Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

# Using Hypergraph-based Clustering Scheme for Traversal Prediction in Virtual Environments

Shao-Shin Hung and Damon Shing-Min Liu

***Abstract*—In many virtual environments (VE) applications, the size of the database is not only extremely large, it is also growing rapidly. Even for relatively simple searches, the time required to move the data off storage media is expensive. However, object correlations are common semantic patterns in VE. They can be exploited for improve the effectiveness of storage caching, prefetching, data layout, and disk scheduling. However, little approaches for discovering object correlations in VE to improve the performance of storage systems. In this paper, we develop a class of view-based projection-generation method for mining various frequent sequential traversal patterns in the VE. The frequent sequential traversal patterns are used to predict the user navigation behavior. Furthermore, the hypergraph-based clustering scheme can help reduce disk access time with proper placement patterns into disk blocks. Finally, 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.**

## I. Introduction

Having inexpensive data storage has enabled the amassing of large amounts of information, especially in VE. These data are rapidly accessible, motivating a significant interest in VE capabilities. At present, these data sets far exceed the capability of modern storage systems, so searching them has become a serious challenge. As [24] cited, "The size of the databases we deal with is no long measured in terabytes, but in exabytes." On the other side, 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 VE applications and storage systems. In such a case, VE 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 VE can only manage data at the rendering and other related levels without knowing any semantic information such as semantic correlations between data. 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 virtual environment, 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 VE. For example, we can reconstruct the placement order of the objects of 3D model in disk according to the common section of users' path. Exploring these correlations is very useful for improving the effectiveness of storage caching, prefetching, data layout, and disk scheduling.

This paper proposes *VSPM* (*Viewed-based Sequential Pattern Mining*), a method which applies a data mining technique called *frequent sequential pattern mining* to discover object correlations in VE. Specially, we have modified several recently proposed data mining algorithms called *FreeSpan* [5] and *PrefixSpan* [11] to find object correlations in several traversal traces collected in real systems. To the best of our knowledge, *VSPM* is the first approach to infer object correlations in a VE. Furthermore, *VSPM* is more scalable and space-efficient than previous approaches. It runs reasonably fast with reasonable space overhead, indicating that it is a practical tool for dynamically inferring correlations in a VE. Besides, we have also proposed two clustering methods to cluster the similar patterns for reducing the access time. One is based on the idea of *co-occurrence* of transaction data have developed. They are usually measured by *Jaccard* coefficient $SIM(T_1, T_2) = |T_1 \cap T_2| / |T_1 \cup T_2|$ [22]. The other clustering scheme is based on the hypergraph-based model. In this model, the vertex set corresponds to the distinct objects in the VE and the hypergedges correspond to the frequent sequential patterns. Both of them will make similar objects much closer to be accessed in one time. These result in less access times and much better performance. We also compare the distinctions between them. Moreover, we also have evaluated the benefits of object correlation-directed prefetching and disk data layout using the real system workloads.

The rest of this paper is organized as follows. Related works are given in Section II. In Section III, we describe our problem formulation. The system architecture is suggested in Section IV. The suggested mining and clustering mechanisms are explained with illustrative examples shown in Section V. Section VI presents our experiment results. Finally, we summarize our current results with suggestions for future research in Section VII.

Shao-Shin Hung is with Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, Republic of China (corresponding author to provide phone: +886-5-272-0411 ext 23101; fax: +886-5-272-0859; e-mail: hss@cs.ccu.edu.tw).

Damon Shing-Min Liu is with Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, Republic of China. (e-mail:damon@cs.ccu.edu.tw).

## II. Related Works

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

### *A. Virtual Environments Methods*

Since the navigation in virtual environments 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 [20], hierarchical view-frustum and occlusion culling [12], working-set management (geometry caching) [20]. In additional, Massive Model Rendering (MMR) system [13] 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 [23], including *kd*-trees, quad-trees, *oct*-tree and *R*-trees [23] 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 [21]. To meet these requirements, an appropriate data structure and an efficient technique should be developed with the constraints of memory consumptions.

