WWW Cache Layout to Ease Network Overload

- Analyzing Regularity of Structured Data -

Kenichi Yoshida
Advanced Research Laboratory, Hitachi, Ltd.

The GBI (graph-based induction) concept learning method is applied to extract typical access patterns of WWW data. By interpreting extracted patterns as the cache site layout, we can reduce the total network data flow by implementing a distributed cache system which is adapted to the WWW access patterns. Although the huge WWW data flow causes the overflow of the conventional hierarchical cache system, the layout created by the GBI method eases this problem. The traffic reduction ratio of this distributed cache system is 2.5 times higher than that of the conventional hierarchical cache system. Our results suggest the importance of the data analyzing methods which can handle structured data. By analyzing regularity in graph structures, the GBI method can reduce the network data flow. The statistical criteria contribute to the analysis of promising patterns.

1 Introduction

The WWW (world wide web) service has recently become a widely used network service which symbolizes the benefits of a networked society. By using the WWW, the user can access various types of information easily and quickly. However, a rapid growth in demand sometimes causes a heavy network overload, and the resulting slow-response spoils the benefits of the WWW. Statistics [WID95] clearly indicate the potential overload in the backbone of the Japanese wide area network. Thus, the demand for better tuning methods in wide area networks has been increasing.

To solve this problem, Chankhunthod et al. [CDN+96] propose the use of the hierarchical cache system, and Gwertzman and Seltzer [GS94] propose the distributed cache system. These systems try to use the redundancy of WWW data. By storing previously accessed data in the cache, they try to omit the later data transmission. If

Kenichi Yoshida e-mail:yoshida@hitachi.co.jp
Advanced Research Laboratory, Hitachi, Ltd.
Hatoyama, Saitama 350-03 Japan

the data required by the user is stored in the cache system, the WWW client can respond to the user's request quickly. In this case, the cache system on the network forms a kind of distributed file system.

However, tuning a distributed file system is a difficult problem. For example, Dowdy and Foster [DF82] report that most of the proposed methods for the distributed file system do not work well if the required conditions are not satisfied. Unfortunately, none of the known methods seem to match the current WWW situation. Although hierarchical cache systems such as [CDN+96] slightly speed up the WWW client, a decrease in the network load is not guaranteed. If the huge load shuts down the network, a cache system cannot solve the problem. The distributed cache system [GS94] also needs investigation. No known method can lay out the distributed cache storage automatically.

In this paper, we examine a method for laying out the distributed cache storage on the network using the GBI (graph-based induction [YM95]) concept learning method. By analyzing the access patterns of WWW data, this method can lay out distributed cache storage which is adapted to the access patterns. The next section describes the idea, and section 3 presents experimental results. Section 4 discusses some aspects of the GBI method.

2 Distributed Cache Layout

GBI was originally developed as a concept learning method [YM95]. In [YMI94], we examine its various learning functions, such as the decision tree learning function [BFOS84, Qui86] and the explanation-based learning function [MKKC86, DM86]. The key idea of GBI is the extraction of frequently appearing patterns from graph-format data, and the use of the extracted patterns for various purposes.

In this study we extract frequently appearing data transmissions from the wide area network data flow. The data transmission log was gathered using a WWW proxy server, and the log has a graph format which is suitable for analysis by the GBI method. By interpreting the extracted patterns as the suggestions for the cache positions, we can lay out distributed cache storage.

Figure 1 shows the idea. We assume that many WWW clients try to get data from relatively few well-known servers. In the actual situation, a number of duplicated data transmissions occur and cause the network overload.

Figure 1 (A) shows a situation where four WWW clients (T1~4) access one WWW server (S) through network routers (N1~3). In Figure 1, single edge stands for some amount of data transmission. The arrow shows the transmission direction. Although Figure 1 (A) shows a simplified situation, we can represent the actual wide area network situation using this graph format.

If the GBI program inputs the data shown in Figure 1 (A), it outputs frequently appearing patterns in the input. If the GBI program admits the hatched portion of Figure 1 (B) as a frequently appearing pattern in the input graph, it contracts the input graph into the graph shown in Figure 1 (C). Figure 1 (D) shows the interpretation of the result as the cache layout.

