

Parallel WAP-mine on PC Cluster

Iko Pramudiono Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan
Phone +81-3-5452-6098 Fax +81-3-5452-6457
{iko,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

Mining web data has become one of the demanding data mining tasks. Web access patterns(WAP) which are essential for applications such as recommendation and prefetching have to be mined from few GBs of web logs in almost timely manner. We develop a parallel web access pattern mining algorithm based on WAP-mine, an algorithm with modern divide-and-conquer paradigm. We propose a new compression method to reduce communication overhead. Performance evaluation on PC cluster shows good speedup ratio.

Keywords : web access pattern mining, PC cluster, parallel processing

1 Introduction

The explosive growth of World Wide Web generates abundant data in tremendous pace. The task to analyze the data from the web has become a hot research area called web mining. The web mining can be classified into three categories : web structure mining, web content mining and web usage mining. Mining web access patterns from access logs of web server is one of the foundational technique in web usage mining. Some applications such as prefetching and recommendation require web access patterns in timely manner from at least a few GBs of web logs, and the size of web logs is growing rapidly.

Thus efficient and scalable algorithms to mine web access patterns are needed as well as scalable platforms.

The web access pattern is essentially a sequential pattern. The problem of mining sequential pattern was first addressed by AprioriAll algorithm [3]. The algorithm and many algorithms for sequential pattern mining employ level-wise “generate-and-test” paradigm introduced by Apriori [2]. They encounter the difficulty when the length of the pattern grows

long, which is exactly the case in web access pattern mining.

Recently an algorithm to mine web access patterns based on novel paradigm of “divide-and-conquer” called WAP-mine is proposed [6]. The paradigm has become popular since the success of FP-growth [5]. WAP-mine compresses the web sequence database into on-memory data structure called WAP-tree so that it can find web access pattern without candidate sequence generation. WAP-mine is reported to mine wap access patterns more efficiently than Apriori derivatives.

Further performance improvement can be expected from parallel execution. PC cluster is a promising platform for this task because of its expandable configuration. However a parallel algorithm for complex data structure like the WAP-tree is much harder to implement compared to a serial program or shared-memory parallel system.

In this paper our contribution is twofold. The first, we propose a parallel algorithm to mine web access patterns on a PC cluster. The algorithm reconstructs conditional access sequence bases from local WAP-trees. This approach exploits both data parallelism and task parallelism efficiently.

On a PC cluster, whose bandwidth and network latency are limited compared to proprietary machines, it is also important to reduce the network activities between nodes since the overhead can easily become the bottleneck to overall performance. The second contribution is a novel technique to compress the conditional access sequence bases that does not only reduce the intercommunication but can also improve the performance of original WAP-mine.

Section 2 lists related works on web access pattern mining and its parallel executions. After explaining the task of web access pattern mining in section 3, the underlying serial WAP-mine algorithm is summarized in section 4. In section 5 we explain our approaches for parallel execution of WAP-mine on a

shared-nothing environment and we give the evaluation in section 6. Section 7 concludes the paper.

2 Related Works

Apriori was the first algorithm which addressed mining frequent pattern in 1995, particularly to generate association rules [2]. Many variants of Apriori based algorithms have been developed since then.

One of the Apriori extensions called AprioriAll also considers the time order constraint to mine sequential patterns [3]. A generalized version of the sequential pattern mining algorithm called GSP also has been devised [8]. GSP is faster than AprioriAll and can handle time constraints, a sliding time window, and a user-defined taxonomy.

Pioneering work on parallel algorithms for sequential pattern mining was done in [7]. It distributes sequence candidates among processing nodes to gain better memory utilization in a shared-nothing environment. A parallel algorithm for shared memory environment also has been proposed [9].

Some alternatives to Apriori-like "generate-and-test" paradigms, such as Tree Projection, were proposed [1]. However it was FP-growth that brought the momentum for the new generation of frequent pattern mining algorithms[5]. FP-growth outperforms both Apriori and TreeProjection.

Parallel sequential pattern mining based on TreeProjection has been implemented on a shared-nothing IBM SP2 parallel computer [4]. However it employs bread first search so it has to generate all sequences on a level of the tree before proceeding to next level. In addition every node has to keep the same tree structure consuming a lot of memory. An improvement to migrate some part of the tree is also proposed but the database has to be repartitioned and additional disk space is also required.