### *B. Sequential Pattern Mining Methods*

Sequential pattern mining was first introduced in [8], 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 [5,11]. Almost all of the previously proposed methods for mining sequential patterns are *apriori*-like [5]. Sequential pattern mining algorithms, in general, can be categorized into three classes: (1) *Apriori-based* : *horizontal partition* methods and *GSP* [7] is one known representative; (2) *Apriori-based*: *vertical partition* methods and *SPADE* [6] is one example; (3) *projection-based pattern growth* method, such as the famous *FreeSpan* [11] and *PrefixSpan* algorithms [5].

### *C. Hypergraph_based Pattern Clustering Methods*

The fundamental clustering problem is to partition a given data set into groups (clusters), such that data points in a cluster are more similar to each other (i.e., *intra-similar property*) than points in different clusters (i.e., *inter-similar property*) [10]. Although there are many clustering algorithms presented above, they can not be applied to our data set directly. These discovered clusters are used to explain the characteristics of the data distribution [Kumar-01-HG, 18]. However, these schemes fail to produce meaningful clusters, if the number of objects is large or the dimensionalities of the VE (i.e., the number of different features) are diverse and relatively large.

In this paper, we propose a new methodology for clustering correlated objects using frequent sequential patterns, and clustering related patterns using clusters of objects. These frequent sequential patterns are used to group objects into hypergraph edges, and a hypergraph partition algorithm is used to find the clusters. The knowledge that is represented by clusters of related objects can also be used to effectively cluster the actual semantic patterns by looking at the clusters that these objects belong to. Therefore, we can layout these clusters onto the storage systems for prediction of user traversal.

## III. Our Motivations

### *A. Motivations on Theoretical Foundations*

Data mining research deals with finding relationships among data items and grouping the related items together. The two basic relationships that are of particular concern to us are:

- *Association*, where the only knowledge we have is that the idea items are frequently occurring together and, when one occurs, it is highly probable that the other will also occur.
- *Sequence*, where the data items are associated and, in addition to that, we know the order of occurrence as well.

Our main interest is to find the sequence among the data items that occur frequently. As the concept of sequence is based on associations, we first briefly introduce the issue of finding associations. The formal definition of the problem of finding association rules among items is provided by [8] as follows: Let $I = i_1, i_2, \ldots, i_n$ be a set of literals, called items, and $D$ be a set of transactions such that $\forall T \in D$, $T \subseteq I$. A transaction $T$ contains a set of items $X$ if $X \subseteq T$. An *association rule* is denoted by an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \varnothing$ . As a rule $X \Rightarrow Y$ is said to hold in the transaction set $D$ with *support s* in the transaction set $D$ if $s$ % of transactions in $D$ contain $X \cup Y$. The rule $X \Rightarrow Y$ has *confidence c* if $c$ % of the transactions in $D$ that contain $X$ also contain $Y$. The thresholds for support and confidence are called *minsup* and *minconf*, respectively.

One of the challenges of mining client access histories is that such histories are continuous while mining algorithms assume transactional data. This causes a mismatch between the data required by current algorithms and the access history we are considering. Therefore, we need to convert continuous requests into transactional form, where client requests in transactions correspond to a session. A session consists of a set of virtual objects accessed by a user in a certain amount of time. Similar researches can be found in [19]. They presented methods for efficiently organizing the sequential web log into transactional form suitable for mining.

### *B. Motivations on Practical Demands*

On the practical view of point, we will demonstrate several practical examples to explain our observation. Suppose that we have a set of data items {*a, b, c, d, e, f, g*}. A sample access history over these items consisting of five sessions is shown in Table 1. The request sequences extracted from this history with minimum support 40 percent are (*a*, *f*) and (*c*, *d*). The rules obtained out of these sequences with 100 percent minimum confidence are $a \Rightarrow f$ and $c \Rightarrow d$, as shown in Table 2. Two accessed data organizations are depicted in Figure 1. An accessed schedule without any intelligent preprocessing is shown in Figure 1a. A schedule where related items are grouped together and sorted with respect to the order of reference is shown in Figure 1b. Assume that the disk is spinning counterclockwise and consider the following client request sequence, *a, f, b, c, d, a, f, g, e, c, d*, shown in Figure 1. Note that dashed lines mean that the first element in the request sequence (counted from left to right) would like to fetch the first item supplied by disk. And directed graph denotes the rotation of disk layout in counterclockwise way. For this request, if we have the access schedule *(a, b, c, d, e, f, g)*, which dose not take into account the rules, the total I/O access times for the client will be *a*:5, *f*:5, *b*:3, *c*:2, *d*:6, *a*:5, *f*:5, *g*:1, *e*:5, *c*:6, *d*:6. The total access times is 49 and the average latency will be 49/11= 4.454. However, if we partition the items to be accessed into two groups with respect to the sequential patterns obtained after mining, then we will have {*a, b, f*} and {*c, d, e, g*}. Note that data items that appear in the same sequential pattern are placed in the same group. When we sort the data items in the same group with respect to the rules $a \Rightarrow f$ and $c \Rightarrow d$, we will have the sequences *(a, f, b)* and *(c, d, g, e)*. If we organize the data items to be accessed with respect to these sorted groups of items, we will have the access schedule presented in Figure 1b. In this case, the total access times for the client for the same request pattern will be *a*:1, *f*:1, *b*:1, *c*:1, *d*:1, *a*:3, *f*:1, *g*:4, *e*:1, *c*:4, *d*:1. The total access times is 19 and the average latency will be 19/11= 1.727, which is much lower than 4.454.

