Peer-to-Peer Usage Analysis: a Distributed Mining Approach

Florent Masseglia
INRIA Sophia Antipolis, Axis Project-Team
BP93 06802 Sophia Antipolis - France
Florent.Masseglia@sophia.inria.fr

Pascal Poncelet
EMA-LGI2P/Site EERIE
Parc Scientifique Georges Besse, 30035 Nîmes Cedex 1 - France
Pascal.Poncelet@ema.fr

Maguelonne Teisseire
LIRMM UMR CNRS 5506
161 Rue Ada 34392 Montpellier Cedex 5 - France
teisseire@lirmm.fr

Abstract

With the huge number of information sources available on the Internet, Peer-to-Peer (P2P) systems offer a novel kind of system architecture providing the large-scale community with applications for file sharing, distributed file systems, distributed computing, messaging and real-time communication. P2P applications also provide a good infrastructure for data and compute intensive operations such as data mining. In this paper we propose a new approach for improving resource searching in a dynamic and distributed database such as an unstructured P2P system. This approach takes advantage of data mining techniques. By using a genetic-inspired algorithm, we propose to extract patterns or relationships occurring in a large number of nodes. Such a knowledge is very useful for proposing the user with often downloaded or requested files according to a majority of behaviors. It may also be useful in order to avoid extra bandwidth consumption.

1 Introduction

With the huge number of information sources available in the Internet and the high dynamics of their data, Peer-to-Peer (P2P) systems offer a novel kind of system architecture providing the large-scale community behavior with applications for file sharing, distributed file systems, distributed computing, messaging and real-time communication [6].

P2P application also provides a good infrastructure for data and compute intensive operations such as data mining. For instance, it may be discovered, in a P2P file sharing network, such as Gnutella [3], that "Mandriva Linux 2005" distribution is often downloaded as "CD1.iso, then CD2.iso and finally CD3.iso". Such an approach is considered in [10] where the authors propose to mine association rules, i.e. sets of objects which tend to associate with one another, in a P2P system. The proposed algorithm combines locally, i.e. at each node, association rule mining algorithm with a majority voting protocol to discover at each node all of the association rules that exist in the distributed database.

In this paper we consider a new approach for improving resource searching in a dynamic and distributed database such as an unstructured P2P system and we consider two new concepts for discovering frequent behaviors among users of such a system: (i) The *sequential order* between the actions performed on each node (requests or downloads) has to be taken into account for better results. (ii) Maintaining, on a *"meter peer"*, the global result of distributed calculations will considerably reduce the amount of communications between connected peers.
First, we argue that considering the timestamps of each action performed on nodes could be very informative. For instance, by examining the frequent actions performed, we can obtain that for 77% of nodes from which a request is sent for *"Mandriva Linux*", the file *"Mandriva Linux 2005 CD1 i585-Limited-Edition-Mini.iso*" is chosen and downloaded, then a new request is performed with the

possible name of the remaining iso images (*i.e "Mandriva Linux 2005 Limited Edition*") and in the large number of returned results the image corresponding to "*Mandriva Linux 2005 CD2 i585-Limited-Edition-Mini.iso*" is chosen and downloaded. Such a knowledge is very useful for proposing the user with often downloaded or requested files according to a majority of behaviors. Proposing the user with often downloaded or requested files could also be useful in order to avoid extra bandwidth consumption. In our previous example, two large broadcasting operations are performed: first when requesting a specific resource ("*Mandriva Linux*") and second when requesting the remaining iso images ("*Mandriva Linux 2005 CD2*"). If we know that a large number of nodes interested by Linux distributions will download the corresponding iso images with a special format, it is easy to improve the first request. In our example, a user searching for the Mandriva distribution ("*Mandriva Linux*") would be answered with the usual files and an additive answer containing the iso images ("*Mandriva Linux 2005 CD* i585-Limited-Edition-Mini.iso*"). Such request modification can thus enhance the resource location as well as avoid the second broadcast.

