm we proposed. All particulars and result will be remarked in Section 6. Finally, conclusion and future work will be mentioned in the last section.

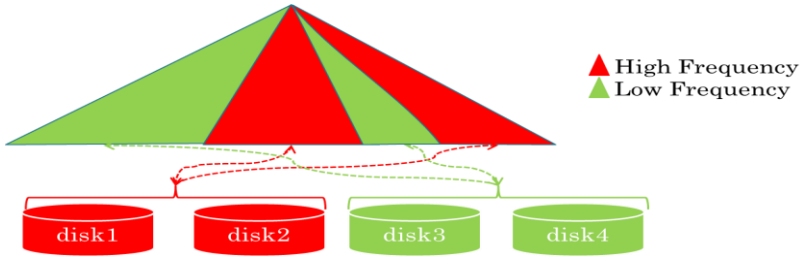


Figure 1: Research Overview

2. Disk Mode

There are 3 different types of mode due to its power consumption: active, idle and standby.(shown in figure 2)

Active Mode: A disk is in active mode when it is processing read or write requests from outside. In the active mode, both disk and magnetic header are rotating to looking for a certain address on disk. As a result, response cost is very low in the active mode. However, power consumption is very high.

Idle Mode: When the disk finished processing requests, the header stops moving. We call this mode as idle mode. In idle mode, power consumption is a bit lower than in active mode and response cost is still very low.

Standby Mode: When there is no access to a disk for a certain period of time(which we call spin down threshold below), the disk spins down disk rotation and the disk stops finally. This is standby mode. The disk should spins up disk rotation to idle mode first, then further to active mode to process requests if there are requests to the disk in standby mode. Generally, spinning up takes from several to a dozens of seconds. As a result, response cost is very high. However, standby mode is the mode that consumes minimum power cost.

The data allocation algorithm in hard disks tries to separate the data into several parts due to its access frequency. When there is a request to the disk in standby mode, the response may take some seconds. However, average response cost is not influenced so much because almost requests happen to the disks which hold frequent data.

In order to separate file due to its access frequency, we should catch which parts of file are requested frequently. Therefore, we should mine frequent nodes by analyzing query log from users first.

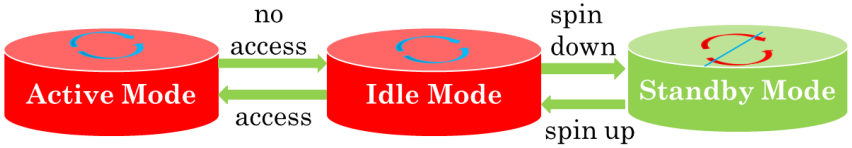


Figure 2: Disk Mode

3. Related Work

As mentioned above, caching algorithms can offer preliminary idea about data allocation algorithm because caching algorithms try to find out the frequent nodes as well. In addition, there is an earlier work which proposed allocation algorithm for XML. Here, we introduce related work in two subsections.

3.1 Caching Algorithm

[3] and [4] proposed a XML caching system called XCache. XCache processes XPath queries using cached XML data and combine result from cached data from remote server in case that not all results could be retrieved from cache memory only. [5] and [6] proposed XML data mining algorithms based on query patterns to save data most retrieved frequently in the cache memory in order to enhance query performance. In [7], authors proposed a positive and negative association rules for XML caching algorithm called LRU_AR. In that work, a node could not be associated into more than one association tree because cache memory has a limitation of its small size that cache does not save duplicated data. In paper [8], authors proposed a XML cache replication management taking ancestor and descendant relationship into account. In many cases, a parent node contains more than a single child node. When a child node is issued in a

query, its parent nodes are issued together because of spatial tree data structural of XML. As a result, ancestors are issued more frequently than descendants and they are less likely to be replaced than descendants in cache memory.