WAP-mine inherits a mining framework of FP-growth [6]. While other sequential pattern mining algorithm can also handle time ordered transaction data, WAP-mine is designed solely for web access pattern mining. However WAP-mine is reported to be a few times faster than Apriori based GSP. More description of WAP-mine will be given later.

3 Web Access Pattern Mining

Before we explain WAP-mine we will give some notions about web access pattern mining which given in [6].

In general, a web log can be regarded as a sequence of pairs of user identifier an event. Here we define a web access sequence as sequences of events and mine the sequential patterns over certain support threshold. For example, the web logs shown in Fig. 1 can be transformed into two web access sequences whose elements : $\{a, b\}$ and $\{a, c, a\}$.

Let E be a set of events. A web sequence $S = e_1e_2e_3 \dots e_n$ ($e_i \in E$) for ($1 \leq i \leq n$) is a sequence of events, while n is called the length of the access sequence. An access sequence with length n is also called n -sequence.

Access sequence $S' = e'_1e'_2e'_3 \dots e'_l$ is called a subsequence of access sequence $S = e_1e_2e_3 \dots e_n$ and S a super-sequence of S' , denoted by $S' \subseteq S$, if and only if there exist $1 \leq i_1 \leq i_2 \dots \leq i_l \leq n$, such that $e'_j = e_{i_j}$ for ($1 \leq j \leq l$). In addition, given sequence $S = e_1e_2e_3 \dots e_k e_{k+1} \dots e_n$, $S = e_{k+1} \dots e_n$ is the suffix of subsequence $S' = e'_1e'_2e'_3 \dots e'_l$ where $e'_l = e_k$ and $S = e_1 \dots e_k$ is the prefix of subsequence $S' = e'_1e'_2e'_3 \dots e'_l$ where $e'_1 = e_{k+1}$.

Please note that repetition of events is allowed in a sequence as shown in previous example : event a in sequence $\{a, c, a\}$ is repeated twice. However the support of event a here is one. When access sequence database $WAS = S_1, S_2, \dots, S_m$ is given, support of access sequence S is the number of supersequences of S in the access sequence database divided by the total number of sequence. The support is denoted $sup(S) = \frac{\|S_i \text{ subseteq } S\|}{m}$. In other word, any access pattern can get support at most once from one access sequence.

The problem of web access pattern mining is : given web access sequence database WAS and support threshold min_supp , mine the complete set of web access patterns of WAS .

```

218.XX.XXX.XXX - - [02/Jul/2002:11:23:36 +0900]
"GET /menu.html HTTP/1.1" 304 -
218.XX.XXX.XXX - - [02/Jul/2002:11:23:37 +0900]
"GET /Main.php3 HTTP/1.1" 200 4758
64.YYY.YY.YY - - [04/Jul/2002:08:20:56 +0900]
"GET /menu.html HTTP/1.1" 200 5019
64.YYY.YY.YY - - [04/Jul/2002:08:21:34 +0900]
"GET /Guestbook/guestbook.html HTTP/1.0" 200 1029
64.YYY.YY.YY - - [04/Jul/2002:08:22:46 +0900]
"GET /menu.html HTTP/1.1" 200 10854

```

Figure 1: Example of web logs

4 Serial WAP-mine Algorithm

Our parallel algorithm is based on WAP-mine [6]. The WAP-mine algorithm can be divided into two phases : the construction of the WAP-tree and mining web access patterns from the WAP-tree.

4.1 Construction of WAP-tree

The construction of the WAP-tree requires two scans of the access sequence database. The first scan accumulates the support of each event, that represents a distinct web page request, and then selects events that satisfy minimum support, i.e. frequent 1-sequences. The second scan constructs the WAP-tree.

First, non-frequent events are stripped off from the access sequences. In the WAP-tree, sequence with same prefix shares the same nodes. If the node corresponds to the events in access sequence exists, the count of the node is increased, otherwise a new node is generated and the count is set to 1.

The WAP-tree registers compactly access sequences and corresponding counts, all and only all information needed by the rest of mining.

The WAP-tree also has a frequent-event header table that holds the head of event-node queue, that connect nodes of same event in the WAP-tree. The event-node queues facilitate event traversal during mining of web access patterns. Note that the position of an event in the queue corresponds to its position in the sequences. Event in the end part of the sequence, comes first in the queue.