Second, mining either association rules or sequential patterns in such a very large distributed database than an unstructured P2P system is far away from trivial. By nature such systems are dynamic, i.e. nodes act independently of one another, and Indeed as nodes must act independently of one another, intermediate results may be overturned as new data arrives. Furthermore, whenever a node departs, the sequence of that node also disappears, and the global database has to be reconsidered for taking into account and propagating such a departure. Traditional approaches for mining sequential patterns [9, 7] are irrelevant in such a dynamic context because they consider that the whole database is available. In the literature, incremental approaches were proposed [4, 1] but they suffer the same limitations. They mainly consider that new clients or new items are added to the original database. In [5], we proposed to discover relationships and global patterns that exist between connecting users (Web Usage Mining). This approach was mainly defined in order to take advantage of the computing power available on the machine a user navigates with. We proposed a new "client/server/engine" architecture for taking advantage of this computing power. In this paper our goal is different since it takes into account the dynamic nature of the considered system. We consider that the connected nodes can act with a special peer (a "*meter peer*") in order to provide the end user with a good approximation of patterns embedded in this very large distributed database.

The rest of the paper is organized as follows. Section 2 goes deeper into presenting the problems of sequential patterns in a very large distributed database. In Section 3 we propose our approach for mining sequential patterns in an efficient way. Section 4 reports the result of our experiments. In Section 5, we summarize our findings and conclude the paper with future avenues for research.

2 Problem Statement

This section is devoted to stating the problem and extend the initial sequential mining problem [9] by considering a large-scale distributed database.

Let $I = \{x_1, \ldots, x_n\}$ be a set of distinct literals called items. I is called an itemset. In the following we assume that for each item we are provided with the action performed, i.e. request or download. An item x_i is thus denoted either $[d, x_i]$ for downloading or $[r, x_i]$ for requesting.

A *sequence* is an ordered list of itemsets denoted by $< s_1 s_2 \ldots s_n >$ where s_j is an itemset. For instance, let us consider the following actions performed on a node S=$<$ $([d,3])$ $([d,4]$ $[d,5])$ $([d,8])$ $>$. S means that 3 was first downloaded, then 4 and 5 at the same moment (i.e. in the same transaction) and finally 8.

Let D be a database of customer *data-sequences*. The $support$ for a sequence s, also called $supp(s)$, is defined as the fraction of total data-sequences that contain s. If $supp(s) \geq minsupp$, with a minimum support value $minsupp$ given by the user, s is considered as a *frequent* sequential pattern.

Here we adapt the problem statement proposed by [10] to our concern. When the database is dynamically updated, i.e. transactions are added to it or deleted from it over time, we denote D_t the database at time t. Let us assume that the database is also partitioned among an unknown number of nodes. We denote such a partition of node u, at time t, D_t^u. In fact D_t^u corresponds to the sequence of actions performed on the node. Let us assume that $D_t = \bigcup D_t^u$ ... D_t^v, where u_t ... v_t are the available nodes at time t. The problem of sequential pattern mining in such a large-scale distributed systems D_t is thus to find the set of frequent sequential patterns in D_t according to the $minsupp$ value.

Let us consider that F_{D_t} is the result to obtain (the result that would be exhibited by an algorithm which would explore the whole set of solutions), F_{D_t} is thus the set of frequent sequential patterns to find in D_t. Let us now consider $\tilde{F_{D_t}}$, the set of approximate sequential patterns. We require that, as nodes u_t ... v_t are dynamics, $\tilde{F_{D_t}}$ will converge as fast as possible to F_{D_t}.