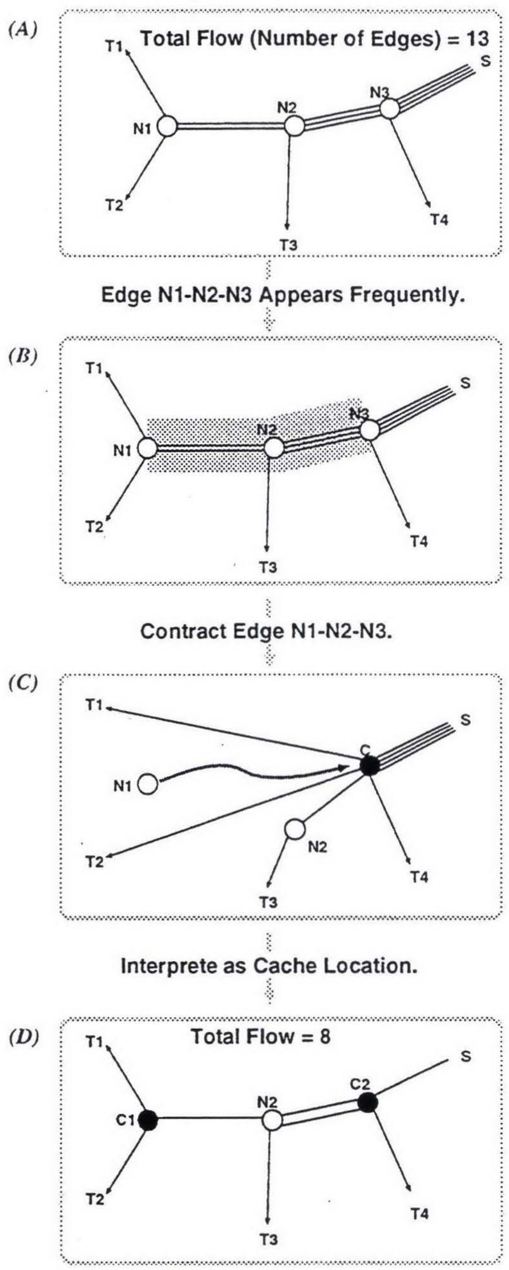

(A)
T1
Total Flow (Number of Edges) = 13
S
N1
N2
N3
T2
T3
T4
Edge N1-N2-N3 Appears Frequently.
(B)
Contract Edge N1-N2-N3.
(C)
C
Interprete as Cache Location.
(D)
Total Flow = 8
C1
C2

Fig. 1: Distributed Cache Layout by GBI method

In Figure 1 (D), two caches (C1 and C2) for a WWW server S are laid out, and the estimated network flow is reduced from 13 to 8. In the actual situation, the input graph for the GBI program involves many edges each of which corresponds to a data flow from various WWW servers, and the above process should be done simultaneously. However, the GBI algorithm can extract important (i.e., frequent) data transmission patterns according to the importance of the transmission. See section 4.1 for the details of the algorithm.

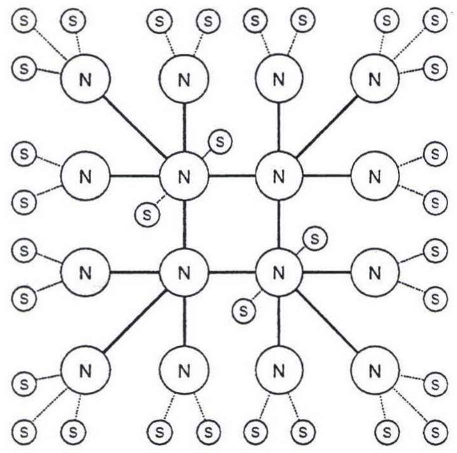


Fig. 2: Network Configuration for Simulation

3 Experimental Results

We performed simulation studies to compare the caching methods. Figure 2 shows the network configuration assumed in the simulations. The circles with an N are network routers which can also act as WWW caching servers. The circles with an S are WWW servers and clients. We collected the names of these WWW servers from the log (see later), and the number of WWW servers is about 32,000. We also assume 16 WWW clients. Each network router is connected with one WWW client and multiple WWW servers. The connections are randomly decided before the simulation. Each WWW client and router has a 32-Mbyte caching capacity.

To generate input data for simulations, we use a WWW access log which was recorded at our WWW proxy server. The log includes 2.3 million data transfers (18.7 Gbytes in size) over 16 days. In the simulations, WWW clients are assumed to access data as is specified in the log. The log is divided into 16 parts. The log of the first day is assumed to be the data requests from the first WWW client. The log of the second day is assumed to be the requests from the second WWW client. We assume 16 clients in total, and the requests for the other clients are generated in a similar manner. All the data transmissions in the log are treated as the data requests of the same date with the time of day retained.

Note that we use other log data to lay out distributed cache storage. The log which was recorded in the preceding 2 days was used to lay out cache storage by the GBI method. By using the log of the succeeding 16 days for simulations, we ensure the independence of the test set (i.e., for simulation) and the training set (i.e., for layout).

