1998 International Computer Symposium
Workshop on Software Engineering and Database Systems
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A PRE-FETCH MECHANISM FOR PROXY SERVERS: USING ASSOCIATION RULES

Kuen-Fang J. Jea, and Cho-Jan Chen

Institute of Computer Science, National Chung-Hsing University,
250 Kuo-Kuang Rd., Taichung, Taiwan, R.O.C.
Email:{kfjea,cjchen}@cs.nchu.edu.tw

ABSTRACT

We consider the problem of decreasing users' waiting time in accessing WWW objects. The traditional solution uses the proxy server to cache remote objects. In this study, we propose a pre-fetch mechanism which uses the technique of mining association rules to improve the cache hit ratio of the Squid proxy server. According to the requests made by proxy server users, this mechanism can pre-fetch WWW objects that are probably accessed next and store them in the local cache of the proxy server. The user can then fetch the next objects in a short time. To verify the feasibility of this pre-fetch mechanism, we use user access logs in the proxy server to test the performance of our mechanism. Experimental results show an improvement of about 30% of the cache hit ratio, and learning more association rules can even increase the cache hit ratio about 47%.

1. INTRODUCTION

World Wide-Web is a fast growing medium which works like a distributed multimedia database system. The increasing number of WWW users and huge amount of WWW information make a heavy workload of the Internet. Users usually spend a lot of time in waiting data transfer. The traditional solution to this problem is using the proxy server [7]. The proxy server is generally good for users who are interested in the same information because the cached objects can be reused. But most of the time, the proxy server can not satisfy the need of information from users with diverse interests. And many configuration setups of the proxy server, like the size of the cache disk, the expiration time of the cached objects, etc., also influence the performance of the proxy server. There is no simple way to manage it except the manager's experience. This may lead some proxy servers to only 10% of cache hit ratio. So in this study, we develop an approach that can increase the cache hit ratio of the proxy server and reduce users' waiting time in accessing WWW objects.

The proxy server provides some useful mechanisms for people in the subnets to access the Internet. Its main idea is that the same documents may be requested more than once by the same or different users, so it makes sense to save a local copy in the proxy server and no longer go to the source again to request the same documents in a period of time. Squid [12] is one of the popular proxy servers. It offers high performance proxy caching for Web clients, and supports FTP, Gopher, and HTTP requests. Its cache can be arranged hierarchically for an improvement in response time and a reduction in bandwidth usage. This software is designed to operate on any modern Unix system. The above merits make Squid be widely used by many campuses and ISPs.

Data mining [3, 1, 2, 10], or knowledge discovery in large databases, is a technique of combining machine learning, statistics and database. It analyzes enormous sets of data in a large database and extracts both the implicit meaningful relations and the implicit interesting knowledge. Recent development has several directions for different applications. One of various important techniques is mining association rules [3, 4, 11] from a large database.

The association rules describe the implied relations between sets of data in a large database. Some famous algorithms for mining association rules are proposed in the related work, which are Apriori [4], AprioriTid [4], DHP [8, 9]. Generally speaking, these algorithms first generate potentially large itemsets based on some candidates and count the actual support for these candidates over the original data. Finally the large itemsets can be determined from these candidates.

Proxy servers can not alleviate the heavy traffic in the Internet efficiently. So in this study, we propose a pre-fetch mechanism which can reduce more users' waiting time than that of a pure proxy server. This mechanism is a combination of the technique of mining association rules and Squid proxy server [12]. This idea comes from the W3 Miner system [6]. The working procedure of the W3 Miner system is as follows. When a WWW object request is passed through the interface of this system to the Internet, a message is sent to the W3 Miner, waiting for the predicted next document from the W3 Miner, making another WWW request to pre-fetch the predicted document, and storing the predicted document in the local cache. Then the interface will also reply a message about the predicted document to the user's browser, which becomes a reference document for the user when he fetches the next one. But there exist several problems in this design. First, the main function of this pre-fetch system is very similar to that of a proxy server. Since the proxy server has been accepted in many fields, to implement and promote such a similar system is a difficult task if it is not powerful enough. Second, in the HTTP 1.0 protocol, one WWW document (homepage) consists of several objects. The predicted object may be just a background of a homepage or an advertisement. It may not have any important meanings as a reference document for users. Third, a new web browser needs to be used when the user uses the W3 Miner. These are all the difficulties in implementing the W3 Miner.