All above studies focused on cache memory and tried to enhance XPath query performance. However, these works have some defects. For example, they did not take the cases when XPath contains some special symbols into account. Generally, an XPath may contains "*" and/or "//" symbols to express some nodes between ancestors and descendants. In this case, the nodes are retrieved as well though they are not express directly in the XPath. Another defect of previous work is that they did not consider attributes of a node. However, the attribute may contain important and valuable information of the node. Thus attributes should be cached as well.

3.2 XML Allocation Algorithm

Earlier, a paper[2] proposed an XML data allocation algorithm. However, we could not find out well which part of data are accessed frequently because transaction definition in that work does not work well. In this work, we redefine transaction and apply Apriori algorithm[9] to make data set.

4. Mining Algorithm

Mining algorithm contains 3 different steps: XPath processing, node mining and association. XPath process is the step that retrieves multiple nodes even they are not issued directly. In the mining step, we mine frequent nodes from the result of XPath process. In association step, infrequent nodes are associated into different class set. The nodes which are likely to be issued together frequently are contained in the same association tree. Finally, we allocate the data due to association result. The data with high access frequency are put into the same disk to offer good retrieve performance. Similarly, infrequent data which are contained in the same association tree are put into the same disk in order to get result just from a disk. It means that there is only a single disk spins up to idle mode. We could save energy by spinning up only a disk than spinning up several disks for a request. In other to process XPath, we should collect log from user before this step. The log is a record of data which contain time stamp of an XPath, XPath ID and XPath. All the steps are shown in figure 3.

4.1 XPath Processing

Xpath processing could be called as XPath expansion step as well because we try to retrieve multiple nodes which are not issued directly and the XPath may be expanded after this step. In our study, we consume that the requests to XML files are in format of XPath. XML data have characteristic of tree structure and we can use XPath to indicate XML's ancestor and descendant nodes and their relationship. In this case, XPath may contain some special symbols, such as "@", "*" and "//". Therefore, we should analyze XPath in the first step. Here, we consider above three special symbols. As example, we use XML data DTD from benchmark[9] program. (shown in figure 4)

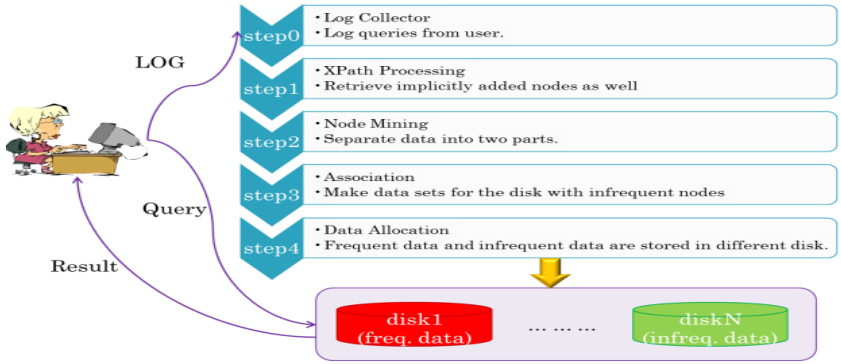


Figure 3: Research outline

Case 1(DTD contains two different nodes with same expression): Some DTD may contain two different nodes with same expression. For example, we assume that an XPath query "*site//name*" is issued. We can find two different nodes with the same expression "*name*" in the DTD when we observe the XML file DTD shown in figure 4. One is a child node of "*item*"; the other one is a child node of "*person*". In this case, we should consider that both nodes expressed as "name" are issued because system does not know which node do the user retrieves and returns both result. In fact, both two XPathes are issued: "site/regions//item/name" and "site/people//name".

Case 2(with "*" and/or "//" symbol(s)): "*" is a wildcard in the XPath and "//" is abbreviation of more than a single original nodes that implicitly added to the between ancestors and descendants or themselves[11]. In this case, of course, the nodes expressed by "*" and/or "//" are not expressed directly. However, they are issued. For instance, we use below 3 different XPathes with different filter conditions to search a node.

XPath1: "*site/regions//item[name="John"]*"
XPath 2: "*site/regions//item[*="John"]*"
XPath 3: "*site/regions//item[//="John"]*"

The first XPath searches an item whose child node "*name*" matches with "*John*". In

this case, the system traverses only the "*name*".