Figure 3 shows the simulated traffic on the network. The solid line shows the traffic with a distributed cache whose layout is designed by GBI. ◇ marks show the traffic without a cache. X axis shows the time of day (unit: minutes) and Y axis shows the amount of data transfer per 10 minutes (unit: M byte).

If we consider the difference between the amount of the data flow in the simulation and that in the actual situation, 32 Mbytes in the simulation corresponds to 100 Mbytes $\sim$ 4 Gbytes in the actual situation. See Appendix A.

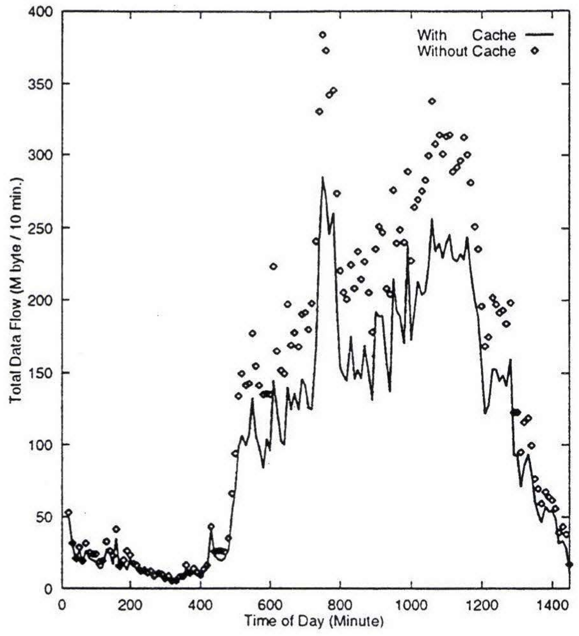


Fig. 3: Decrease of Network Data Flow

Figure 3 shows that the WWW traffic starts to increase from around 9 AM (60×9=540 minutes), and reaches its peak at noon. There is little traffic at midnight. When we use the cache system, the average traffic between 10 AM and 8 PM is 186 Mbytes per 10 minutes. Since the average traffic without a cache is 249 Mbytes, the decrease rate is 26%. The most important point here is the traffic reduction at the peak time, and our cache system was able to achieve a 100-Mbyte data reduction at the peak time (see noon time in Figure 3).

Figure 4 compares the distributed cache system using GBI (solid line) and the conventional hierarchical cache system (dotted line). Both methods use local cache systems with the same storage size (32 Mbytes) simultaneously. Additional traffic reductions are shown. The distributed cache system using GBI was able to reduce