Example 1 *Let us consider three sequences standing for downloading operations performed on nodes u_t, v_t and w_t:*

D_{u_t}	$<([d,1])([d,2])([d,3])([d,4])([d,5])>$
D_{v_t}	$<([d,1])([d,2])([d,1])([d,3])([d,5])>$
D_{w_t}	$<([d,1])([d,2])([d,4])([d,5])([d,6])>$

From such sequences, the set of items with their associated support is the following: ($[d,1]$)[100%], ($[d,2]$) [100%], ($[d,3]$) [66%], ($[d,4]$) [66%], ($[d,5]$) [100%] and ($[d,6]$) [33%]. Let us assume that a support value, minsupp, is set to 100%, then the set of frequent sequences at time t_i on $F_{D_{t_i}}$ is: $F_{D_{t_i}}$ =$\{<([d,1])\ ([d,2])\ ([d,5])>\}$. Let us now assume, at time t_{i+1} that the node w_t departs, then the set of frequent sequences becomes $F_{D_{t_{i+1}}}$ =$\{<\ ([d,1])\ ([d,2])\ ([d,3])\ ([d,5])\ >\}$ since the support of the item ($[d,3]$) is now 100%.

3 A new heuristic approach for mining sequential patterns in Peer-to-Peer systems

As a matter of fact, the nodes in a P2P context may connect or depart frequently while D_t is still being analyzed. Our proposal is to consider D_t as a unit able to receive candidate sequences, to evaluate the support of each candidate on sequence in D_t and to send back the result. This kind of "scan", distributed on all the connected nodes relies on a stochastic algorithm for combinatorial optimization problems. We first give an overview of our approach and our algorithms are then described.

First D_t is empty until a node u_t sends its sequence. The unstructured P2P architecture we propose allows a special peer (hereafter the "$Distributed_{SP}$ peer") to get connected to each new peer arriving on the network (let us consider the peer "v" in algorithm 2, then the instruction *send(v,@$Distributed_{SP}$)* allows $Distributed_{SP}$ to be aware of v's arrival). Our method then relies on a distribution of the candidate sequences, as described in Figure 1. The "$Distributed_{SP}$ peer" performs the instructions of the $Distributed_{SP}$ heuristic, from "GetValuation" to "Broadcast". At first, the set of frequent items is extracted from the connected peers (a mere counting for each peer is sufficient). Then the whole set of candidates having length 2 is generated from the set of frequent items. The candidates get evaluated by the connected peers ($u_t..v_t$) in order to count those having the required threshold in the whole database, i.e. on $D_t = \bigcup D_t^u$ ... D_t^v. The results are collected by the $Distributed_{SP}$ peer (i.e. the "GetValuation" function). Then the heuristic is applied (based on genetic operators) and the new set of candidates is sent to the connected peers for evaluation. This process is repeated until no change enhances the solution (As data evolves each time and as nodes are dynamic, each change might enhance the solution and the solution will thus follow the nodes behavior on and on...).

Now let us consider the $Distributed_{SP}$ algorithm (Cf. Algorithm 1). $Distributed_{SP}$ first starts when a node u_t joins our system ($(recvu_t)$). The set of frequent patterns is thus initialized with the sequence of u_t. While nodes are available, we consider patterns sent to $Distributed_{SP}$ by the $getValuation$ function in order to test if sequences are frequents. SCORE stands for the average marks given by all nodes for candidates. If SCORE is greater or equal to a minimum support value, this candidate becomes frequent. Nevertheless, sequences are not only evaluated as included or not, we also consider the distance between the size of the candidate's subset included, compared to the total size of candidates (C.f Algorithm 2). The aim of this approach is to take into account that a sequence could be unfrequent but could be useful for generating candidates. For efficiency reasons, i.e. in order to avoid to generate too much candidates, we consider a user defined threshold $mindist$. When the score of a sequence is greater than or equal to mindist, then this sequence will be inserted into the approximate set in order to be considered for the generating phase. Thanks to frequent patterns, approximate sequences and neighborhood operators, candidates are generated and sent to all connected nodes by broadcasting in order to know either their threshold or their distance from a frequent sequence.