The second XPath searches an item whose children match with "*John*". In other word, the system returns all results when it finds any child nodes of "*item*" matches with

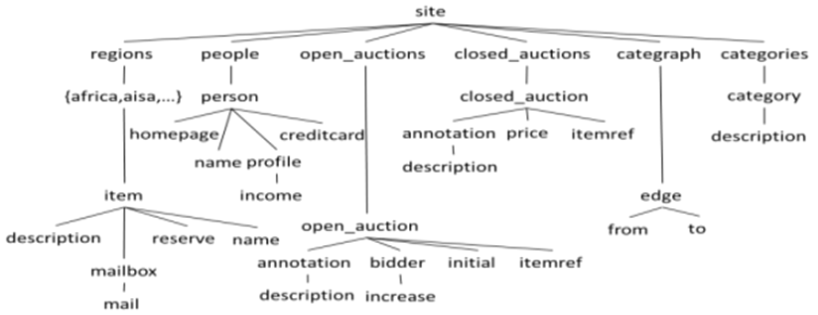


Figure 4: XML File DTD

"John" because "*" is a wildcard. In this case, the system traverses all child nodes: "*name*", "*mailbox*", "*desc*", "*res*". Therefore, the log after XPath processing should contain all nodes retrieved.

The last XPath searches an item whose descendants or the node itself matches with "*John*". In this case, the system traverses all descendants and itself: "*item*", "*description*", "*mailbox*", "*mail*", "*reserve*", "*name*". Similarly with above case, the log after XPath processing should contain all nodes retrieved.

Case 3 (with "@"): "@" is expression of an attribute of a node due to XML specification[12]. Sometimes, the attribute contain valuable and/or unique information about a node, such as ID of a node. As a result, there will be an XPath which search a certain node with its ID information. In this case, we should mine the attribute as well as node.

4.2 Node Mining

Subsequently, we do node mining in this step. Purpose of the node mining is to separate the data into two parts: data with high access frequency and data with low access frequency. We use a minimum hit_count (which we call as minimum support as well later) to decide where a class is a frequently access one or not. Class means attribute and element of a node with the same name. In other word, when an instance of a certain node is issued in a query, we consider that all instances are issued at the same time, because all the instances should be read to retrieve final result. As a result, the best way to calculate hit_count is in class unit. However, a node may appear more than once in different position, in this case, we add a digital number at the end of the node to distinguish the nodes.

As example, we use 6 different XPath log to explain process of node mining which is shown in table 1. The XPathes show both original XPath and its result after applying XPath processing. Node highlighted with red is the node implicitly added and retrieved in fact.

The node is decided by its hit_count, as a result, we should +1 to the class when its instance issued once. Table 2 shows statistic result grouped by hit_count.

If we apply the minimum hit_count as 4, which means that the class whose hit_count is greater or equal to 4 is sorted as frequent classes; other classes are sorted as infrequent classes. In the example, we consider the 5 classes ("*site*", "*regions*", "*africa*", "*item*", "*name1*") are frequent nodes, else are infrequent nodes. In the previous work[7], the hit_count for "*africa*", "*mailbox*", "*mail*", "*des*" and "*res*" is 0 because they do analyze XPath before mining step. In other words, the nodes express directly in the XPath can be mined by previous algorithm[7]. In our algorithm, "*africa*" and "*name1*" are sorted as frequent nodes.

Table 1: XPath Log

Time	ID	XPath
0s	Q0	site/people/person[@id]/name0
8s	Q1	site//name →site/regions/Africa/item/name1 →site/people/person/name0
20s	Q2	site//open/bidder/increase →site/opens/open/bidder/increase
27s	Q3	site/regions//item[name="John"] →site/regions/africa/item/name1
29s	Q4	site/regions//item[*="John"] →site/regions/africa/item/name1,mailbox,des,res
35s	Q5	site/regions//item[//="John"] →site/regions/africa/item/name1,mailbox,mail,des,res
...	...	... ...
1290s		site/people/person

Table 2: hit_count Result

hit_count	classes
7	site
4	regions,africa,item,name1
3	people, person
2	name0, mailbox, des, res
1	@id, opens, open, bidder, increase, mail

In node mining step, we get two part data: **frequent data** and **infrequent data**. Frequent data are the data whose hit_count is not less than minimum hit_couint; infrequent data are the rest part of the file.