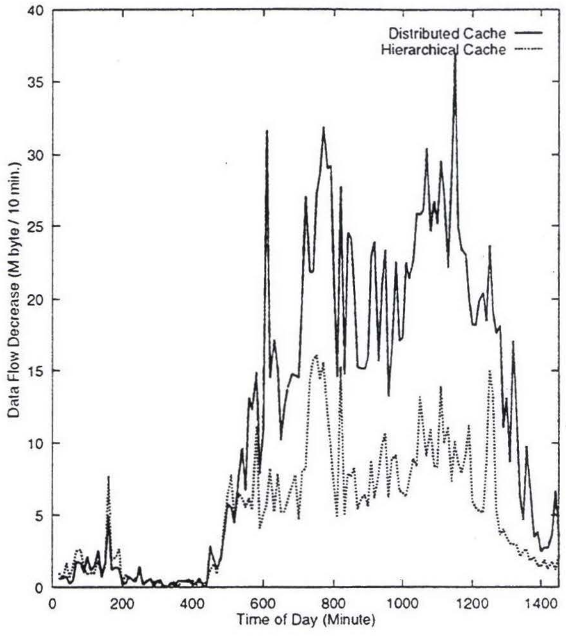


Fig. 4: Comparison of Cache Systems

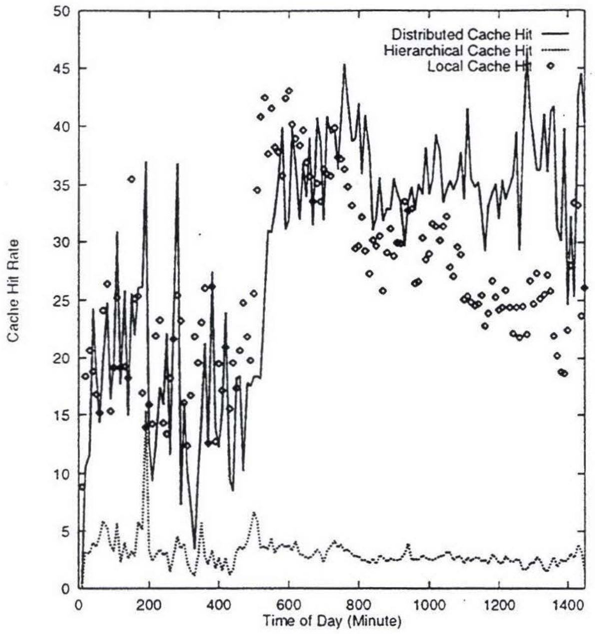


Fig. 5: Comparison of Cache Hit Rate

traffic by 21.7 Mbytes/10 minutes (average between 10 AM and 8 PM). Since the hierarchical cache system was able to reduce traffic by 8.8 Mbytes/10 minutes, the distributed cache system with GBI is 2.5 times more efficient than the hierarchical cache system.

The advantage of the distributed cache system with GBI comes from the fact that the huge data flow causes the overflow of the hierarchical cache system. Although the hierarchical cache system tries to memorize the data, the huge data flow prevents the system from keeping the data until the data is used again. The distributed cache system with GBI avoids this problem by sharing the role.

To confirm the above ideas, we examined several data. Figure 5 shows the cache hit rate of each caching method. Here, the cache hit rate is calculated as follows:

$$\text{Hit Rate} = 100\times \frac{\text{Number of Files Reused}}{\text{Number of Files Stored in the Cache}}$$

The hit rate of the local cache system is 30.9 (Figure 5, ◇) and that of the distributed cache system with GBI is 35.9 (solid line). However, it is 2.8 for the hierarchical cache system (dotted line).

Figure 6 shows the total data size in each cache system. The hierarchical cache system (dotted line) stores more data than the local cache system (◇) and the distributed cache system with GBI (solid line). We interpret these results as meaning that the hierarchical cache system stores too much data and reduces its efficiency (hit rate).

4 Discussion

The key idea of GBI is the extraction of frequently appearing patterns from graph format data, and the use of the extracted

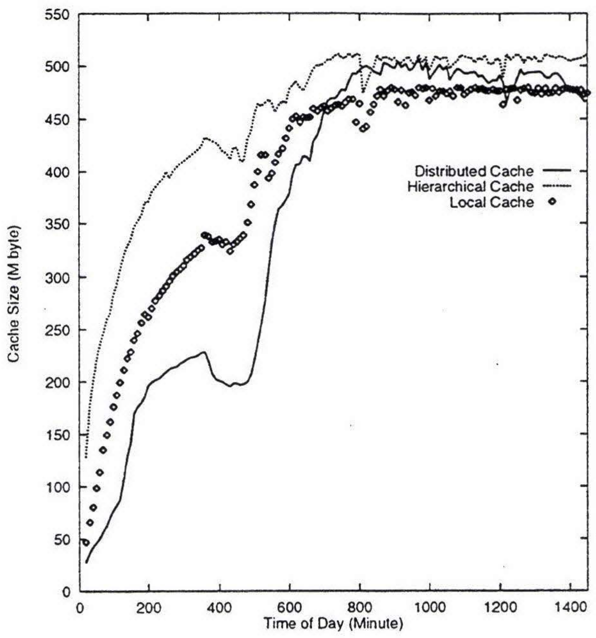
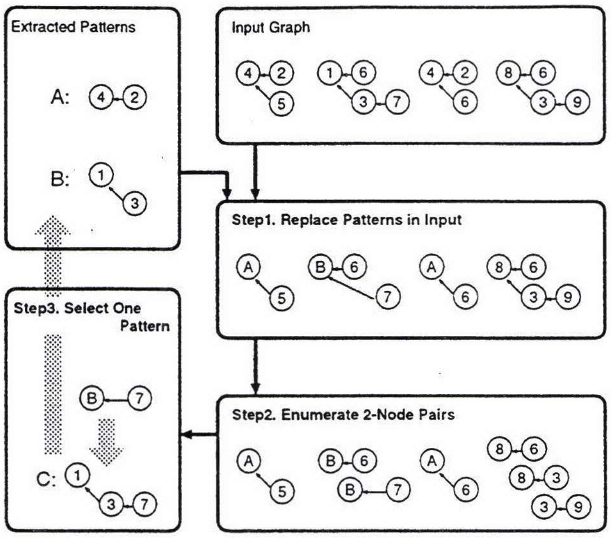


Fig. 6: Stored Data In Cache



Extracted Patterns
A:
B:
Input Graph
Step1. Replace Patterns in Input
Step2. Enumerate 2-Node Pairs
Step3. Select One Pattern
C:

Fig. 7: GBI as Pattern Extraction Algorithm

patterns for various purposes. In [YM96], we describe the use of the statistical criteria to extract patterns. In this section, we first describe the GBI algorithm together with the correspondence between the graph pattern extraction and the decision tree learning. By understanding this correspondence, we can use various statistical techniques (e.g., gini index, cross validation [BFOS84]) to find important patterns in the graph. Later, we examine the various learning functions of GBI.

