

Parallel Web Access Pattern Mining 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

Abstract *Mining web access patterns from web logs is an extensive task that requires a lot of computational resources. However, only a little research addresses the parallelization of such a task. Here we propose a parallel web access pattern mining system on a PC cluster. The system is designed to be adaptable, scalable, and able to offer timely response. Our parallel algorithm adopts a pattern-growth paradigm that has become a characteristic of modern frequent pattern mining algorithms. We also examine techniques to reduce the intercommunication necessary to gain a sufficient speedup ratio.*

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

1 Introduction

Mining access patterns from Web logs has become fundamental to many applications such as business intelligence, recommendation, personalization and intelligent prefetching. Such applications often require fast a response time, while the web logs that have to be processed are accumulating at a tremendous pace. Major portals accumulate visitor access logs of a few GBs per day, but mostly only for limited use because of their size. The need of faster algorithms with better scalability makes the mining of web access patterns an ideal candidate for parallel processing.

Essentially, a web access pattern is a sequential pattern in a large set of parts of a web log, which is pursued frequently by users. Sequential pattern mining is defined by Agrawal and Srikant as follows: given a sequence database

where each sequence is a list of transactions ordered by transaction time and each transaction consists of a set of items, find all sequential patterns with a user-specified minimum support, where the support is the number of data sequences that contain the pattern [3].

Most of the algorithms used to mine sequential patterns such as web access patterns, are derivations of an Apriori like level-wise generate-and-test method. They encounter the difficulty when the length of the pattern grows long, which is exactly the case in web access pattern mining.

An algorithm to mine frequent patterns called FP-growth has recently motivated the emergence of the pattern-growth paradigm [5]. The compression of transaction databases into an on-memory data structure called a FP-tree benefits FP-growth with performance that is better than that previously reported by algorithms such as Apriori [2]. Here our parallel algorithm is based on WAP-mine. WAP-mine also uses a data structure called WAP-tree to register access sequences and corresponding counts compactly [7].

Further performance improvement can be expected from parallel execution. A parallel engine is essential for a large-scale data warehouse. Particularly, the development of parallel algorithms on a large scale shared nothing environment such as a PC cluster has attracted a lot of attention, since it is a promising platform for high performance data mining.

However, a parallel algorithm for a complex data structure like the WAP-tree is much harder to implement compared to a sequential program or shared-memory parallel system. Here we propose a framework for web

log mining system based on PC cluster and develop a parallel algorithm to mine web access patterns on the framework.

Section 2 lists related works on web access pattern mining and its parallel execution. Section 3 briefly describes the web mining system on a PC cluster. In this paper, we are focusing on the performance of our system to mine web access patterns. The underlying sequential WAP-mine algorithm is summarized in section 4. In section 5 we explain our approach 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 that addressed mining frequent patterns in 1995 [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 [10]. GSP is faster than Apriori-All 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 [9]. A parallel algorithm for a shared memory environment also has been proposed [11].

Some alternatives to Apriori-like "generate-and-test" paradigms, such as TreeProjection, 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 breadth 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 required.

WAP-mine inherits a mining framework of FP-growth [7]. While other sequential pattern mining algorithms 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 PC Cluster Based Web Log Mining System

The objective of our system is the development of a prototype of a very large-scale platform for integrated web access log mining. In particular the system should be able to cope with dynamic web pages that have already become the mainstream for large sites. Sites with dynamic web pages usually employ CGI scripts to get data from backbone databases. Sometime additional application servers are used to generate the dynamic web pages. Dynamic web pages add a new dimension to log mining since there are virtually an infinite number of URL variations.

The architecture of the system is depicted in Fig. 1. To achieve the objective, the platform has to fulfill several requirements.

1. **Adaptability**
The ability to adapt to various CGI formats and various mining algorithms. A metadata based CGI parameter extractor translates web requests into a format that is easier to manage by the weblog warehouse and mining engine. The weblog warehouse stores not only the access sequences but also other site dependent attributes such as search keywords and user identity.
2. **Scalability**
The ability to handle large-scale web logs

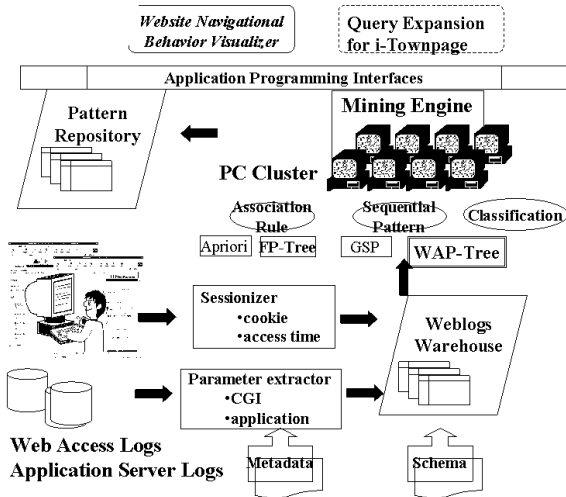


Figure 1: Web log mining system on PC cluster

with sufficient performance by parallel execution. PC cluster is a promising platform for this purpose because of its cost performance and expandable configuration. Parallel algorithms for the shared nothing environment are developed to exploit the potential.