A similar pre-fetch mechanism, Wcol, is also proposed by Ken-ichi Chinen (from Nara Institute of Science and Technology, Japan). Wcol simply analyzes the WWW request and pre-fetches all the linking text objects. Then it caches these text objects like a proxy server does. This mechanism however, cannot save users' waiting time very much because only text objects are pre-fetched and other types of objects (such as graphics, audio, images) are ignored.

In order to overcome the disadvantages of W3 Miner and Wcol, we treat WWW as a large database and every WWW object is an item in this database. We develop an algorithm for mining association rules in WWW and use it to implement a pre-fetch mechanism in the Squid proxy server. Our mechanism first collects data from the proxy server. We use the algorithm of mining association rules to count out the associative objects beyond some specified *minimum support* and *minimum confidence*. The support is the number of transactions where the object appears. And the confidence is the probability that the antecedent may imply the descendant. Later, when a user makes a request for a WWW object which corresponds to some association rule, the mechanism will pre-fetch the predicted object specified in that rule and store it in the cache of the proxy server. So the user can save his time in waiting object transfer if he indeed requests the predicted object next time.

The rest of this paper is organized as follows. Section 2 describes the pre-fetch mechanism and its features. In Section 3, we design several experiments to test this mechanism and show that this mechanism can raise the cache hit ratio of the proxy server. Last, we discuss the impacts of our pre-fetch mechanism and conclude this study in Section 4.

2. PRE-FETCH MECHANISM

The problem that this study investigates can be defined as follows:

> In a WWW environment, given a proxy server, how can we find an approach that can increase the cache hit ratio of the proxy server and reduce users' waiting time in accessing WWW objects?

In order to solve this problem, we design an algorithm for mining association rules in WWW and use this algorithm to implement a pre-fetch mechanism.

2.1 Approach

We divide our approach into three phases. The first two phases are derived from mining general association rules, while the third one is an application of the association rules discovered in the previous two phases. In the first two phases, we use a similar procedure as showed in [1, 4, 5] but adapt it for the WWW application.

There are two kinds of association rules existing in WWW. The first one is that the antecedent and descendant objects belong to the same homepage. The other is that antecedent and descendant objects do not belong to the same homepage but they have semantic relationships so that they will be accessed in sequence. If we want to find out the relationships between them, we need a lot of user access records to support this type of association rules.

Before describing the mining procedure, we give some definitions. An *item* means a WWW object; an *itemset* is a set of WWW objects; a *transaction* indicates the itemsets visited by some WWW users in a period of time. And all of the transactions are stored in a database.

- **Phase 1: Find Out Large Itemsets**

In this phase, we scan the original transaction database to find out all large itemsets.

The algorithm for discovering large itemsets can be separated into several passes. In the first pass, we count the support of individual items and determine which of them are large (i.e. beyond the minimum support). We call them *large 1-itemsets*. Then we store them into a new database table as a seed set. In the next pass, we use this seed set to generate potentially large 2-itemsets, called *candidate itemsets*. Every pair of two items can form an itemset. We start with this table to find out the actual large 2-itemsets. Those 2-itemsets, which are beyond the minimum support, are stored in a table for counting out the desired association rules.

- **Phase 2: Generate Desired Association Rules**

We use $<s, t>$ to denote the two components of each itemset in large 2-itmesets. We have already counted the support of s, t and $<s, t>$, then we count the confidence of every pair of $<s, t>$. If the quotient of $support<s, t> / support<s>$ is larger than the minimum confidence, then this itemset is our desired association rule. We store this rule in a database table for the next phase. The item s is the antecedent of the association rule, and the item t is the descendant.

We do not limit the number of times that an item appears, so the antecedent and descendant are not one to one mapping although every pair is unique. That means one antecedent may lead to several descendants, and one descendant may have several antecedents. The relationships between them may be either in sequence or correlated. A user may visit one of these association rules or none.