4.3 Association Algorithm

Association is process of making data sets which contain classes issued in the same transaction frequently. Association algorithm is applied to infrequent nodes only. The purpose of association step is to make data sets which contain the nodes accessed together frequently that the XPath could retrieve result from only a single disk. It means that there is only a single disk processing the request. If the disk is in standby mode, it should spin up disk rotation to active mode. There will happen power consumption in the process of spin up. Therefore, the best case is when we spin up only a disk to minimize power consumption.

The association is done in transaction unit. In our work, we define transaction as a query set contained in a certain period of time. We define a period of a transaction as 13 seconds; a spin down threshold which is referenced from a specification of a disk from Hitachi Co., Ltd,.[13]. Then XPath queries issued in the same time slot are contained in the same transaction. Meanwhile, a transaction is a set of classes which are issued in the queries in the transaction. However, we calculate the hit_count in query unit because some class may be issued more than once if the transaction contains several queries.

Figure 5 shows queries in transaction unit. Transaction0 is set of queries issued between (0~13)s. In our example, Q0 and Q1 are contained in transaction0. Similarly, we make 1000 transactions from (0~1300)s. Object of association algorithm is the classes sorted as infrequent part.Table 3 shows infrequent classes order by hit_count which are sorted as infrequent data in the mining step.

Association process is done by using Apriori Algorithm published in [9] which finds out frequent item sets. We make an item set until the size of classes in the item set is

Table 3: Infrequent classes

hit_count	class(transaction)
3	people,person(T0,T999)
2	name0(T0) mailbox,res,des(T2)
1	@id(T2) opens,open,bidder(T1) mail(T2)

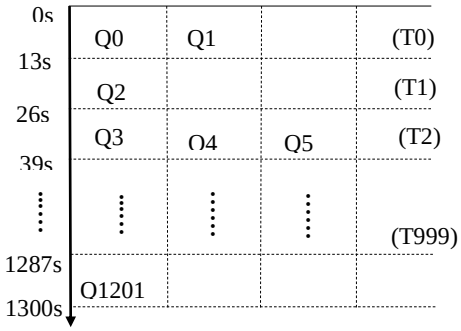


Figure 5: Transaction

equal to the size of disk because a disk has a class set and all the classes contained in a class set should be stored in the same desk. The size of an item set is big if the size of class is small; the size of an item set is small if the size of class is big.

5. Data Allocation

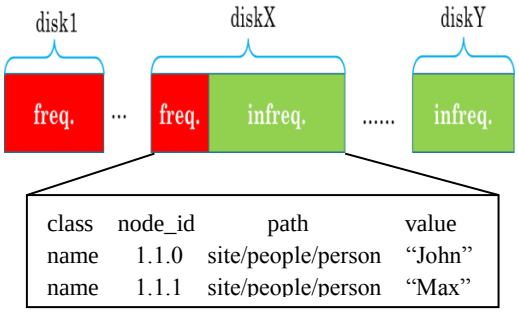
XML data are stored in class unit. A class may have several instances and all these instances of the class are stored in the same disk. Data are allocated in two steps.

Step 1: Frequent data are allocated in the first step. They are allocated order by its hit_count from disk1 to disk X (figure 6). The number of disk in high power consumption mode is X.

Step 2: In the step 2, we allocate class sets got with association algorithm. Each class is corresponds to a certain disk and the classes included in the class set are allocated to the correspond disk. The class sets for disks from disk(X+1) to diskY have the same sizes which are equal to a single disk size. The size of frequent data may not always be times of disk size, in other words, diskX may have free space for infrequent data. As a result, we should make a data set for diskX whose size is equal to the free space in the diskX.