Figure 2 gives an illustration of WAP-tree generation from the database on each node.

4.2 WAP-mine

The philosophy of this mining algorithm is *conditional search*. Instead of searching pattern-level wise as Apriori, conditional search narrows the search space by looking for patterns with the same suffix, and count frequent events in the set of prefixes with respect to condition as suffix.

Input to the WAP-mine algorithm is the WAP-tree and the minimum support. To find all access patterns whose support are higher than the minimum support, WAP-mine traverses nodes in the WAP-tree. The event-node queue originating from each event in the frequent-event header table connects to the same event in the WAP-tree. The queue also guarantees that the suffixes of the sequences are processed first.

While visiting each node, WAP-mine also collects the prefix-sequence of the node, that is the set of events on the path from the node to the root of the tree. WAP-mine also stores the count of the prefix-sequence on the node. The prefix-sequences form the so called *conditional access sequence base* of that event.

In a web access sequence there are many occasions where a same page is visited more than once. For

example in the sequence *abacad*, the page *a* is visited three times. However the support of the event in the sequence is only one. To avoid double counting, WAP-mine devises *unsubsumed count property*.

When G and H are two prefix sequences of suffix event a , and G is also formed by the subpath from root of that H is formed by, H is called a *super-prefix sequence* of G and G is a *sub-prefix sequence* of H . For instance $\{a, c, a\}$ is a super-prefix sequence of $\{a\}$. For a prefix sequence of a with some super-prefix sequences, the unsubsumed count of it is the count of that sequence minus unsubsumed counts of all its super-prefix sequences.

For each prefix sequence of event a with count c , when it is inserted into a -conditional access sequence base, all of its sub-prefix sequences of a are inserted also but with count $-c$.

Take the WAP-tree of Node1 at Fig. 2 as an example. WAP-mine extracts the conditional access pattern base of m from the WAP-tree as : $mam : 1$, $ma : -1$, $mamc : 1$, $ma : -1$, $ma : 2$.

The conditional access sequence base is a small database of sequences which co-occur with the event. Then WAP-mine create a small WAP-tree from the conditional access sequence base called *conditional WAP-tree*. The process is recursively iterated until no conditional access sequence base can be generated and all web access patterns that consist the event are discovered.

The same iterative process are repeated for other frequent events.

5 Parallel execution of WAP-mine

Since the processing of a conditional access sequence base is independent of the processing of other conditional access sequence bases as shown by Lemma 1, it is natural to consider it as the execution unit for the parallel processing.

Lemma 1 *The processing of a conditional access sequence base is independent from other conditional access sequence base.*

Proof From the definition, event a 's conditional access sequence base generation is only determined by existence of the event a in the sequence database. The conditional access sequence base is generated by traversing the event-node queue. The resulting conditional access sequence base is unaffected by event-node queues of other events since event-node queue

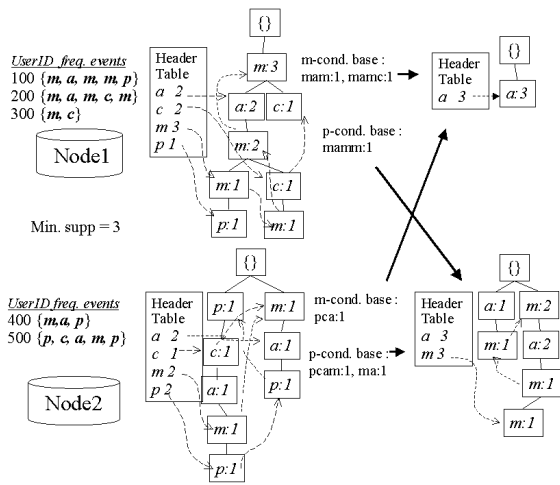


Figure 2: Illustration of parallel execution

connects nodes of same event only and there is no node deletion during the processing of the WAP-tree.

Here we describe a parallel version of WAP-mine. We assume that the access sequence database is distributed evenly among the nodes.

5.1 Parallelization Scheme

The basic idea is that each node accumulates a complete conditional access sequence base and processes it independently to the completion before receiving other conditional access sequence bases.

Pseudocode for this algorithm during the exchange phase is depicted in Fig. 3 and the illustration is given in Fig. 2.