3. Timely analysis

Timely response provided by intelligent management of mining results. Since usually mining from scratch is costly, a pattern repository is used to make optimal reuse of the results.

On top of the system, we develop applications such as NAVIZ(Website Navigational Behavior Visualizer) [8] and Query Expansion recommendation system [6]. A snapshot of NAVIZ is given in Fig. 2.

4 Serial Web Access Pattern Mining

Our parallel algorithm is based on WAP-mine [7]. The WAP-mine algorithm can be divided into two phases : the construction of WAP-tree and mining web access patterns from WAP-tree.

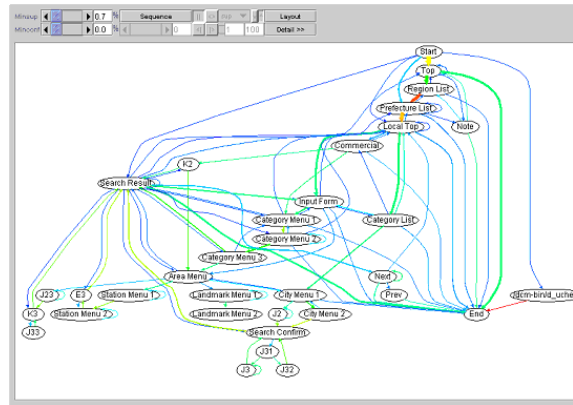


Figure 2: NAVIZ : website navigational behavior visualizer

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 the count of the node is increased, otherwise a new node is generated and the count is set to 1.

The WAP-tree also has a frequent-event header table that holds the head of the event-node queues, which connect nodes of the same event in the WAP-tree. The event-node queues facilitate event traversal during mining of web access patterns.

4.2 WAP-mine

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 starting from the least event in the frequent-event header table. The

node-link originating from each event in the frequent-event header table connects to the same event in WAP-tree.

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 node as the count of the prefix-sequence. The prefix-sequences form the so called *conditional access sequence base* of that event.

The conditional access sequence base is a small database of sequences that co-occur with the event. Then WAP-mine create small WAP-tree from the conditional access sequence base called the *conditional WAP-tree*. The process is recursively iterated until no conditional access sequence base can be generated and all web access patterns that contain 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 base, it is natural to consider it as the execution unit for the parallel processing.

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 base.

Pseudocode for this algorithm is depicted in Fig. 3 and the illustration is given in Fig. 4.

After the first scan of the access sequence database the support count of all events

```

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

```

Figure 3: Pseudo code of parallel execution

are compared 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 a hash function to determine which node should process it, instead of processing the conditional access sequence base locally. We can do this because the accumulation of local conditional access sequence base results in a global conditional WAP-tree.

5.2 Intercommunication Reduction

Intercommunication is essential in parallel processing since processing nodes have to coordinate and exchange data. However on a PC cluster, with limited bandwidth and network latency compared to a proprietary machine, it

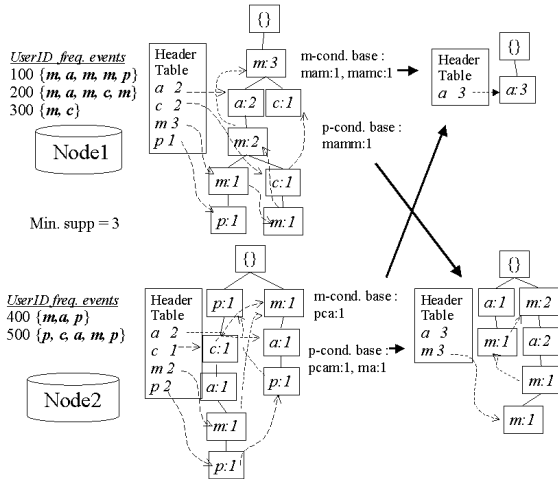


Figure 4: Illustration of parallel execution

is also important to reduce the network activities between nodes since the overhead can easily become the bottleneck to overall performance.

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*. 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*.

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

To reduce the size of conditional access sequence bases, we use a direct count decrement method. While traversing the path from the node to the root in order to collect the prefix-sequence of event *a* with count *c*, we simply decrement *c* from the count of the duplicate

nodes directly. . The duplicate node here is the node on the path with the same event label *a*. Thus we do not have to generate the sub-prefix sequences. In addition, if the count of a duplicate node becomes zero, we do not have to generate the conditional access sequence base starting from the node.