There is a table(Table 4) from which we could get information about in which disk a certain class is stored. Each class has a unique id to distinguish with others. Therefore, when the system processes a request, it should first read this table to confirm to which disk the request should be sent. There should be another table in each disk shows relationship between parent node and child node to confirm which instance should be returned to client. If the result are form more than 1 disk, the result from different disks

should be combined first then return to client.



class_id	disk_id
0	0
1	X
2	Y
...	...

Figure 6: Data Allocation

Data should be reallocated if the power consumption of disk in high power consumption mode gets to lower, oppositely, the power consumption in disk in low power consumption mode gets to higher, or the performance gets worse. This case will happen when the query patterns are changed. For example, hot search topic is changed from a field to another field which hardly has relationship. In this case, we should analyze query patterns again and reallocate data to save power consumption.

6. Simulation and Evaluation

Finally, we do a simulation to evaluate power consumption in hard disks and performance. We compare three different approaches: striping method, LRU_AR and XAPC.

Striping Method: Striping method is the case that data allocated in the disks randomly without considering query patterns.

LRU_AR: LRU_AR is the approach which analyzes query patterns, but did not take special symbols into account;

XAPC: XAPC is the approach which is proposed in this paper.

6.1 Simulation and Parameters

Table 6 shows simulation environment and parameters used in the simulation. Parameters are referenced from the disk specification[13]. We select the DTD from benchmark program[10] as a simulation file. The DTD contain 100 different classes, including 6 attributes. We make a query set, which are made up by 96% frequent queries and 4 % infrequent queries. Frequent query set contain 41 different queries, which are also from XPathMark Bench program[10]. Among them, 4 XPathes have "//" expression (9.76%), 2 have "*" symbols (4.88%) and 14 XPathes have retrieve attributes as well. An infrequent query is made by nodes which contains 2 ~ 11 different classes and/or attributes chosen from the DTD randomly. Time intervals are generated by a random algorithm from our simulation system as well which is between 0 to 59 seconds.

Table 6: Environment and Parameters

Environment	Parameters
OS: Windows 7	active_power: 15W
Language: Java 1.6	idle_power: 11W
Database: PosttreSQL 8.0	standby_power: 2.5W
queries: 1000	spindown_power: 2.5W
min_supp: 0.100,0.015,0.010,0.005,0.001	spinuup_power: 2.5W
time interval for queries: 0~59s	spindown_threshold: 13s
disk number: 4	spinup_time: 14s
	spindown_time: 14s

6.2 Evaluation of Power Consumption

Figure 7 shows evaluation of power consumption. Power consumption of striping algorithm is greater than other algorithms because all 4 disks are in the high power consumption mode almost all the time. Power consumption of XAPC is less than of LRU_AR algorithm when we apply the same minimum support. We can notice that electric power consumed in idle, standby and spin down mode are almost same in all 3 algorithms. Therefore, the main difference in power consumption is coursed by the difference in the spin up mode. Thus, we observe how many times the disks are spun up for each algorithm. Figure 8 shows calculate result for spinning up. Similarly with the evaluation result of power consumption, striping has maximum times of spinning up and XAPC has the least times.

6.3 Evaluation of Performance

Above two evaluations focus on how much power consumption we can save. However, the algorithm is not useful if its response cost is very high. Then, we evaluate the performance in this step. Here, performance is measured by the rate of QPS(*number of queries per second*)/PPS(*power consumption per second*).Figure 8 shows evaluation result. We can see that performance gets better when we apply LRU_AR and XAPC

than use striping. It means that we can cut down power consumption at the same time offer better performance.