After the first scan of the access sequence database the support count of all events are exchanged to determine globally frequent events. During the second database scan, each node builds a local WAP-tree from the local access sequence database.

From the local WAP-tree, local conditional access sequence bases are generated. We use hash function to determine which node should process it, instead of processing conditional access sequence base locally. We can do this because of the following lemma :

Lemma 2 *Accumulation of local conditional access sequence base results in a global conditional WAP-tree.*

Proof When a prefix-path in the global conditional WAP-tree already exists, the counts of prefix-paths in the other conditional access sequence bases are simply added to the path in the tree. Thus the final global conditional WAP-tree is not affected by how

input : database D, events I,
minimum support min_supp;

```

SEND process :
{
1:local_support = get_support(D,I);
2:global_support = exchange_support(local_support);
3:WAPtree = construct_waptree(D);

;exchange conditional access sequence base
4:forall event do begin
5:  cond_sbase = build_sequence_base(WAPtree, event);
6:  dest_node = event mod num_nodes;
7:  send_sequence_base(dest_node, cond_sbase);
8:end
}

RECV process :
{
1:cond_sbase = collect_sequence_base();
2:cond_WAPtree = construct_waptree(cond_sbase);

3:WAP-mine(cond_WAPtree, NULL);
}

```

Figure 3: Pseudo code of parallel execution

the prefix-paths are contained in the conditional access sequence bases.

5.2 Intercommunication Reduction

Intercommunication is essential in parallel processing since processing nodes have to coordinate and exchange data. The two main elements for an efficient parallel code are load balancing and efficient communication among processors. In particular, PC cluster as well as grid configuration of PCs that use commodity network are known to have bigger latency and limited bandwidth compared to proprietary parallel computer. Thus it is important to keep intercommunication between nodes as small as possible.

WAP-mine utilizes the property of unsubsumed count to avoid double counting of sequences. To implement the property, WAP-mine includes also additional sub-prefix sequences when generating conditional access sequence bases.

Although unsubsumed count property correctly counts the support of events in the sequences, the size of conditional access sequence base increases linearly with the number of the event duplications in the sequences. Since our parallelization scheme exchanges the conditional access sequence bases between nodes, a lot of data has to be sent through the network.

To reduce the size of conditional access sequence bases, while traversing the path from the node to the root to collect the prefix-sequence of event a with count c , we simply decrement the count of the nodes with c directly if we encounter the same event a on the path. Thus we do not have to generate the sub-

prefix sequences. In addition, if the count of a node becomes zero, we do not have to generate the conditional access sequence base starting from the node.

For example, the conditional access pattern base of m from the WAP-tree of Node1 at Fig. 2 only contains two prefix-sequences : $mam : 1, mame : 1$. We do not need to generate ma sequences since the count of third node in the m -queue has become zero.

It is easy to show that our method also correctly results unsubsumed counts of sequences. By definition, unsubsumed count of a sequence is obtained by reducing the count of that sequence minus by the counts of all its super-prefix sequences. Since the event-node queue link the suffix part first, the super-prefix sequences are processed before their sub-prefix counterparts. Therefore we can simply reduce the count of their sub-prefix sequences found along the way, instead of materializing all the super-prefix sequences. Since it only alters the count of the same event, it is obvious that the direct modification of the WAP-tree does not affect the result of subsequent processing.

Our method significantly saves both time to generate conditional access sequence base and the required network bandwidth. Here we give simple analysis of the efficiency gained by our method.

Let the number of event duplications in a sequence as dup and its average ratio as dup_{avg} . Note that we can calculate the ratio by dividing the total occurrences of the duplicates by the number of sequences. For instance, in the sequence database shown in Fig. 2, there are two duplicates of event m in each sequence with user ID 100 and 200. Since there are totally five duplicates in the database of five sequences, the dup_{avg} of the database is one.

We also denote the compression rate of frequent sequences in the WAP-tree as $compress_{WAP}$. The more similar sequences are in the database, the WAP-tree is more compact and the compression rate is higher.