4.1 GBI algorithm

Figure 7 displays the graph extraction algorithm of GBI. Suppose that GBI already extracted two patterns (i.e., A and B). This algorithm proceeds as follows:

1. First it replaces each occurrence of the extracted patterns in the input graph with a single node. In Figure 7, two occurrences of pattern A ($4 \leftarrow 2$) are replaced with a single node (A), and one occurrence of pattern B ($1 \leftarrow 3$) is also replaced.
2. Next, it enumerates all two-node pairs in the graph.
3. Then it selects one pair. If one or both of the nodes in the selected pair is replaced in step 1, restore the node to its original pattern. In Figure 7, pattern $B \leftarrow 7$ is selected and its original pattern $1 \leftarrow 3 \leftarrow 7$ is stored as a newly extracted pattern.

By repeating these steps, the GBI algorithm extracts patterns from the input graph. Here the criteria to select a pair in step 3 is important. To explain this criteria, Figure 8 shows the decision tree learning aspect of the same GBI algorithm. As a

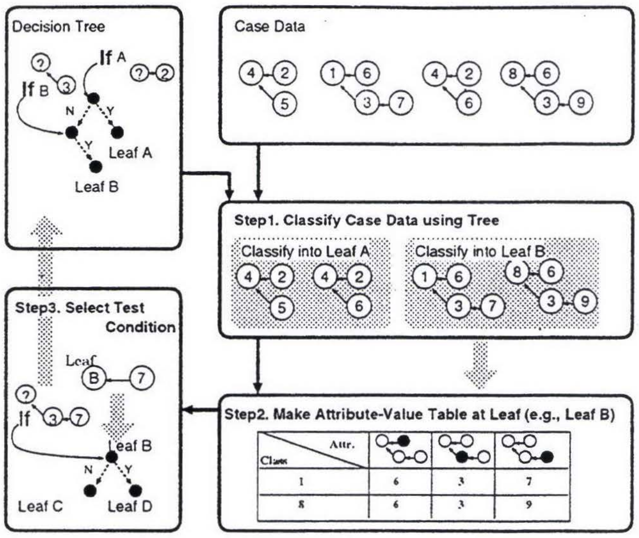

Decision Tree
If A
If B
Leaf A
Leaf B
Case Data
Step1. Classify Case Data using Tree
Classify into Leaf A
Classify into Leaf B
Step2. Make Attribute-Value Table at Leaf (e.g., Leaf B)
Attr.
Class
Step3. Select Test Condition
Leaf
Leaf C
Leaf D

Fig. 8: GBI as Decision Tree Learning Algorithm

decision tree learning algorithm, GBI proceeds as follows:

1. First, it classifies input data using the decision tree under construction. Here, we use extracted patterns as a decision tree whose test conditions correspond to extracted patterns. We also interpret the input graph so that each subgraph corresponds to case data. The class information is stored as the root node color (i.e., the label of the root node) of the subgraph, and the attribute value is stored as the leaf node color.

2. Then it generates standard attribute-value tables at each leaf node of the decision tree. To do this, use the root node color of the subgraph as the class information (i.e., the row of the table), and use the leaf node position as the attribute name (i.e., the column of the table).
Here, GBI only considers the leaf nodes having a direct connection with root nodes. If the node in the pair is the replaced node in step 1 (see step 1 of Figure 7), the generated table will have extended attributes (i.e., new columns).
For example, the right-most column of the table in Figure 8 is the attribute which is extended by replacing each occurrence of pattern B with a single node. This attribute (i.e., the right-most column) is included in the table because the replacement in step 1 directly connects the corresponding node with a root node.

3. Next it selects a new test condition from the tables using statistical criteria, such as the gini index [BFOS84], information gain [Qui86], and adds it to the decision tree.

The above process essentially follows the standard decision tree learning framework except that GBI dynamically changes the attribute-value table which is fixed in the conventional decision tree learning process. Thus we can admit the pattern extraction process as an extension of the decision tree learning process.