Algorithm 1: $Distributed_{SP}$ algorithm

Data: F_{D_t} the frequent sequences for node u_t, nodes u_t ...v_t on line and a user-defined parameter $mindist$ standing for the minimum distance for a candidate to be considered.
Result: F_{D_t} the sequential patterns corresponding to the frequent behavior patterns on D_t.

```
// Initialization
recv (u_t);
F_{D_t} ← D_t^u;
// main process
while nodes are on line do
    nodesAvailable ← recv(u_t...v_t);
    F̃_{D_t} ← ∅;
    candidates ← getValuation(nodesAvailable);
    for c ∈ candidates do
        if SCORE(c) > minsupp then
            insert(c, F_{D_t});
        else
            if SCORE(c)) ≥ mindist then
                insert(c, F̃_{D_t});
    candidates ← neighborhoodOperators(F̃_{D_t}, F_{D_t});
    Broadcast(@nodesAvailable, candidates);
```

As they proved to be efficient in [5], for neighborhood operators we propose "Genetic-like" operators as well as operators based on sequential patterns properties. When we talk about random sequence, we use a biased random such that sequences having a high threshold may be chosen before sequences having a low threshold. In the following,

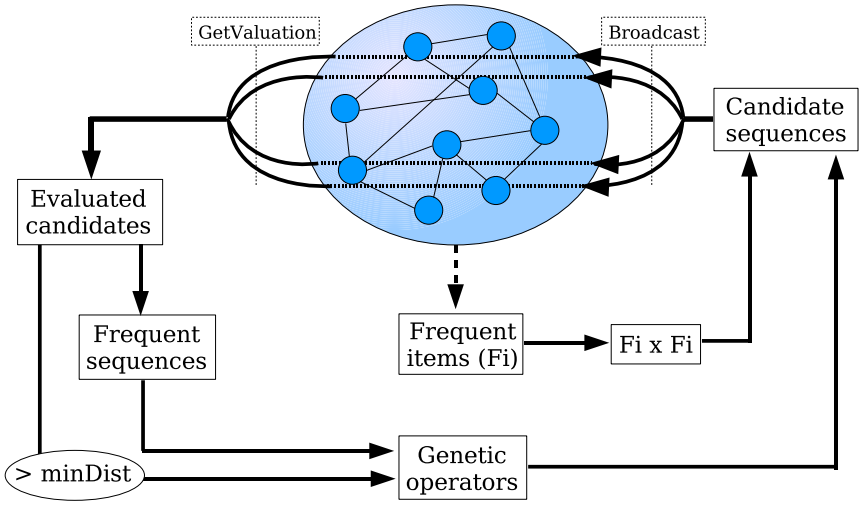

Figure 1. Overview of the $Distributed_{SP}$ heuristic

for brevity, we note an item x_i without considering its associated operation (download or request).
New frequent items: When a new frequent item is occurring (after being requested by one or more users) it is used to generate all possible 2-candidate sequences with other frequent items. The candidates set generated is thus added to the global candidates set.
Adding items: This operator aims at choosing a random item among frequent items and adding this item to a random sequence s, after each item in s. This operator generates $length(s) + 1$ candidate sequences.
Basic crossover: This operator uses two different random sequences and proposes two new candidates coming from their amalgamation.
Enhanced crossover: This operator aims at choosing two random sequences, and the crossover is not performed in the middle of each sequence, but at the end of the longest prefix common to the considered sequences.
Final crossover: An ultimate crossover operator was designed in order to improve the previous ones. This operator is based on the same principle than the enhanced crossover operator, but the second sequence is not randomly chosen. Indeed, the second sequence is chosen as being the one having the longest common prefix with the first one.
Sequences extension: This operator aims at adding new frequent items at the end of several random frequent sequences.