Table 1: Sample database of user requests.

| Session No | Accessed Request |
|---|---|
| 1 | e, a, f |
| 2 | b, d |
| 3 | c, d, a, f, g |
| 4 | b, a, f, g |
| 5 | c, d, a, f |

Table 2: Sample association rules.

| Rule | Support | Confidence |
|---|---|---|
| $a \Rightarrow f$ | 80% | 100% |
| $c \Rightarrow d$ | 40% | 100% |

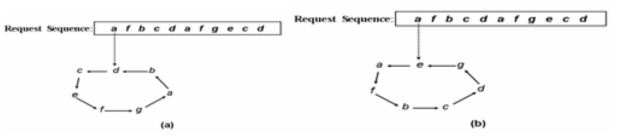

Request Sequence: a f b c d a f g e c d
c
d
b
e
a
f
g
(a)
Request Sequence: a f b c d a f g e c d
a
e
g
f
d
b
c
(b)

Fig. 1. Effects on accessed objects organization in disk. (a): without association rules; (b) with association rules.

Another example that demonstrates the benefits of rule-based prefetching is shown in Figure 2. We demonstrate three different requests of a client as a snapshot. With the help of the rules obtained form the history of previous requests, the prediction can be achieved. The current request is *c* and these is a rule stating that, if data items *c* is requested, then data items *d* will be also be requested (i.e., association rule $c \Rightarrow d$). In Figure 2a, data item *d* is absent in the cache and the client must spend more waiting time for item *d*. In Figure 2b, although the item *d* is also absent in the cache, the client still spend one disk latency time for item *d*. In Figure 2c, the cache can supply the item *d* and no disk latency time is needed.

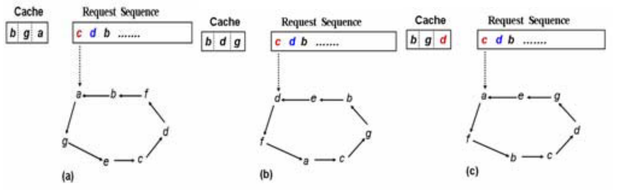


Fig. 2. Effects of prefetching.

These simple examples show that, with some intelligent grouping, reorganization of data items and with predictive prefetching, average latency for clients can be considerably improved. In the following sections, we describe how we can extract sequential patterns out of client requests. We also explain how we group data items with respect to sequential patterns.

## IV. Mining Traversal Histories 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. The client request consists of the objects which a client browse and traverse at will in the VE. 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*) [18]. 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 $<a_1\ a_2\ ...\ a_n>$ is *contained in* another sequence $<b_1\ b_2\ ...\ b_m>$ if there exist integers $i_1 < i_2 < … < i_n$ such that

$a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$. 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 walk through 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 in two phases. Phase I: given a sequence database $D = \{s_1, s_2, \ldots, s_n\}$, we design a efficient mining algorithms to obtain our sequential patterns *P*; phase II: In order to reduce the disk access time, we distribute *P* into a set of clusters, such that minimize inter-cluster similarity and maximize intra-cluster similarity.

## V. Pattern-Oriented Mining Algorithms

In this section, we will explain our sequential pattern mining method, called ***V**iew-based **S**equence **P**attern **M**ining (VSPM)*. Since our input data are different from those of traditional data mining algorithms [16]. We make several major modifications about the idea of pattern-growth method. Its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. This process partitions both the data and set of frequent sequential patterns to be tested, and confines each test being conducted to the corresponding smaller projected database.