Note that the root node color is used in Figure 7 and is ignored in Figure 8. This difference affects the learning function of GBI. See Section 4.3 for a discussion about this.

```
Algorithm Induction
  Variable    $G_{in}$: Case Data
              $T$ : Decision Tree
  Begin
    $T \leftarrow \emptyset$
    Repeat
      Classify All Case Data $G_{in}$ Using $T$
      Make Attribute Table by Proc. 1, 2, 3
      Select New Test Condition, and Add it to $T$
  End
Proc. 1 Conventional Method:
  Always Use Same Attribute Set.
Proc. 2 GBI Method:
  Select New Attributes from Graph.
Proc. 3 ILP Method:
  Select New Attributes from First-Order Logic.
```

Fig. 9: Comparison of Induction Algorithms

4.2 GBI as Induction Algorithm

The decision tree learning method is one of the well-known inductive learning methods. According to the previous section, GBI can be seen as an extension of the standard induction method. The statistical methods, such as linear discrimination and k-nearest-neighbor [Jam84], and conventional inductive learning methods, such as [BFOS84, Qui86], use attribute-value tables as the data representation language. Inductive logic programming (ILP [Sha83, Qui90, PK92, MF92]) is also an extension of the conventional inductive learning method. ILP uses first-order logic.

Figure 9 compares these methods. Although they all share some common aspects, the data representation language used is different. The GBI method uses graphs as its data representation language. Its level of expressiveness lies between that of tables and first-order logic. Thus its learning potential is weaker than that of ILP, but stronger than that of the conventional methods.

Using a least-expressive data representation language for learning sometimes gives better learning performance by making the potential search space smaller. In [YM96], we examined one such domain; that is, we tried to model relationships between user tasks. The enhanced expressiveness of graphs enables the analysis of the relationship between user tasks. See [YM96] for details.

4.3 Various Learning Functions of GBI

Another important point is the various functions of the GBI method. In [YMI94], we examine 1) its macro rule learning function, and 2) its concept finding function together with the decision tree learning function. By considering the correspondence shown in Figures 7 and 8, we can apply the various methods developed in the decision tree learning studies to the macro rule learning and to the concept finding. The unified view of various learning functions may accelerate research in these fields.

Note that the root node color is used in Figure 7 and is ignored in Figure 8. In [YMI94], we use root node color in the process of macro rule learning and concept finding, but ignore it in the process of decision tree learning. In the framework of GBI, its learning function is realized by finding typical data patterns. Since the root node color of the new case data (i.e., class information) is not known, GBI can not use this information for a matching process. However, the essence of the decision tree learning function of GBI is the finding of typical class and attribute pairs. The essence of the macro rule learning function of GBI is the finding of typical rule combinations, and the essence of the concept finding is typical data co-occurrences.

5 Conclusion

By extracting typical access patterns of WWW data, we can reduce the total network data flow by implementing a distributed cache system which is adapted to the WWW access patterns. Although the huge WWW data flow causes the overflow of the conventional hierarchical cache system, the layout created by our method eases this problem.

The results suggest the importance of the data analyzing methods which can handle structured data. By extracting regularity in graph structures, the GBI method can reduce the network data flow. The statistical criteria contribute to the extraction of promising patterns. We also propose a unified view of various learning functions. This unified view may accelerate research in related fields by enabling the sharing of findings among these fields.

References

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[CDN+96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrll. A hierarchical internet object cache. In *USENIX96*, p. 153~163, 1996.

[DF82] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM computing surveys*, Vol. 14, No. 2, p. 287~314, 1982.

[DM86] G. DeJong and R. Mooney. Explanation-Based Learning: An Alternative View. *Machine Learning*, pp. 145–176, 1986.

[GS94] J. Gwertzman and M. Seltzer. The case for geographical push-caching. In *HotOS Conference*, pp. ftp: //das-ftp.harvard.edu /techreports /tr-34-94.ps.gz, 1994.

[Jam84] M. James. *Classification Algorithms.* A Wiely-Interscience Publication, 1984.

[MF92] S. Muggleton and C. Feng. Efficient Induction of Logic Programs. In S. Muggleton, editor, *Inductive Logic Programming*, pp. 281–298. Academic Press, 1992.

[MKKC86] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-Based Generalization: A Unifying View. *Machine Learning*, pp. 47–80, 1986.

[PK92] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, Vol. 9, pp. 57–94, 1992.

[Qui86] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, Vol. 1, pp. 81–106, 1986.

[Qui90] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, Vol. 5, pp. 239–266, 1990.

[Sha83] E.Y. Shapiro. *Algorithmic Program Debugging.* MIT Press, 1983.

[WID95] 1994 research report of WIDE project (in Japanese), 1995.

[YM95] K. Yoshida and H. Motoda. CLIP: concept learning from inference patterns. *Artificial Intelligence*, Vol. 75, pp. 63–92, 1995..

[YM96] K. Yoshida and H. Motoda. Automated User Modeling for Intelligent Interface. *International Journal of Human Computer Interaction*, Vol. 8, No. 3, pp. 237–258, 1996.