Let us now consider the $node$ algorithm (C.f. Algorithm 2). Two mains operations are performed. First when a new neighbor v_t tries to connect on a node u_t (recv(v,$connect$)), a message is sent from u to v in order to give the address of the $Distributed_{SP}$. The node v_t is thus included into the process. Second, when a message from $Distributed_{SP}$ is received then the following operation is performed. A score, representing the distance between a candidate and the local operations performed on the node is computed, D_t^u. If a candidate is included in D_t^u then the score is set by $100+size(candidate)$. As our approach is heuristic-based, the main idea is thus to give a rewards to candidate fully included into a sequence. Furthermore, setting a number greater than 100, assure that our sequence is fully included in $Distributed_{SP}$. Otherwise as we are interested in long sequences because they are much more informative, long sequences are rewarded. This is done by considering the Longest-Common-Subsequence (LCS) [2] algorithm. Then the most the sequence is included, the most its score increases.

4 Experiments

To evaluate our approach, we implemented a simulator capable of running simulated unstructured P2P system. Conducted experiments were done by using real datasets. We use a classical benchmark (Pumsb) [8] and an access log file (AccessLog). The access log files contains all the request performed by the users on the web site of Inria. Since the activities of users on a P2P network are expected to be quite similar to requests on a web site, we believe that this file gives a good approximation for our purpose. As the file "pumsb" was first defined for the association rules problem, it has been transformed as follows: a different date is added

Algorithm 2: Node algorithm

Data: CS the candidate sequences to evaluate and D_u^t the partition of node u at time t.
Result: $LSCORE$ the set of local scores assigned to each sequence.

if *(recv(v,connect)* **then**
  send(v,@$Distributed_{SP}$);
if *(recv(@$Distributed_{SP}$,CS)* **then**
  for $S \in CS$ **do**
    $LSCORE$[S] $\leftarrow \emptyset$
    if *($S \subseteq D_u^t$)* **then**
      $LSCORE[S] \leftarrow$ 100+size(S);
    else
      //Give S a mark, and a
      //better mark to long sequences
      $LSCORE[S] \leftarrow \frac{size(LCS(S,D_u^t))*100}{size(S)} - size(S)$;
  send(@$Distributed_{SP}$, $LSCORE$);

for each items in the files entries.
Experiments were firstly conducted in order to analyze the convergence of the results as well as the communications costs. Furthermore experiments were conducted in order to analyze how our approach behaves when the nodes depart or arrive, e.g. at the beginning of the process only *classical music* is downloaded and after a long time only *pop songs* are downloaded.

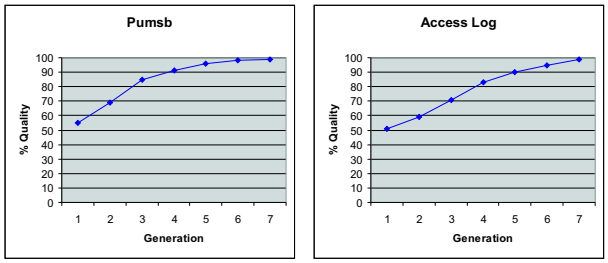

Figure 2. Results quality for proposed populations

4.1 Convergence of the result and communication costs

In this section, we consider the convergence of the result. As previously said (C.f. Section 2), we require that if nodes u_t ... v_t remain static long enough then the approximated solution $\tilde{F_{D_t}}$ will converge to F_{D_t}. Another important aspect that we have to consider is the communication cost. Broadcasting operations are in fact closely related to the number of candidate generation performed in order to obtain the set of real frequent sequences in the whole database D_t. The lower the number of generation is, the lower the number of broadcasting operations are. In the same way, the candidate generation is also closely related to the convergence of the result.

Algorithm 3: Quality measurement algorithm

Data: $candidatePopulation$, a candidate population to value. $realResults$ the real results to obtain (for comparison).
Result: $quality$, the quality percentage for proposed results compared to real results.