### *A. View-based Sequential Pattern Mining (VSPM) Algorithm*

Now, we will explain our mining algorithms. 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, we demonstrate the complete steps as follows.

Path1: <(1, 2)(3, 4)(5, 6)>.
Path2: <(1, 2)(3, 4)(5)>.
Path3: <(1, 2)(3)(4, 5)>.

**Step1**. *Find frequent patterns* with length-1. //in the form of "*item: support*"
First, we will have the following data: 1:3, 2:3, 3:3, 4:3, 5:3, 6:1. Therefore, we have *length-1* frequent sequential patterns: <1>, <2>, <3>, <4>, and <5>. Finally, we will have 5 projection-based sub-databases <1>_DB, <2>_DB, <3>_DB, <4>_DB and <5>_DB, respectively.

**Step2**. Take the projection-based sub-database, <1>_DB, for example. First, since item 2 and item 1 are *in same view*, the *intra-view appending* works. After the projection, we will get the sub-database <(1,2)>_DB. And the original database is shrunk to the following database.
P1: <(3,4)(5,6)>; P2: <(3,4)(5)>; P3: <(3)(4,5)>.
In this step, pattern <(1,2)> becomes a frequent sequent pattern since its support satisfies the minimum support. Next, item 3 is projected for the candidate.

**Step3**: the remaining steps are the same as the above. The final mining result is depicted in Figure 3. In Figure 3, 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).

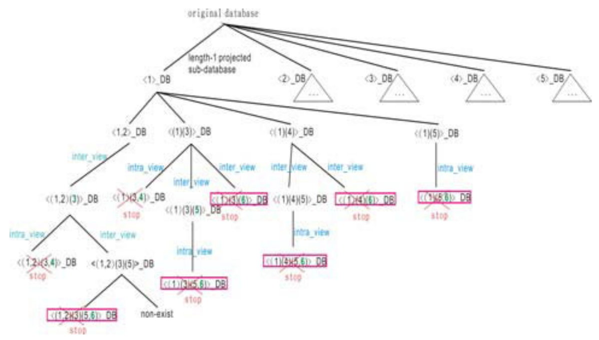


Fig. 3: Demonstration of our *VSPM* for generating projected-based sub-databases and sequential patterns.

*B. Disk Organization by Hypergraph_based Clustering Sequential Patterns*

Clustering is a good candidate for inferring object correlations in storage systems. As the previous sections mentioned, object correlations can be exploited to improve storage system performance. First, correlations can be used to direct prefetching. For example, if a strong correlation exists between objects *a* and *b*, these two objects can be fetched together from disks whenever on of them is accessed. The disk read-ahead optimization is an example of exploiting the simple data correlations by prefetching subsequent disk blocks ahead of time. Several studies [10,15] have shown that using these correlations can significantly improve the storage system performance. Our results in Section 6.2 demonstrate that prefetching based on object correlations can improve the performance much better than that of non-correlation layout in all cases.

A storage system can also lay out data is disks according to object correlations. For example, a object can be collocated with its correlated objects so that they can be fetched together using just one disk access. This optimization can reduce the number of disk seeks and rotations, which dominate the average disk access latency. With correlation-directed disk layouts, the system only needs to pay a one-time seek and rotational delay to get multiple objects that are likely to be accessed soon. Previous studies [9] have shown promising results in allocating correlated file blocks on the same track to avoid track-switching costs.

The hypergraph was introduced by Berge [3] and has been considered as a useful tool to analyze the structure of a system and to model a partition, covering, and clustering. A hypergraph *H=(V, N)*is defined as a set of vertices and a set of hyperedges (nets [26]). Every net $n_j \in N$ is a subset of vertices, i.e., $n_j \subseteq V$. The size of a net $n_j \subseteq N$ is equal to the number of vertices it has, i.e., $s_j = |n_j|$. Weight ($w_i$) and cost ($c_j$ ) can be assigned to the vertices ($v_i \in V$) and edges ($n_j \in N$) of the hypergraph, respectively. $K = \{V_1, V_2, \ldots, V_k\}$ is a *K*-way partition of *H* if (a) each partition is a nonempty subset of *V*, (b) partitions are pairwise disjoint and (c) union of *K* partitions is equal to *V*.