- **Phase 3: Pre-fetch Next Item**

We have already derived some association rules of which items were visited by some WWW users. When someone

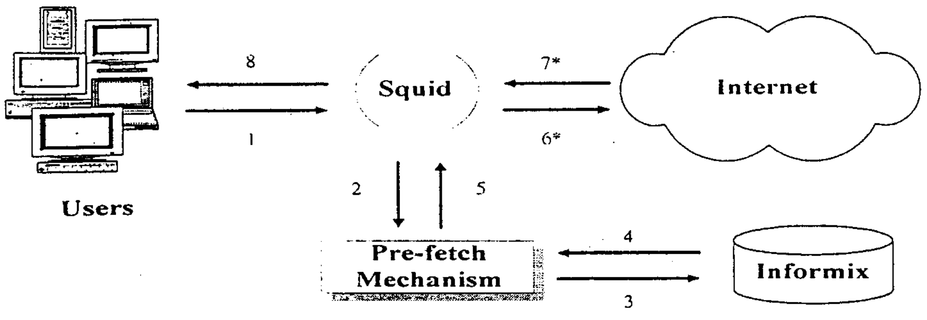

Figure 1: System Model

visits the antecedent of one of the rules, he may visit one or none of the descendants following the antecedent. If we pre-fetch all the following descendants, the user can easily read one of them in the cache of the proxy server. But this action will make the Internet traffic worse, so we just choose the one with highest confidence to pre-fetch. If this antecedent does not exist in our association rules, we just do nothing.

2.2 Implementation

According to our pre-fetch algorithm, we implement a mechanism in the following platform to verify its feasibility.

Machine	*Sun SPARC classic* with 32MB DRAM
OS	*SunOS Release 4.1.3*
Proxy server	*Squid 1.1.18* with 500MB cache
Database	*Informix 4.11*
Language	*GNU C and C++ compiler v2.7* *Informix ESQL/C version 5.0* AWK

Our pre-fetch mechanism is independent of the Squid proxy server, and all messages are passed through the proxy server. All the mining information including association rules is stored in Informix. The system model for our mechanism is showed in Figure 1.

The meaning of each message passing in the model is described as follows.

1. **Users → Squid**: Request a WWW object.
2. **Squid → Pre-fetch Mechanism**: Pass the requested object to predict the next requested object.
3. **Pre-fetch Mechanism → Informix**: Use this requested objet to search association rules in the Informix table, and find out the corresponding antecedent.
4. **Informix → Pre-fetch Mechanism**: Reply the descendant of the antecedent in some association rule.
5. **Pre-fetch Mechanism → Squid**: Make a request for the descendant.
6. **Squid → Internet**: Make a request to the original web site in the Internet if Squid does not have the requested object in its cache.
7. **Internet → Squid**: The Internet replies the WWW object.
8. **Squid → Users**: Return the requested object to the user.

We use the *access.log* file of Squid to collect the mining information. This file is used to store the information of the request, like timestamp, requested URL, client address, and so on. The details are described in [12].

Before we start mining association rules according to the mining algorithm, we separate these request objects into transactions. The set of WWW objects, which are visited by the same client in a period of time, is defined as a transaction. We only count the object which will take a long transmission time, because short transmission time will not make the user spend a lot of time in waiting. In the first phase, we give an identifier for each transaction and store the information of the transaction objects into the *access_log* table (as showed in Table 1). Then we can easily find out the items (1-itemset) which are larger than the specific minimum support value, and store them into another table, *first_frequency* (as showed in Table 2). Like the previous procedure, we set a different minimum support value at this procedure to find out large 2-itemsets. All the 2-itemsets are stored into the Informix table *pre_rule* (as showed in Table 3). In this table, each pair of two objects is a potential association rule. All the above procedures can be finished by SQL query statements.

Then we enter the second phase. By using these supports of large 2-itemsets, we can calculate their confidences and generate the desired association rules. If the confidence of a 2-itemset is larger than the specific minimum confidence, we store them into the *ass_rule* table (as showed in Table 4).

In the third phase, when Squid receives a request from its user, it will record the request in the *access.log* file while the socket closes. So we monitor this file, and if some log record has been appended, we use this log record as our candidate item. Then we scan the Informix table to decide whether the candidate item exists in the antecedent list.

Because the same antecedent may have several descendants, we have to count out the corresponding descendants with the largest frequency as our pre-fetch item. Then the mechanism will work like a client to make a request to Squid with the descendant.