$sum \leftarrow 0$;
for $s1 \in candidatePopulation$ **do**
  $localQuality \leftarrow 0$;
  for $s2 \in realResults$ **do**
    $localQuality \leftarrow$ max($localQuality$, (LCS($s1$,$s2$)/size($s2$))*100);
  $sum += localQuality$;
$quality \leftarrow sum$/size($candidatePopulation$);
return($quality$);

A traditional level-wise algorithm was first applied on the whole database. Each candidate population proposed by our approach is thus compared to real result in order to obtain its local quality. This quality (Algorithm 3) is obtained as follows: we measure for each candidate population, the Longest-Common Subsequence (LCS) between the candidate sequences and the real result. Then the database was partitioned between different nodes and our approach was applied. Result of experiments are reported on Figure 2. For instance, we can notice that, for Pumsb, at first generation, the quality of the candidate population is greater than 50%. At second, we obtain 70%. We can notice that, for both datasets, from generation 6 the result quality is near 95% and we have to wait until the seventh generation to obtain 100%. These result show the efficiency of our approach since long frequent sequences are obtained when considering no more than 7 broadcasting operations.

4.2 Behaviors of nodes

One important characteristic of our approach is its convergence rate. Tests were thus performed in order to validate ability of our approach at adapting to the node behaviors. Experiments were conducted on the two real datasets. These files were chosen since they are from very different thematic. The main idea was the following: we would like to analyze what is the behavior of our approach when x% sequences from the original database Pumsb are replaced by x% sequences from AccessLog the destination database. Of course this process is repeated until Pumsb is entirely replaced by AccessLog. Our experiment then consist in estimating the quality of the results compared to those obtained by the traditional level-wise algorithm, as soon as

the replacement is achieved. In order to simulate a realistic unstructured P2P behavior, we considered a range value of 1-3% of sequences progressively, i.e. at each generation, replaced. The results for this test is reported in Figure 3. For instance, when Pumsb is replaced at the rythm of 1% per generation, results quality at the end of the replacement process is 100%. Of course the faster the initial database is replaced by the final database, the worst the results get. In other words, when the replacement is less progressive the quality decreases (e.g. we have tested 10% of modifications and we have noticed that the quality decreased to 55%).

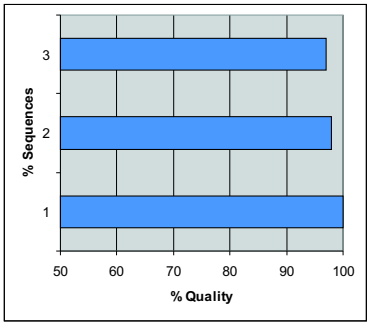

Figure 3. Results quality after a complete replacement of Pumsb by AccessLog

5 Conclusion

In this paper, we addressed the problem of improving resource location in unstructured P2P systems. This approach is based on data mining techniques and more precisely on sequential patterns mining. Even if efficient approaches for mining are proposed we have shown that they are inadequate in such a very large distributed database than a unstructured P2P system since: (i) they consider that we are provided with the whole database; (ii) they do not consider the dynamic nature of such a system, i.e. nodes act independently of one another; (iii) they are usually based on a join operation for candidate generation.
We proposed a new approach inspired by genetic algorithms in order to efficiently retrieve frequent sequences. Experiments conducted have shown that our approach is efficient.

Our approach can also be used in order to define clusters of P2P, i.e. nodes having quite similar behaviors. We can consider that $Distributed_{SP}$ stands for a special node for mining in a P2P system and that nodes included in the cluster will be used in order to find sequential patterns. Our future work consist in generalizing such an approach to a set of clusters in a way similar to the Superpeer concept [11]. The main idea is the following. We can consider that we are thus provided with a set of $Distributed_{SP}$ nodes. Each node can receive a set of sequential patterns from an other $Distributed_{SP}$ node and thus can consider these patterns against its own clustered nodes.

References

[1] X. Cheng, X. Yan, and J. Han. Incspan: Incremental mining of sequential patterns in large database. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD 04)*, Seattle, WA, 2004.

[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[3] Gnutella. http://www.gnutella.com.

[4] F. Masseglia, P. Poncelet, and M. Teisseire. Incremental mining of sequential patterns in large databases. *Data and Knowledge Engineering*, 46(1):97–121, 2003.

[5] F. Masseglia, M. Teisseire, and P. