Mining Objects Correlations to Improve Interactive Virtual Reality Latency

Shao-Shin Hung[1] , Hsing-Jen Chen[2] and Damon Shing-Min Liu[3]

Department of Computer Science and Information Engineering,
National Chung Cheng University

Chiayi, Taiwan 621, Republic of China
{hss[1], damon[3]}@cs.ccu.edu.tw; staren[2]@gmail.com

Abstract

Object correlations are common semantic patterns in virtual reality systems. They can be exploited for improving the effectiveness of storage caching, prefecthing, data layout, and minimization of query-response times. Unfortunately, this information about object correlations is unavailable at the storage system level. Previous approaches for reducing I/O access time are seldom investigated. On the other side, data mining techniques extract implicit, previously unknown and potentially useful information from the databases. This paper proposes a class of novel and efficient pattern-growth method for mining various frequent sequential traversal patterns in the virtual reality. Our pattern-growth method adopts a divide-and-conquer approach to decompose both the mining tasks and the databases. Moreover, our efficient data structures are proposed to avoid expensive, repeated database scans. The frequent sequential traversal patterns are used to predict the user navigation behavior and help to reduce disk access time with proper placement patterns into disk blocks. We also define the terminologies such as paths, views and objects used in the model. We have done extensive experiments to demonstrate how these proposed techniques not only significantly cut down disk access time, but also enhance the accuracy of data prefetching.

Keywords: Virtual reality, frequent pattern mining, clustering, object correlations, data layout

1. Introduction

With the advent of advanced computer hardware and software technologies, virtual reality (VR)are becoming larger and more complicated. To satisfy the growing demanding for fidelity, there is a need for interactive and intelligent schemes that assist and enable effective and efficient storage management. Unfortunately, it is not an easy task to exploit the intelligence in storage systems. One primary reason is the system latency between VR applications and storage systems. In such a case, VR do not consider the problem of access times of objects in the storage systems. They always simply concerned about how to display the object in the next frame. As a result, the VR can only manage data at the rendering and other related levels without knowing any semantic information such as semantic correlations between data [6]. This motivates a more powerful analysis tool to discover more complex patterns, especially semantic patterns, in storage systems. Therefore, the aim of our work is to decrease this latency through intelligent organization of the access data and enabling the clients to perform predictive prefetching.

In this paper, we consider the problem and solve this using data mining techniques [1,2]. Clearly, when users traverse in a VR, some potential semantic characteristics will emerge on their traversal paths. If we collect the users' traversal paths, mine and extract some kind of information of them, such meaningful semantic information can help to improve the performance of the interactive VR. The rest of this paper is organized as follows. Related works are given in Section 2. The Problem definitions and suggested mining algorithm are explained with illustrative examples shown in Section 3. Section 4 presents our experiment results. Finally, we summarize our current results with suggestions for future research in Section 5.

2. Related Works

In this subsection, we will briefly describe related works about VR, sequential pattern mining and pattern clustering, respectively.

2.1. Virtual Reality Methods

Since the navigation in VR consists of many different detailed objects, e.g., of CAD data that cannot all be stored in main memory but only on hard disk. Many techniques were proposed for rendering complex models used today, including the use of hierarchical spatial structures, level-of-detail (LOD) management [10], hierarchical view-frustum and occlusion culling [10], working-set management (geometry caching) [10]. In additional, Massive Model Rendering (MMR) system [10] was the first published system to handle models with tens of millions of polygons at interactive frame rates. Besides, many *out-of-core* spatial data structures [9], including *kd*-trees, quad-trees, *oct*-tree and *R*-trees [9] were presented. On the other side, it is desirable to store only the polygons and not to produce additional data as, e.g., *textures* or *pre-filtered points*. However, polygons of such highly complex scenes require a lot of hard disk space so that the additional data could exceed the available capacities [10]. To meet these requirements, an appropriate data structure and an efficient technique should be developed with the constraints of memory consumptions.