In the implementation detail, in order to find out the descendant with the largest frequency in a short time, we deal with the "ass_rule" table first. The step is to count out all antecedent and descendant pairs with largest frequency and store them in the *ass_rule_max* table (as showed in Table 5). This step can save a lot of time when we have a lot of association rules which all satisfy the minimum confidence.

We also implement some functionalities which can make our pre-fetch mechanism more powerful and efficient. First, we design a configuration file which records the setup information of the mechanism. The minimum support and minimum confidence can be setup and tuned in this file. Second, our mechanism can learn more rules automatically after setting the duration of the maintenance time.

3. PERFORMANCE

In order to verify the feasibility of our pre-fetch mechanism, we design two experiments to test the Squid combined with our pre-fetch mechanism.

We use three testing data sets for measurement. Experiment 1 measures the cache hit ratios based on different confidence settings, while Experiment 2 measures the hit rates of cache based on different supports in order to find out the proper setting for our mechanism. Then we compare the hit rates of our pre-fetch mechanism with those of the original Squid proxy server.

Our testing data is collected from three proxy servers which are used by different groups of users at National Chung-Hsing University. The proxy server 1 is in the Database Laboratory, Institute of Computer Science. The proxy server 2 is in the Institute of Computer Science. And the users' group of the proxy server 3 is in the National Chung-Hsing University. Table 6 lists the information with regard to our testing data.

We use the following two training data sets to generate association rules.

- **Training Data 1**

Collecting time:	1998/4/1 ~ 1998/4/8
Data source:	Proxy Server 1
Data size:	9450 tuples

- **Training Data 2**

Collecting time:	1998/4/1 ~ 1998/4/8
Data source:	Proxy Server 2
Data size:	5837 tuples

3.1 Experiment 1: Different Confidences

In this experiment, we use three different minimum confidence values with the same support value to derive six sets of association rules from the two training data sets. Then we use the six rule sets in our pre-fetch mechanism to test the testing data. We simulate the actions of our pre-fetch mechanism running with the Squid proxy server. That means if the descendant of some requested object is not in the cache of the proxy server, it is pre-fetched by the mechanism. When a pre-fetch is made, the cache hit number will be increased by one if the descendant is used in the following requests. We count the total cache hit number no matter what the requested objects are in the cache or pre-fetched by our mechanism, and the results are showed in Table 8.

From Table 8, we observe that when we change the minimum confidence value, the hit ratio is affected slightly. That means the association rules with lower confidences are not used very often. The association rules derived from different training data will have different performance impacts on different testing data. Generally speaking, training data and testing data belonging to the same access domain will result in a higher hit ratio. As showed in Table 8, the performance of the shadowed parts (indicating the same access domain) in each column is better than that of non-shadowed parts in the same column. Further measurement is the usage of the rule sets. This usage ratio indicates the proportion of association rules that have been used. The results are showed in Table 9.

For a higher confidence, the usage rate of our association rules becomes lower. That means some rules with lower confidences are still used sometimes.

It may be realized from this experiment that a higher confidence can not save much database space for storing rules and may lead to a lower cache hit ratio and a lower rule usage ratio.

3.2 Experiment 2: Different Supports

In this experiment, we use three different minimum support values to derive six sets of rules from the two training data sets. It is obvious from Table 10 that, a higher support results in a lower number of association rules in a rule set. And the testing results are showed in Table 11. In Table 11, although the number of rules decreases a lot, its cache hit ratio is not affected as much as we might expect.

Further, we measure the rule usage ratio with different rule sets. The rule usage ratio with a high support is higher than that with a low support. The results in Table 11 and Table 12 point out one thing: the rule generation procedure using a higher support can find out more accurate association rules. If we want to increase the rule usage of our pre-fetch mechanism, we can set a higher support. This can save database space and rule searching time.

According to the above two experimental results, when we use the pre-fetch mechanism, we should use proper

Column Name	Type	Length(bytes)
transaction_id	Integer	4
ip_address	Character	15
time_stamp	Float	8
request_url	Character	128
content_type	Character	9

Table 1: Schema of the access_log Table

Column Name	Type	Length(bytes)
request_url	Character	128
frequency	Integer	4

Table 2: Schema of the first_frequency Table

Column Name	Type	Length(bytes)
antecedent	Character	128
descendant	Character	128
frequency	Integer	4

Table 3: Schema of the pre_rule Table

Column Name	Type	Length(bytes)
antecedent	Character	128
descendant	Character	128
frequency	Integer	4

Table 4: Schema of the ass_rule Table

Column Name	Type	Length(bytes)
antecedent	Character	128
descendant	Character	128
frequency	Integer	4

Table 5: Schema of the ass_rule_max Table

	Proxy Server	**Collecting Time**	**Data Size**	**Hit Ratio**
Testing Data 1	Proxy Server 1	1998/4/8 ~ 1998/4/15	2273 tuples	8.34%
Testing Data 2	Proxy Server 2	1998/4/8 ~ 1998/4/15	2362 tuples	17.44%
Testing Data 3	Proxy Server 3	1998/4/15	78226 tuples	17.23%

Table 6: Testing Data from Access Log

	Data Source	**<support, confidence>**	**# Association Rules**
Rule Set 1	Training Data 1	<5, 0>	2273
Rule Set 2	Training Data 1	<5, 0.25>	1979
Rule Set 3	Training Data 1	<5, 0.5>	1770
Rule Set 4	Training Data 2	<5, 0>	500
Rule Set 5	Training Data 2	<5, 0.25>	434
Rule Set 6	Training Data 2	<5, 0.5>	378

Table 7: Rule Sets with Different Confidences

	Testing Data 1	**Testing Data 2**	**Testing Data 3**
Rule Set 1	11.31%	22.78%	18.01%
Rule Set 2	11.20%	22.78%	18.01%
Rule Set 3	11.20%	22.78%	18.01%
Rule Set 4	11.20%	22.82%	18.02%
Rule Set 5	11.20%	22.82%	18.02%
Rule Set 6	11.20%	22.82%	17.81%

Table 8: Hit Ratio Comparison with Different Confidences

	Testing Data 1	Testing Data 2	Testing Data 3
Rule Set 1	1.76%	6.64%	10.38%
Rule Set 2	1.62%	6.32%	10.01%
Rule Set 3	1.58%	6.33%	9.83%
Rule Set 4	2.80%	34.00%	15.80%
Rule Set 5	2.76%	32.71%	14.74%
Rule Set 6	2.91%	30.95%	15.08%

Table 9: Rule Usage Ratios with Different Confidences

	Data Source	<support, confidence>	# Association Rules
Rule Set 7	Training Data 1	<5, 0.5>	1770
Rule Set 8	Training Data 1	<6, 0.5>	411
Rule Set 9	Training Data 1	<7, 0.5>	199
Rule Set 10	Training Data 2	<5, 0.5>	378
Rule Set 11	Training Data 2	<6, 0.5>	300
Rule Set 12	Training Data 2	<7, 0.5>	296

Table 10: Rule Sets with Different Supports

	Testing Data 1	Testing Data 2	Testing Data 3
Rule Set 7	11.20%	22.78%	18.01%
Rule Set 8	11.20%	22.61%	17.81%
Rule Set 9	10.76%	22.26%	17.76%
Rule Set 10	11.20%	22.82%	17.81%
Rule Set 11	11.20%	22.69%	17.80%
Rule Set 12	10.43%	22.65%	17.80%

Table 11: Hit Ratio Comparison with Different Supports

	Testing Data 1	Testing Data 2	Testing Data 3
Rule Set 7	1.58%	6.33%	9.83%
Rule Set 8	5.11%	13.62%	29.93%
Rule Set 9	8.04%	14.57%	28.64%
Rule Set 10	2.91%	30.95%	15.08%
Rule Set 11	3.67%	33.00%	11.33%
Rule Set 12	3.04%	32.77%	11.49%

Table 12: Rule Usage Ratios with Different Supports

	Testing Data 1	Testing Data 2	Testing Data 3
Rule Set 1	11.31%	22.78%	18.01%
Rule Set 4	11.20%	22.82%	18.02%
Original Proxy Server	8.34%	17.44%	17.24%