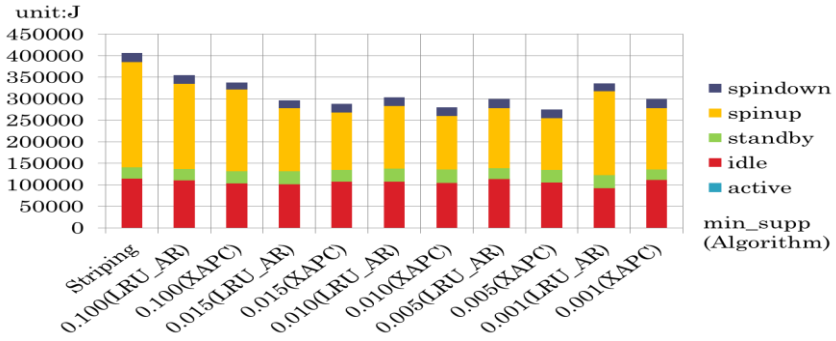


Figure 8: Power Consumption

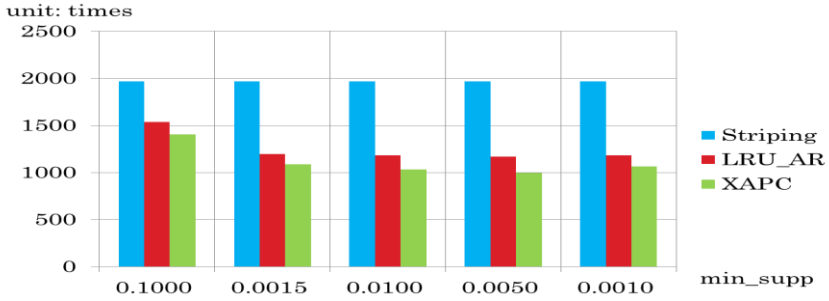


Figure 9: Spinning Up Count

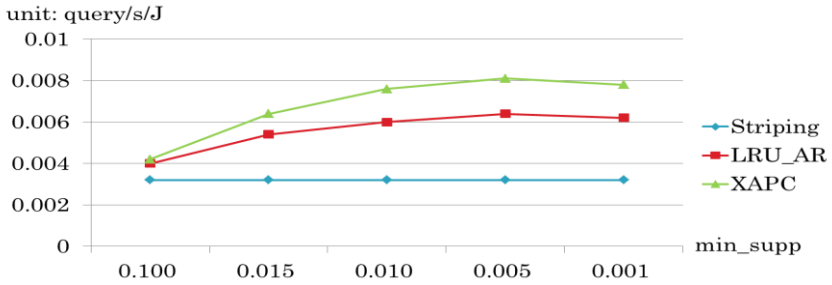


Figure 10: Performance (QPS/PPS)

7. Conclusion and Future Work

This study proposed an XML data allocation algorithm for power consumption by analyzing query patterns in XPath format. We separated the approach into 3 steps: node mining, association and data allocation. We also did a simulation to evaluate power consumption and performance.

In our method, power consumption is minimum when the minimum support is set as 0.005 and offers best performance. Almost half of the data are sorted as frequent part and 2 disks among 4 are in high power consumption mode when the minimum support is 0.005. This result means that the power consumption is least when we use the disks in high power consumption mode effectively.

In the future, we plan to improve our algorithm continuously and do a real experiment to evaluate actual result using XML data from Wikipedia, DBLP etc.. Other work we are planning to do is doing experiment in instance level which means that the data are allocated in instance unit, which they are allocated in class unit in this wrok. We allocate data in class unit in this work, however, the evaluation result may different if we allocate data in instance unit. Thus, we want to compare the result and analyze difference.

Acknowledgment

This research was supported in part by JSPS Grant-in-Aid for Scientific Research (A) (#22240005).

References

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

[2] X.Jiang and H. Yokota.”XML Data Allocation in Hard Disks Based on Query Patterns”, Proc.DEIM,2012

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

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

[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] 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

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

[9] Agrawal, R., Imielinski, T. and Swami, A.: “Mining association rules between sets of items in large database”, SIGMOD Rec., Vol.22, No.2, pp.207-216(1993)

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

[11] http://www.w3.org/TR/xpath20/

[12] http://www.w3.org/TR/xml/

[13] http://www.hitachigst.com/tech/techlib.nsf/techdoc