[YMI94] K. Yoshida, H. Motoda, and N. Indurkhya. Graph-based Induction as a Unified Learning Framework. *Applied Intelligence*, Vol. 4, pp. 297–328, 1994.

APPENDIX A

To make the proposed method practical, we should confirm the adequacy of the assumptions used in the simulation. In Appendix A, we examine some parameters which are used based on the assumptions.

The most important parameter used in the simulation is the cache storage size (i.e., 32 Mbytes). This number would have been too small if we had used it in the actual environment. However, if we consider the difference between the amount of the data flow in the simulation and that in the actual situation, 32 Mbytes in the simulation corresponds to 100 Mbytes $\sim$ 4 Gbytes in the actual situation. In the simulation, the data flow in the virtual network line was 26 Kbytes/sec. The data flow capacity of the T3 line which is used in the wide area network is about 5.5 Mbytes/sec. We assume 60% of the capacity is used for WWW data. If the cache size is in proportion to the data flow, 32 Mbytes in the simulation corresponds to 4 Gbytes ($= 32Mbytes \times \frac{5.5Mbytes \times 0.6}{26Kbytes}$) in the actual situation. We believe that the use of the T3 line and the 60% WWW load is an adequate assumption based on [WID95].

Another important assumption is the number of network routers (i.e., 16). In the actual network, an enormous number of routers is used. However, we believe the analysis of domain level routers, i.e., the main routers of large organization, is enough for our purposes. We also believe that a single analysis per month is enough to obtain the network flow tendency. The layout by the GBI method took 5 minutes on a personal computer for a network with 16 routers. Even if we assume a network with 1000 routers, the estimated CPU requirement is about 5 hr/month. Thus we believe that the required computing resource for the GBI layout is feasible.

APPENDIX B

In Appendix B, we show some additional results of simulation studies. Figures 10 and 11 show the cache hit rate of the hierarchical cache system and the distributed cache system with GBI. Both figures show the average hit rate of 1) the local caches, 2) the four center caches, and 3) the other surrounding 12 caches.

In the hierarchical cache system, only the local cache systems work. In the distributed cache system, the hit rate of the four center caches is higher than that of the local caches. The decrease of the hit rate for the 12 surrounding caches is caused by