Table 13: Hit Ratio Comparison

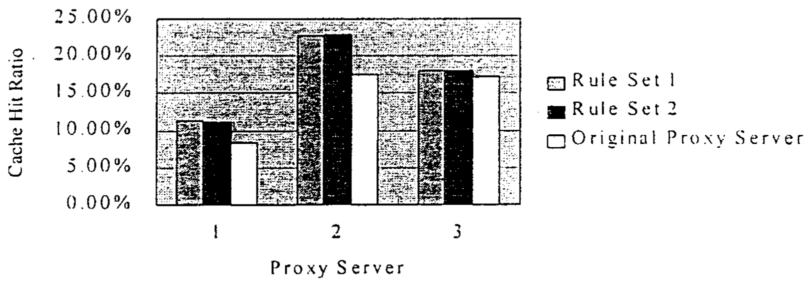

Figure 2: Hit Ratio Comparison with Original Squid

training data to derive association rules. As fewer rules are stored in the database, we may set a lower confidence and lower support in order to obtain a good cache hit ratio. As the number of rules grows, we should set a medium confidence and higher support to get a better rule usage ratio and to spend less time in searching rules.

	Proxy Server 1	Proxy Server 2	Proxy Server 3
Rule Set 1	35.49%	30.56%	4.47%
Rule Set 4	34.22%	30.85%	4.52%

Table 14: Increasing Rate of Cache Hit Ratio

	Proxy Server	Collecting Time	Data Size	Hit Ratio
Testing Data 4	Proxy Server 1	1998/4/15 ~ 1998/4/22	10014 tuples	23.66%
Testing Data 5	Proxy Server 2	1998/4/15 ~ 1998/4/22	5358 tuples	16.14%

Table 15: New Testing Data Sets

	Testing Data 1	Testing Data 2
Rule Set 1 (from Training Data 1)	11.31%	22.78%
Rule Set 4 (from Training Data 2)	11.20%	22.82%
Squid	8.34%	17.44%
	Testing Data 4	**Testing Data 5**
Rule Set 13 (from Training Data 3)	34.99%	17.71%
Rule Set 14 (from Training Data 4)	24.20%	17.79%
Squid	23.66%	16.14%

Table 16: Cache Hit Ratios with Different Training Data

	Testing Data 1	Testing Data 2
Rule Set 1 (from Training Data 1)	1.76%	6.64%
Rule Set 4 (from Training Data 2)	2.80%	34.00%
	Testing Data 4	**Testing Data 5**
Rule Set 13 (from Training Data 3)	4.39%	1.54%
Rule Set 14 (from Training Data 4)	10.83%	16.39%

Table 17: Rule Usage Ratios with Different Training Data

3.3 Comparison with Original Squid

We choose the results from Rule Set 1 and Rule Set 4, which have the better performance in our experiments, to compare with the cache hit ratio of the original Squid. The results are showed in Table 13, which are derived from Table 6 and Table 8.

From Table 14, we observe that the cache hit ratios of the proxy server combined with our mechanism is clearly higher than those of the original proxy server. This indeed shows that our mechanism can increase the cache hit ratio of the proxy server. We may show this result more clearly in Figure 2.

The increasing rate of cache hit ratios is computed in Table 14. For Proxy Servers 1 and 2, our mechanism can increase the hit ratio about 30%. In the case of Proxy Server 3, it only increases the ratio about 5%. The reason is the users' domains of these three proxy servers are quite different. Proxy Server 3 has a user group much larger than the other two, and its users' interests are so diverse that the cache object reuse rate is low. From these results, we know that the training data for association rules play an important role if we want to have a good performance pre-fetch mechanism. If the proxy server users have the same access domain as the training data, we can derive more accurate association rules and have better cache hit ratios.

3.4 Performance of Rule Maintenance

In this section, we use Training Data 3 and Training Data 4 to test the rule maintenance function of our mechanism. From them we obtain two new sets of rules: Rule Set 13 and Rule Set 14, whose numbers of rules are 5548 and 720, respectively.

- **Training Data 3**

Collecting time:	1998/4/1 ~ 1998/4/15
Data source:	Proxy Server 1
Data size:	10361 tuples

- **Training Data 4**

Collecting time:	1998/4/1 ~ 1998/4/15
Data source:	Proxy Server 2
Data size:	8200 tuples

We also use two new testing data sets to test the new rule sets. They are Testing Data 4 and Testing Data 5, collected from April 15 to April 22. And the testing results are showed in Table 16.