Original WAP-mine also generates all sub-prefix sequences when processing a suffix event. When the suffix event has dup duplicates on the path to the root, dup subsequences are also generated. When those duplicates become suffix events, they also generate their own sub-prefix sequences. There are $(dup + 1) + (dup) + (dup - 1) + (dup - 2) + \dots + 1 = \frac{(dup+1)+1}{2}(dup+1)$ subsequences. Therefore WAP-mine generates approximately $m(dup_{avg} + 1)^2 compress_{WAP}$ subsequences into the conditional access sequence bases. Here $m = \|WAS\|$ is the number of sequences in the database. Since our parallel WAP-mine does not need to generate those sub-prefix sequences, the amount of communication re-

quired to exchange conditional access sequence base is $(dup_{avg} + 1)^2$ times smaller.

After decremented by the counts of all its super-prefix sequences, some nodes will lose all its count. In other word, some nodes will have zero count. Since we also do not have to examine the paths starting from nodes with zero count, significant time and space saving can be expected. It is easy to show that the saving is proportional with dup_{avg} .

6 Implementation and Performance Evaluation

6.1 Implementation

As the shared nothing environment for this experiment we use a PC cluster of 15 nodes that are interconnected by a 100Base-TX Ethernet Switch. Each PC node runs the Solaris 8 operating system on a 1.5GHz Pentium4 with 384 MB of main memory.

Three processes are running on each node :

1. SEND process
creates the WAP-tree, and sends the conditional access sequence bases
2. RECV process
receives the conditional access sequence bases, and processes the conditional access sequence bases after the exchange phase finishes
3. EXEC process
processes the conditional access sequence base in background during the exchange phase

There are also small COORD processes that receive requests for the conditional access sequence base from idle RECV processes. After the exchange phase, the COORD processes coordinate how to distribute the conditional access sequence base in order to balance the processing load. The strategy is on first come first serve basis.

6.2 Performance Evaluation

For performance evaluation, we use a synthetically generated dataset as described in the AprioriAll paper [3]. In this dataset, the average access sequence size and average maximal potentially frequent sequence size are set to 20 and 10 respectively. While the number of access sequences in the dataset is set to one million with 10K events.

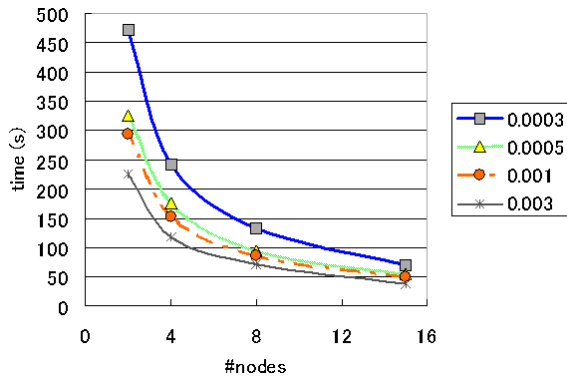


Figure 4: Execution time

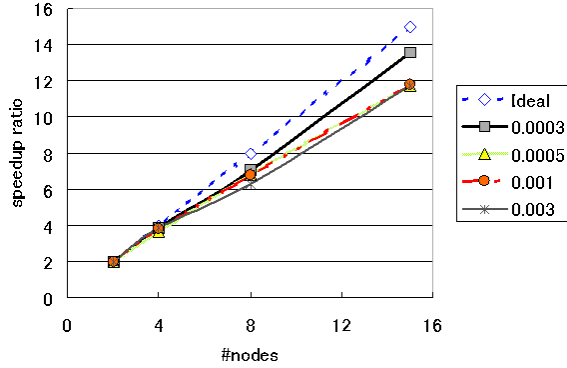


Figure 5: Speedup ratio

6.2.1 Execution Time and Speedup Ratio

We have varied the minimum support to see how it affects performance. The experiments are conducted on 2, 4, 8 and 15 nodes. The execution time for minimum support of 0.0003%, 0.00005%, 0.001% and 0.003% is shown in Fig. 4. The execution time of single node is excluded because memory trashing occurs on single node.

To get a better understanding of the performance evaluation, we also measure speedup ratio. The speedup ratio is the ratio of performance gain when more processing nodes are used.

Fig. 5 shows that a good speedup ratio is achieved for all minimum support. When the minimum support is set to 0.0003%, 15 processing nodes are 13.6 times faster in performance. Our algorithm suitably balances the processing load and keeps the intercommunication overhead low. When the minimum support is lower, the overhead is relatively smaller, thus the speedup is improved.