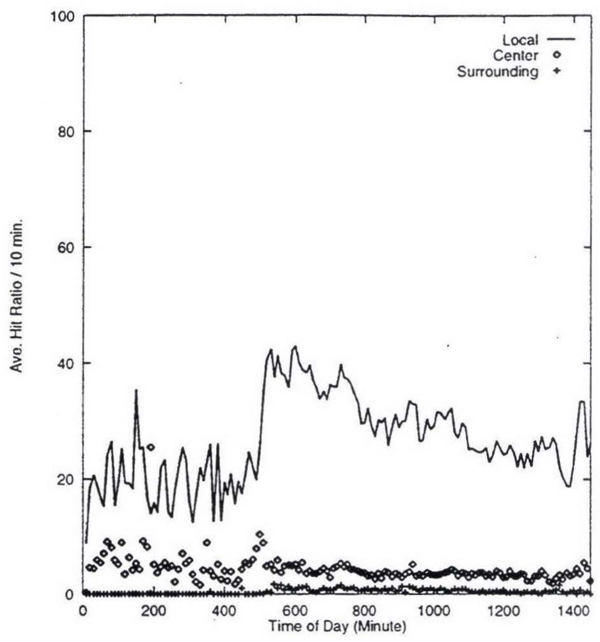


Fig. 10: Hit Rate of Hierarchical Cache

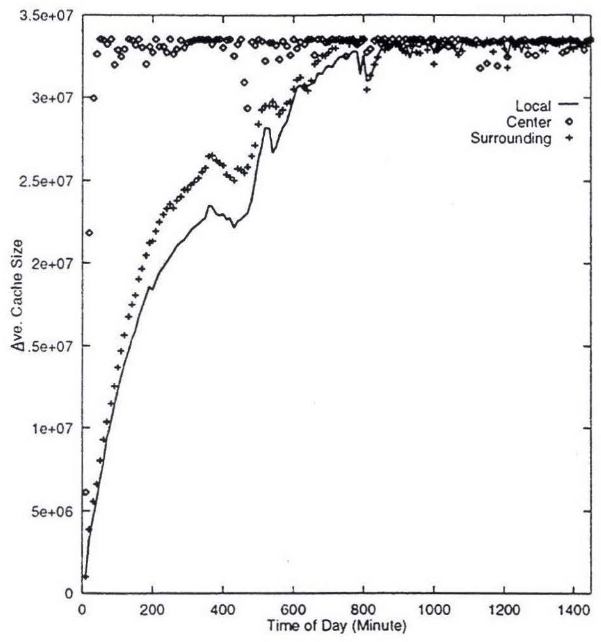


Fig. 12: Stored Data in Hierarchical Cache

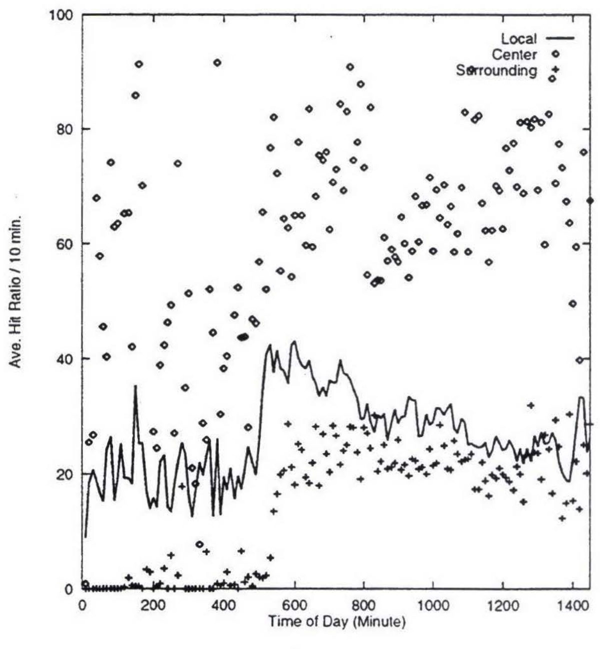


Fig. 11: Hit Rate of Distributed Cache with GBI

the local caches. Since the duplication in the data is removed by the local cache, the use of these surrounding caches is difficult even in our methods. However, its hit rate is close to that of the local cache.

Figures 12 and 13 show the size of stored data in the cache systems. The hierarchical cache system (Figure 12) stores the data very quickly, and reaches its maximum capacity within 30 minutes from the starting point of the simulation. Note that the first 30-minute period is a relatively low-load period. This seems to imply the inefficiency of the hierarchical cache system.

The distributed cache system with GBI (Figure 13) shows a different trend. This system reaches its maximum capacity at noon, i.e, the maximum load period. After the heavy peak load at noon time, a little space is left over. Note that the hit rate of the local cache system keeps decreasing

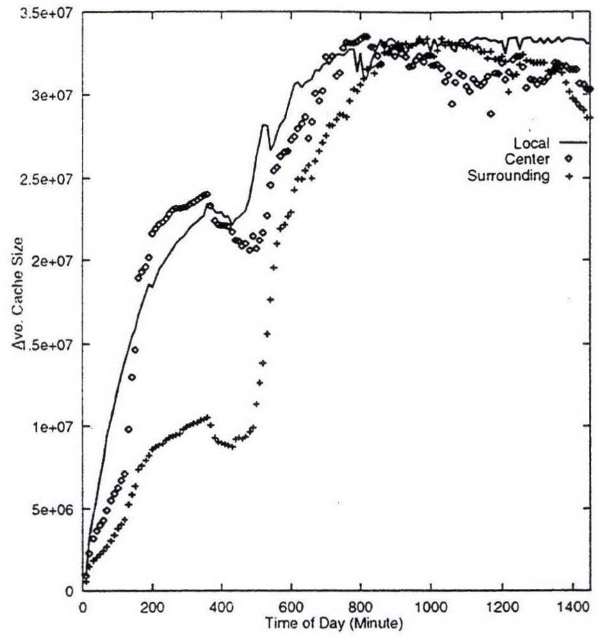


Fig. 13: Stored Data in Distributed Cache

during the simulation (See Figure 5). However, the distributed cache system does not show this tendency (see the same Figure). A little space left over in the distributed cache system contributes to this difference. By reducing the data stored in the cache, the distributed cache system achieves high durability against the cache overflow.

Note that the trend of data size in the center of the distributed cache (Figure 13, ◇) and that in the surrounding area of the hierarchical cache (Figure 12, +) are similar. However, they are different in the hit rate (Figure 11◇, Figure 10+). This also shows the effect of the local cache which decreases the duplication in the data. If we do not consider the mutual effects of cache systems, the mutual effects of the cache systems decrease their overall performance.

These results reveal the limitation of the conventional approach with the hierarchical cache system. The cache storages which construct the hierarchy tend to affect each other, and the resulting redundancy decreases their overall performance. The distributed cache system can avoid this problem with the help of the layout created by the GBI method.

Author Index