*C. Similarity Measure for Jaccard function*

In the simplified hypothesis that frequent patterns do not contain frequencies, but behave simple as Boolean vectors (like a value 1 corresponds to the presence and a value 0 corresponds to the absence), and a more intuitive but equivalent way of defining the *Jaccard distance function* can be provided. This measure captures our idea of similarity between items, which are directly proportional to the number of common values, and inversely proportional to the number of different values for the same item.

**Definition 1: *Intra-distance measure (Co-occurrence)***
Let $P_1$ and $P_2$ be two sequential patterns. We can represent $D(P_1, P_2)$ as the normalized difference between the cardinality of their union and the cardinality of their intersection:

$$D(P_1, P_2) = 1 - \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|} \tag{1}$$

**Example 2 (*Intra-distance measure*).**
Let $P_1$ and $P_2$ be two sequential patterns: $P_1$ =<(*a*, *b*, *c*), (*b*, *c*, *d*), (*e*,*f*)>and $P_2$ =<(*a*, *b*, *c*, *d*), (*e*,*f*, *g*)>. The distance between $P_1$ and $P_2$ is

$$D(P_1, P_2) = 1 - \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|} = 1 - \frac{|\{a,b,c,e,f\}|}{|\{a,b,c,d,e,f,g\}|} = 1 - \frac{5}{7} = \frac{2}{7}$$

*D. Sequential Pattern Clustering Algorithms*

Intuitively, a cluster representative for virtual environment data should model the content of a cluster, in terms of the objects that are most likely to appear in a pattern belonging to the cluster. A problem with the traditional distance measures is that the computation of a cluster representative is computationally expensive. As a consequence, most approaches [6] approximate the cluster representative with the Euclidean representative. However, those approaches may suffer the following drawbacks:

• Huge cluster representatives cause poor performances, mainly because as soon as the clusters are populated, the cluster representatives are likely to because extremely huge.
• For different kinds of patterns, it seems to be difficult to find the proper cluster representatives.

In order to overcome such problems, we can compute an approximation that resembles the cluster representatives associated to Euclidean and mismatch-count distances. Union and intersection seem good candidates to start with. Since our clustering operations are based on set operations, we ignore the order of frequent patterns.

*D1.Jaccard _based Pattern Clustering Algorithms*

To avoid these undesired situations, we supply three tables. The first table is *FreqTable*. It records the frequency of any two patterns co-existing in the database $D$. The second table is *DistTable*. It records the distance between any two patterns. The last table is ***Cluster***. It records how many clusters are generated. The following is our clustering algorithm.

***Pattern Clustering Algorithm for Jaccard Function***

```
// P is the set of frequent patterns. T is the set of clusters, and is set to empty initially.
Input: P and T.
Output: T
Begin
1. FreqTable={$ft_{ij}$| the frequency of $pattern_i$ and $pattern_j$
     co-existing in the database D};
2. DistTable={$dt_{ij}$| the distance between of $pattern_i$ and
     $pattern_j$ in the database D};
3. $C_1$={$C_i$| At the beginning each pattern to be a single
     cluster }
4. // Set up the Extra-Similarity Table for evaluation
5. $M_1$= Intra-Similar ($C_1$, $\varnothing$);
6. k = 1;
7. while $|C_k| > n$ do Begin
8.        $C_{k+1}$ = PatternCluster ($C_k$, $M_k$, FreqTable,
              DistTable);
9.        $M_{k+1}$ = Intra-Similar ($C_{k+1}$, $M_k$);
10.       k = k +1;
11. End;
12. return $C_k$ ;
13. End;
```

*D2.Hypergraph _based Pattern Clustering Algorithms*

As mentioned before, the vertex set corresponds to the distinct objects in the VE and the hypergedges correspond to the frequent sequential patterns. The weight of hyperedge is the support of that sequential pattern. In this paper, we adopt the hypergraph partition algorithm in [25]. Since there are no expensive data structures or special constraints hidden, we can implement them very efficient and time/space complexity also meet our demands.

## VI. PERFORMANCE EVALUATION

A traversal path database was recorded each user's traversal path and used for mining target. The simulation model we used and the experimental results are provided in Section 6.1 and Section 6.2, respectively.