Our method significantly saves both time to generate conditional access sequence base and the required network bandwidth. We omit the proof of the correctness and efficiency analysis here due to the space limitation.

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 interconnected by 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 base
2. RECV process
receives the conditional access sequence base, and processes the conditional access sequence base after the exchanging phase finishes
3. EXEC process
processes the conditional access sequence base in the background during the exchanging phase

There are also small COORD processes that receive requests for the conditional access sequence bases from idle nodes and coordinate how to distribute them.

6.2 Performance Evaluation

For the performance evaluation, we use a synthetically generated dataset as described in

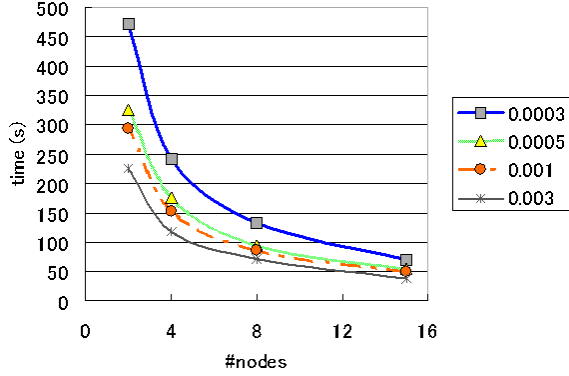


Figure 5: Execution time

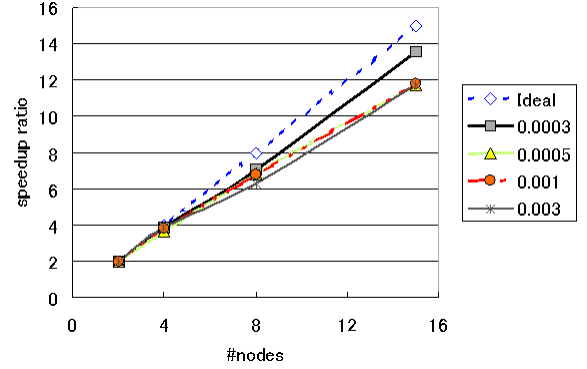


Figure 6: Speedup ratio

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 one million with 10K events.

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%, 0.0005%, 0.001% and 0.003% is shown in Fig. 5. It shows that even for such low minimum support, our system can offer fast response time.

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

Fig. 6 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 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.

6.2.2 Execution trace

We have developed a tool to monitor the parallel execution on the PC cluster. The execution trace of parallel WAP-mine on 15 nodes for minimum support 0.0005% is shown in Fig. 7. The figure shows CPU resource usage, path-depth, interconnection network (send/receive) and also the number of conditional access sequence bases. The horizontal axis is elapsed time. The vertical axis for the top graph denotes CPU utilization ratio for the overall process and EXEC process. 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 kind of such 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.

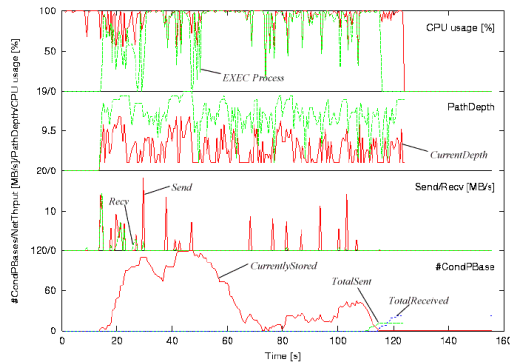


Figure 7: Execution trace (min. support 0.0005% 15nodes)

7 Conclusion

We have proposed a framework for parallel web log mining on PC cluster. We have also reported the development of parallel WAP-mine that is designed to run on a shared-nothing environment. The algorithm has been implemented on top of the PC cluster system with 15 nodes. We have also introduced a novel compression method of conditional access sequence bases to reduce the intercommunication.

Although the data structure of WAP-tree is complex and naturally not suitable for parallel processing on a shared-nothing environment, the experiments show our algorithm can achieve reasonably good speedup ratio.

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] Y. Ohura, K. Takahashi, I. Pramudiono, and M. Kitsuregawa "Experiments on Query Expansion for Internet Yellow Page Services Using Web Log Mining". In *Proc. of the 20th International Conference on VLDB*, 2002.
- [7] 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.
- [8] B. Prasetyo, I. Pramudiono, K. Takahashi, and M. Kitsuregawa "Naviz: Website Navigational Behavior Visualizer" In *Proc. of sixth Pacific-Asia Conference in Knowledge Discovery and Data Mining(PAKDD'02)*, 2002.
- [9] 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.
- [10] 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.
- [11] M. J. Zaki "Parallel Sequence Mining on Shared-memory Machines". In *J. of Parallel and Distributed Computing* 61(3):401-426, March 2001.