2.2. Sequential Pattern Mining Methods

Sequential pattern mining was first introduced in [5], which is described as follows. A sequence database is formed by a set of data sequences. Each data sequence includes a series of transactions, ordered by transaction times. This research aims to find all the subsequences whose ratios of appearance exceed the minimum support threshold. In other words, *sequential patterns* are the most frequently occurring subsequences in sequences of sets of items. A number of algorithms and techniques have been proposed to deal with the problem of sequential pattern mining. Many studies have contributed to the efficient mining of sequential patterns [3,7]. Almost all of the previously proposed methods for mining sequential patterns are *apriori*-like[3]. Sequential pattern mining algorithms, in general, can be categorized into three classes: (1) *Apriori-based* : *horizontal partition* methods and *GSP*[2] is one known representative; (2) *Apriori-based*: *vertical partition* methods and SPADE[4] is one example; (3) *projection-based pattern growth* method, such as the famous *FreeSpan* [7] and *PrefixSpan* algorithms [3].

3. Traversal Histories Mining and Problem Formulation

In this section, we extract the useful information in the access history in the form of sequential patterns. In order to mine for sequential patterns, we assume that the continuous client requests are organized into discrete sessions. *Sessions* specify user interest periods and a *session* consists of *a sequence of client requests* for data items ordered with respect to the time of reference [8]. The client request consists of the objects which a client browse and traverse at will in the WT. We denote this type of clients request as *view*. A session consists of one or more views. In correspond to with terminologies used in data mining, a session can be considered as a *sequence*. The whole *database* is considered as a set of sequences. Formally, let $\sum = \{l_1, l_2, ..., l_m\}$ be a set of *m* literals, called *objects* (also called *items*[2]). The *view v* is defined as snapshot of sets of objects which a user observes duration the period. A *view* (also called *itemset*) is an *unordered*, non-empty set of objects. A sequence is an *ordered* list of views. We denote a sequence *s* (also called *transaction*) by $\{v_1, v_2, ..., v_n\}$, where v_j is a view and ordered property is obeyed. We also call v_j an *element* of the sequence. An item can occur only once in an element of a sequence, but can occur multiple times in different elements. We assume, without loss of generality, that items in an element of a sequence are in lexicographical order.

A sequence $<a_1\ a_2 \dots a_n>$ is *contained in* another sequence $<b_1\ b_2 \dots b_m>$if there exist integers $i_1 < i_2 < \dots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_n \subseteq b_{i_n}$. For example, <(*a*)(*b, c*)(*a, d, e*)> is contained in <(*a, b*) (*b, c*)(*a, b, d, e, f*)>, since (*a*) $\subseteq$ (*a, b*), (*b, c*) $\subseteq$ (*b, c*), and (*a, d, e*) $\subseteq$ (*a, b, d, e, f*). However, the sequence <(*c*)(*d*)> is not contained in <(*c, d*)>)and vice versa. The former represents objects *c* and *d* being observed one after the other, while the latter represents objects *c* and *d* being observed together. In a set of sequences, a sequence *s* is *maximal* if *s* in not contained in any other sequence. Let the database *D* be a set of sequences and ordered by increasing recording time. Each sequence records each user's traversal path in the VR system. The *support* for a sequence is defined as the fraction of *D* that "contains" this sequence. A *sequential pattern p* is a sequence whose *support* is equal to or more than the user-defined threshold. *Sequential patter mining* is the process of extracting certain sequential patterns whose support exceeds a predefined minimal support threshold. Given a database *D* of client transactions, the problem of mining sequential patterns is to find the maximal sequences among all sequences that have a certain user-specified minimum support. Each maximal sequence represents a *sequential pattern*.

Finally, we will define our problem as follows. Given a sequence database $D = \{s_1, s_2, \dots, s_n\}$, we design an efficient mining algorithms to obtain our sequential patterns *P* for reducing the disk access time.

3.1. Pattern Growth-based Mining Algorithms

In this section, we will explain our sequential pattern mining method, called ***V**iew-based **S**equence **P**attern **M**ining (VSPM)*. The main ideas come from both *bounded-projection* and *pattern appending* mechanisms. The *bounded-projection* mechanism has one special characteristic, i.e., it always projects the remaining sequence recursively after a new sequential pattern found. They will not mine the objects across the different prefix views. As a result, we would mine the trimmed database recursively. The *pattern appending* mechanism uses the concept of *prefix property*. When we want to find a new sequential pattern in our database, we use the sequential pattern found in previous round as prefix, and append a new object as the new candidate pattern for verification. If the candidate pattern satisfied the minimum support, we regard it as a new sequential pattern and create a bounded projection of it recursively. In order to explore the interesting relationships among these objects, we propose two different kinds of appending methods − called *Intra-View-Appending* method and *Inter-View-Appending* method. The *Intra-View-Appending* method is used to *append a new object in the same view*, and the *Inter-View-Appending* method is used to *append a new object in the next view*. Demonstration example will be given later. The following is the pseudo codes of view sequence mining algorithm.