### *A. Test data and Simulation Model*

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. The Power Plant Model is a complete model of an actual coal fired power plant. The model consists of 12,748,510 triangles. Its size is 128 MBytes. Our traversal database keeps track of the traversal of the power plant by many anonymous, randomly users. For each user, the data records list all the areas of the power plant that user visited in a one week timeframe. Each path consists of 30 ~ 40 views. Each view consists of 20~30 objects on average. The number of objects is 11, 949, where each object is a some meaningful combination of triangles of power plant and it is considered as a data item.

### *B. Experimental Results and Performance Study*

In this section, the effectiveness of the proposed clustering algorithm is investigated. All algorithms were implemented in Java. The experiments were run on a PC with a AMD Athlon 1800+ and 512 megabytes main memory, running Microsoft Windows 2000 server. 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.

Since we have two major topics ─ mining algorithm and clustering algorithm. We report our experimental results on the performance of mining and clustering, respectively.

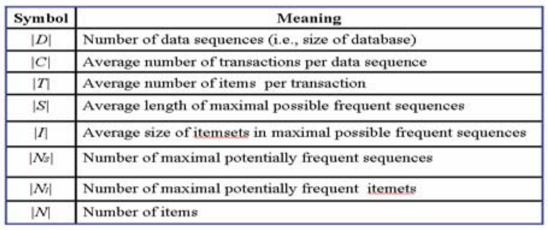
#### *B.1. Experimental Results on Mining Unit*

In this subsection, we report our experimental results on the *VSPM* algorithm. Since *GSP* and *SPADE* are the two most important sequential pattern mining algorithms, we conduct an extensive performance study to compare *VSPM* with them. To evaluate the effectiveness and efficiency of the VSPM algorithm, we performed an extensive performance study of *GSP*, *SPADE*, *FreeSpan*, and *PrefixSpan*, on real data sets, with various kinds of sizes and data distribution. Besides, these four algorithms, *GSP*, *SPADE*, *FreeSpan*, and *PrefixSpan* were implemented in Java.

***Virtual environment traces***

The performance of our traversal data base is reported as follows. First, we follow the procedure described in [4] to set up out data set parameters. The meanings of all parameters are listed in Table 3. Figure 4 and 5 show the performance comparison among the five algorithms for our virtual environment data set. From Figure 4 and 5, we can see that *VSPM* is as efficient as *PrefixSpan* does, but it is much more efficient than *SPADE*, *FreeSpan*, and *GSP*.

Table 3. Parameters for our traversal data set



| 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 |

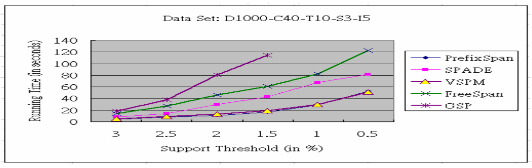


Fig. 4 Execution time with respect to various support thresholds using our real data set-1.

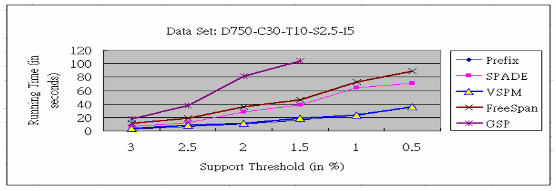


Fig. 5. Execution time with respect to various support thresholds using our real data set-2.

*B.2. Experimental results on clustering unit*

For quality measure of clustering result, we adopted the cluster cohesion and the inter-clustering similarity. All are defined as follows.

**Definition 2:** ***Large item***

Given a *pattern*$_i$ and a user-defined threshold $\theta$, if it satisfies the following criterion:

$0 <$ *minimum support threshold* $< \theta \leq$ *support(pattern*$_i$*)* $\leq 1$. We call the *pattern*$_i$ as a *large item.*

**Definition 3:** ***Cluster Cohesion (Cluster-Coh($C_i$))***

It is the ratio of the large items to the whole items $T(C_i)$ in the cluster $C_i$. This is calculated by the following formula, and if it is near 1, it is a good quality cluster; otherwise, it is not.

$$Cluster-Coh(C_i) = \frac{C_i(L)}{T(C_i)} \qquad (2)$$

where $C_i\ (L)$ is the number of large item in cluster $C_i$ and $T(C_i)$ is number of all items in cluster $C_i$.