From the Database Laboratory cases in Table 16 (the marked areas in the second column), we can observe that after training more rules, the hit ratio increases. The increasing rate is about 47.93%. This indicates that our mechanism makes Squid more powerful after working with it for a period of time.

But in the cases of Institute of Computer Science (the marked areas in the third column), training more rules does not increase the cache hit ratio. This is because the collecting time includes the spring vacation, and fewer users used this proxy server. From this, we can also observe that the number of users and their using time may influence the cache hit ratio.

Further, we compare the rule usage ratios. The results are showed in Table 17. The Database Laboratory cases also show the improvement. From this, we can know that, after training more rules, more accurate rules can be found. If we also change the minimum support and minimum confidence when the number of rules increases, we can obtain a better rule usage ratio.

4. CONCLUSION

In order to alleviate the problem of long waiting time when we access the WWW information through the proxy server, we propose a pre-fetch mechanism which can increase the cache hit ratio of the proxy server. This mechanism is a combination of the technique of mining association rules and Squid proxy server. When a user accesses a WWW object, the pre-fetch mechanism will use relevant association rules to predict the object to be accessed next and pre-fetch it from the Internet. In order to implement this mechanism, we develop a mining algorithm in accordance with the characteristics of the HTTP 1.0 protocol. And we use an Informix database system to handle the mining information.

To show the feasibility of our pre-fetch mechanism, we perform experiments to measure it with several testing access logs. From the experiments, we observe that our mechanism can increase about 30% of the cache hit ratio in the proxy server and hence reduce the user's waiting time effectively. Furthermore, it can increase about 47% of the cache hit ratio after learning more rules.

The design of our mechanism is independent of the Squid proxy server, so it is easy to use this mechanism in other existing Squid proxy servers without any changes or recompiling. If the mechanism is removed from Squid, Squid can still work like it used to be. This design hence can increase the portability of our mechanism.

This research work can be extended in several ways. First, the Informix database system plays an important role in our mechanism. But if the cost of the database system is a major concern, we may want to use a file system instead. Second, we may use association rules to implement the cache replacement algorithm in the proxy server. Squid uses the LRU algorithm to decide which expiring cache objects should be swapped out when the cache is full. However, since association rules have more information than the LRU strategy in predicting which objects are frequently used and which objects will be used next, a better cache replacement strategy based on association rules can be expected.

5. REFERENCES

[1] R. Agrawal, T. Imielinski and A. Swami, "Mining association rules between sets of items in large database," *Proc. of ACM SIGMOD*, 1993, pp.207-216.

[2] R. Agrawal, T. Imielinski and A. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 6, 1993, pp.914-925.

[3] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning and T. Bollinger "The Quest data mining system," *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, 1996.

[4] R. Agrawal and R. Srikant "Fast algorithms for mining association rules," *Proc. of 1994 Int'l Conference on Very Large Data Bases*, September 1994, pp.487-499.

[5] R. Agrawal and R. Srikant, "Mining sequential patterns," *Proc. of the Int'l Conference on Data Engineering (ICDE)*, 1995, pp.3-14.

[6] J.-C. Chen, *Data mining of strong association rules in the world wide web*, Master thesis, Department of Computer Science, National Chung-Hsing University, Taiwan, R.O.C., 1996.

[7] A. Luotonen and K. Altis, "World-wide web proxies," *Proc. of the 1st Int'l. World Wide Web Conference*, 1994.

[8] J. S. Park, M. S. Chen and P. S. Yu, "An effective hash-based algorithm for mining association rules," *Proc. of ACM SIGMOD*, 1995, pp.175-186.

[9] J. S. Park, M. S. Chen and P. S. Yu, "Using a hash-base method with transaction trimming for mining association rules," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 5, 1997, pp.813-825.

[10] T. Rathburn, "Data-mining in the financial markets," *PC AI*, 1997, pp.19-20.

[11] R. Srikant and R. Agrawal, "Mining generalized association rules," *Proc. of the 21st VLDB Conference*, 1995.

[12] http://squid.nlanr.net/ - Proxy Server: Squid.