View-based Sequential Pattern Mining (VSPM) Algorithm

```
// D is the database. P is the set of frequent patterns, and is set to empty initially.
Input: D and P.
Output: P.
Begin
1. Find length-1 frequent sequential patterns.
2. While (any projected sub-database exits) do
3. Begin
4.   Project corresponding sub-sequences into sub-databases under the intra-view appending and inter-view appending.
5.   Mine each sub-database corresponding to each projected sub-sequence.
6.   Find all frequent sequential patterns by applying step 4 and step 5 on the sub-databases recursively.
7. End;  // while
8. return P;
9. End;   // procedure VSPM ends
```

Example 1 (*VSPM*). Given the traversal data base *S* and *min_support* = 3. The final mining result is depicted in Fig. 1.

Path1: <(1, 2)(3, 4)(5, 6)>.

Path2: <(1, 2)(3, 4)(5)>.

Path3: <(1, 2)(3)(4, 5)>.

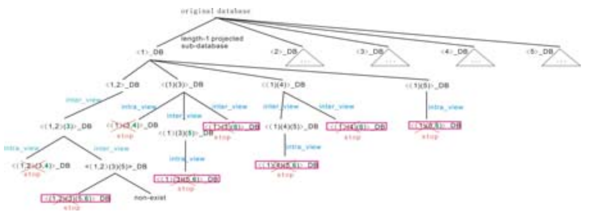


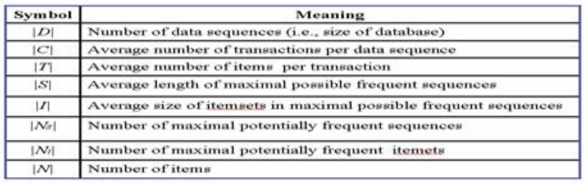
Fig. 1. Demonstration of our VSPM for generating projected-based sub-databases and sequential patterns.

Besides, in Figure 1, the patterns which contain item 6 are circled. They show that the differences between projected-based mining and non-projected-based mining. In other words, without projecting mechanism, we have to expand *eight* sub-databases for candidates (i.e., *two* "stop" without circled plus *six* "stop" with circled). Compared to this case, with projecting mechanism, we only expand *two* sub-databases for candidates (i.e., "stop" without circled).

4. Performance Evaluation

We use the virtual power plant model from http://www.cs.unc.edu/~walk/ created by Walkthrough Laboratory of Department of Computer Science of University of North Carolina at Chapel Hill.

Our main performance metric is the *average latency*. We also measured the *client cache hit ratio*. A decrease in the average latency is an indication of how useful the proposed methods are. The average latency can decrease as a result of both increases cache hit ratio via prefetching methods and better data organization in the disk. An increase in the cache hit ratio will also decrease the number of requests sent to server and, thus, lead to both saving of the scare memory source of the server and reduction in the server load. The performance of our traversal database is reported as follows. First, we follow the procedure described in [2] to set up out data set parameters. The meanings of all parameters are listed in Table 1. From Fig. 2and 3, we can see that *VSPM* is as efficient as *PrefixSpan* does, but it is much more efficient than *SPADE*, *FreeSpan*, and *GSP*.



Symbol	Meaning
$\|D\|$	Number of data sequences (i.e., size of database)
$\|C\|$	Average number of transactions per data sequence
$\|T\|$	Average number of items per transaction
$\|S\|$	Average length of maximal possible frequent sequences
$\|I\|$	Average size of itemsets in maximal possible frequent sequences
$\|N_S\|$	Number of maximal potentially frequent sequences
$\|N_I\|$	Number of maximal potentially frequent itemets
$\|N\|$	Number of items

Table 1. Parameters for our traversal data set

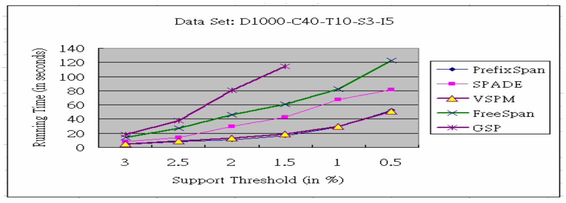


Fig. 2 Execution time with respect to various support thresholds using our real data set-1.