**Definition 4:** ***Inter-Cluster Similarity (inter-sim($C_i$, $C_j$)))***

It is based on the large items is the rate of the common large items of the cluster $C_i$.and $C_j$. We calculate the inter-cluster similarity by the following formula, and if it is near 0, it is the good clustering. Otherwise, it is not.

$$Sim(C_i,C_j) = \frac{LarCom(C_i \cap C_j)}{C_i(L)+C_j(L)} \times \frac{|LarCom(C_i \cap C_j)|}{|LarCom(C_i + C_j)|} \qquad (3)$$

where *LarCom*($C_i \cap C_j$) is the number of *common large items* in the cluster $C_i$.and $C_j$, |*LarCom*($C_i \cap C_j$)| is the total occurrence number of the common large items, and |*LarCom*($C_i \cap C_j$)| is the total occurrence number of the large items in the cluster $C_i$and $C_j$.

**Definition 5:** ***View-radius***

The v*iew radius* is defined as the radius of the visible circle in the virtual environments. 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 support threshold for comparison. Figure 6 and 7 show the results. Algorithms with clustering outperforms other algorithms without clustering. Since the clustering mechanisms can accurately support prefetching objects for future usage. Not only the access time is cut down but also the I/O efficiency is improved. Note that HG_clustering represents the Hypergraph clustering scheme.

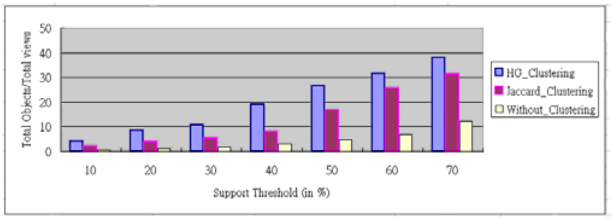


Fig. 6. Comparison of different algorithms on the number of objects retrieved under the same view_radius.

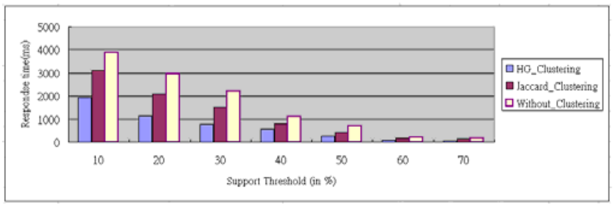


Fig. 7. Comparison of different algorithms on system response time under the same view_radius.

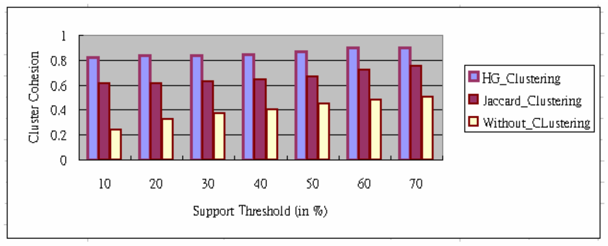


Fig. 8. Comparison of different support threshold on cluster cohesion under the same view_radius.

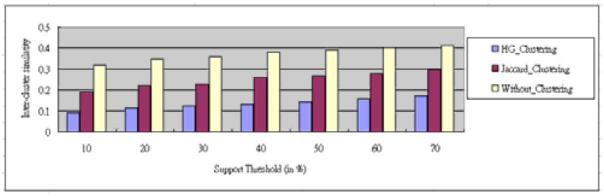


Fig. 9. Comparison of different support threshold on inter-cluster similarity under the same view_radius.

By observing both Figure 8 and 9, we can easily realize that there exist relations between the number of clusters and inter-cluster similarity, and also between the number of clusters and cluster cohesion. Among them, the HG_clustering outperforms the other two. Since HG_clustering scheme can capture the inter-/intra-relationships in clusters as many as possible, the Jaccard clustering does less and the last one does nothing, just based on its random behavior.

In summary, we can determinate that our HG_clustering algorithm is better overall at cluster cohesion and inter-cluster similarity. This means that our HG_clustering algorithm can groups more similar patterns together and do more improvements on the efficiency of storage systems.