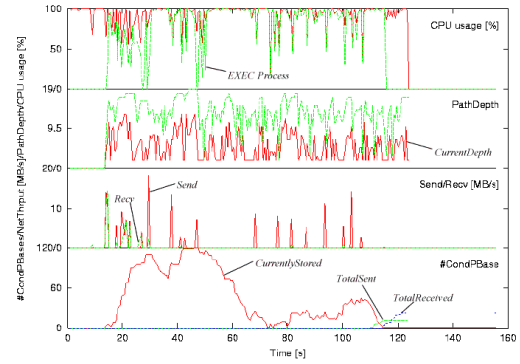


Figure 6: Execution trace (min. support 0.0005% 15nodes)

6.2.2 Execution trace

We have developed a tool to monitor the parallel execution on the PC cluster. The execution trace of a node during the execution for a minimum support of 0.0005% on 15 nodes is shown in Fig. 6. The figure shows CPU resource usage, path-depth, interconnection network (send/receive) and also the number of conditional pattern bases. The horizontal axis is elapsed time. The vertical axis for the top graph denotes the CPU utilization ratio for the overall process on the node and the EXEC process alone. The second graph denotes the path depth of the conditional access sequence base and current processing pattern length. The third graph denotes data transfer throughput in MB/s for an interconnection network. The network throughput is divided into two parts, send throughput and receive throughput. The fourth graph shows the number of conditional access sequence bases. There are three kinds of information: those currently stored in the node, total sent to other nodes, total received from other nodes.

One can observe that the background EXEC process fills the CPU idle time when the nodes are exchanging the conditional access sequence bases. Overall the processing is CPU bound, which explains why adding more processing nodes results in faster performance. We can also confirm that the network overhead is small.

7 Conclusion

The divide-and-conquer paradigm has become the main drive of modern high performance frequent pat-

tern mining algorithms. The mining of web access patterns is one kind of the frequent pattern which has become a hot research topic. Many applications need to mine large scale web logs in timely manner. WAP-mine employs a divide-and-conquer paradigm to efficiently mine web access patterns from compact database representation called WAP-tree. Although the data structure is complex and naturally not suitable for parallel processing on shared-nothing environment, we have developed a parallel version of WAP-mine that showed good speedup even on commodity PC cluster. The novel way to compress conditional access sequence base becomes the key contribution to reduce the network load among nodes.

More improvements are under way. Although simple round-robin like distribution of access sequence bases works well for many datasets, better resource-aware distribution is needed. We are examining two approaches. The first the method to balance the memory consumption of WAP-tree among nodes. The second one is asynchronous demand-based exchange phase. The approach is also important in heterogenous environment such as grid configuration.

References

- [1] R. Agarwal, C. Aggarwal and V.V.V. Prasad "A Tree Projection Algorithm for Generation of Frequent Itemsets". In *J. Parallel and Distributed Computing*, 2000
- [2] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules". In *Proc. of the 20th International Conference on VLDB*, pp. 487–499, September 1994.
- [3] R. Agrawal and R. Srikant. "Mining Sequential Patterns". In *Proc. of International Conference of Data Engineering*, pp. 3–14, March 1995.
- [4] V. Guralnik, N. Garg, and G. Karypis "Parallel Tree Projection Algorithm for Sequence Mining" In *Proc. of Europar*, 2001
- [5] J. Han, J. Pei and Y. Yin "Mining Frequent Pattern without Candidate Generation" In *Proc. of the ACM SIGMOD Conference on Management of Data*, 2000
- [6] J. Pei, J. Han, B. Mortazavi-asl and H. Zhu "Mining Access Patterns Efficiently from Web Logs" In *Proc. of fourth Pacific-Asia Conference in Knowledge Discovery and Data Mining(PAKDD'00)*, 2000.
- [7] T. Shintani, M. Kitsuregawa "Mining Algorithms for Sequential Patterns in Parallel: Hash Based Approach". In *Proc. of second Pacific-Asia Conference in Knowledge Discovery and Data Mining(PAKDD'98)*, pp. 283–294, 1998.
- [8] R. Srikant, R. Agrawal. "Mining Sequential Patterns: Generalizations and Performance Improvements". In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, , March 1996.
- [9] M. J. Zaki "Parallel Sequence Mining on Shared-memory Machines". In *J. of Parallel and Distributed Computing* 61(3):401-426, March 2001.