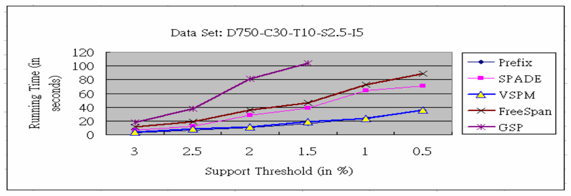


Fig. 3. Execution time with respect to various support thresholds using our real data set-2.

Definition 1: *View-radius*

The *view radius* is defined as the radius of the visible circle in the VR. As the radius increases, the more objects are observed. In other words, it controls how many objects are observed at the same time in one view.

In the meanwhile, we select the different views-radius for comparison. Fig. 4 and 5 show the results. Algorithm with VSPM outperforms other algorithms without VSPM. Since the clustering mechanism can accurately support prefetching objects for future usage. Not only the access time is cut down but also the I/O efficiency is improved.

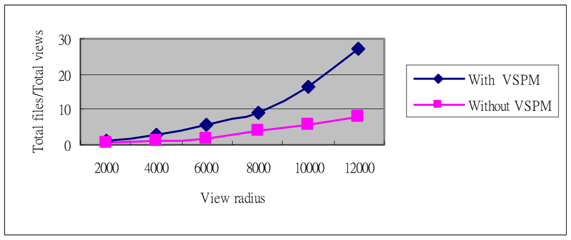


Fig. 4. Comparison of different algorithms on the number of files retrieved under the same view.

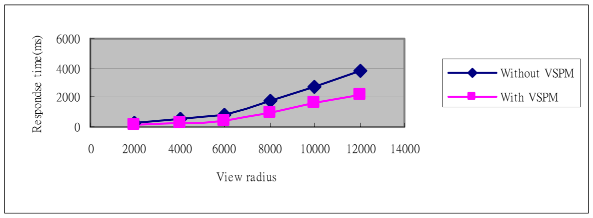


Fig. 5. Comparison of different algorithms on system response time under the same view_radius.

5. Conclusions and Future Works

In this paper, we have designed a frequent projection-based sequential pattern mining algorithm to find correlations among objects. Using the VR traces, our experiments show that *VSPM* is an efficient algorithm. Besides, we have evaluated correlation-directed prefetching and data layout. Our experimental results have shown that correlation-directed prefetching and data layout can improve both I/O average response time and the number of retrieved files. Our study still has limitations. One important limitation is that our disk layout was not especially designed for the extra long frequent sequential patterns. This direction will enhance the system performance.

References

[1] M. S. Chen., J S. Park, and P.S. Yu, "Efficient Data Mining for Path Traversal Patterns," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 2, pp. 209-221, 1998.

[2] R. Srikant and R. Agrawal, 'Mining Sequential Patterns: Generalizations and Performance Improvements," *Proceeding Fifth International Conference Extending Database Technology (EDBT'96)*, pp. 3-17, Mar,1996.

[3] J. Pei et al., "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 3-14, 2001.

[4] M. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences, " *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, pp. 103-114, June,1996.

[5] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *11th International Conference on Data Engineering*, Montreal, Quebec, Canada, Vol. 25, No. 2, pp. 1-12, 1995.

[6] M. Sivathanu, V.Prabhakaran, F. Popovici, T.E. Denehy, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Semantically-smart disk systems," *Proceedings of the Second USENIX Conference on File and Storage Technologies*, 2003.

[7] J. Han, J. Pei, B, Mortazavi-Asl Chen Q, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining," *Proceeding 2000 ACM SIGKDD International Conference Knowledge Discovery in Database (KDD'00)*, pp. 355-359, August 2000.

[8] A. Joshiand R. Krishnapuram, "On Mining Web Access Logs," *Proceeding SIGKDD Workship Research Issues on data mining and Knowledge Discovery(DMKD)*, 2000.

[9] L. Arge, K. Hinrichs, J. Vahrenhold, J.Vitter, "Efficient Bulk Operations on Dynamic R-tress," *Proceeding Workship on Algorithm Engineering*, pp. 104-128, 1999.

[10] Y. Zhu, "Uniform Remeshing with an Adaptive Domain: A New Scheme for View-Dependent Level-Of-Detail Rendering of Meshes,"*IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 3, pp. 306-316, May-June 2005.