## VII. Conclusions and Future Work

In this paper, we have designed a frequent projection-based sequential pattern mining algorithm to find correlations among objects. Using the VE 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 I/O average response time by 1.998 to 3.201 compared to no-prefetching, and 3.102 to 9.121compared to the number of retrieved objects. Finally, we have also designed two criteria to verify the validity of clustering method.

Our study has several 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] S. Chakrabarti, *Mining the Web: Discovering Knowledge from Hypertext Data*, Morgan Kaufmann Publishing, 2003.

[2] 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. 10, Issue 2, 1998. pp. 209-221.

[3]C. Berge, Graphes et Hypergraphs, Paris: Dunod, 1970.

[4] R. Srikant and R. Agrawal, "Mining sequential patterns: generalizations and performance improvements", *Proceeding Fifth International Conference Extending Database Technology (EDBT'96)*, Mar, 1996, pp. 3-17.

[5] J. Pei et al, "PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth", *Proceedings of the International Conference on Data Engineering (ICDE)*, 2001, pp. 3-14.

[6] M, Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Volume 25 Issue 2, Montreal, Quebec, Canada, June 1996, pp. 103-114.

[7] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables", *Proceedings* of the 1996 ACM *SIGMOD International Conference on Management of Data*, Volume 25 Issue 2, Montreal, Quebec, Canada, 1996, pp. 1-12.

[8] R. Agrawal and R. Srikant, "Mining sequential patterns", *11th International Conference on Data Engineering*, Volume 25, Issue 2, Montreal, Quebec, Canada, 1995, pp. 1-12.

[9] 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.

[10] J. Choi, S. H. Noh, S. L. Min. and Y. Cho, "Towards applications/files-level characterization of block reference: a case for fine-grained buffer management," *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computes*, 2000.

[11] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, 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),* August 2000, pp. 355-359.

[12] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communication of the ACM*, Vol. 19, No. 10, October, 1976, pp. 547-554.

[13] J.M. Airey, J.H. Rohlf, and J.F.P. Brooks, "Towards image realism with interactive updates in complex virtual building environments," *1990 ACM Symposium on Interactive 3D Graphics*, 24(2), Mar, 1990, pp. 41-50.

[14] L. Kaufman and Peter J. Rousseuw, *Finding Groups in Data*, Wiley, New York, 1990.

[15] A.J. Smith, "Sequentiality and prefetching in database systems," *ACM .Transactions on Database Systems*, Vol. 3, No. 3, September, 1978, pp. 223-247.

[16] S.S. Hung, T.C. Kuo, and D.S.M. Liu, "PrefixUnion: mining traversal patterns efficiently in virtual environments", *International Conference on Computational Science (ICCS-2005)*, LNCS vol. 3516, May 2005, pp. 830-834.

[17] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , Vol. 15, No. 13, 1993, pp. 1101-1113.

[18] S. Goil, H. Nagesh, and A. Choudhary, "MAFIA: efficient and scalable subspace clustering for very large data sets," Technical Report CPDC-TR-9906-010, Northwestern University, 2145 Sheridan Roas, Evanston IL 60208, June, 1999.

[19] A. Joshi and R. Krishnapuram, "On mining web access logs," *Proceeding SIGKDD Workship Research Issues on data mining and Knowledge Discovery(DMKD)*, 2000.

[20] Zhu, Y, "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, Issue 3, May-June, pp. 306-316, 2005.

[21] E. Ohbuchi, "A Real-Time refraction renderer for volume objects using a polygon-rendering scheme," Proceeding of Computer Graphics, 2003, pp. 190-195.

[22] P. Jaccard, "The distribution of the Flora of the Alpine Zone," *New Phytologist*, 1912, pp. 37-50.

[23] L. Arge, K. Hinrichs, J. Vahrenhold, and J. Vitter, "Efficient bulk operations on dynamic R-tress," *Proceeding Workship on Algorithm Engineering*, 1999, pp. 104-128.

[24] J. Reynders, "Computing biology," invited talk at 5th High performance Embedded Computing Workshop, November 2001.

[25] H.L. Kwang and C.H. Cho, "Hierarchical reduction and prediction of hypergraph," *IEEE transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, Vol. 26, No. 2, April 1996, pp.340-344.

[26] E-H. Han, G. Karypis, V. Kumar and B. Mobasher, "Clustering based on association rule hypergraph," Workshop on Research Issues on Data Mining and Knowledge Discovery, 1997.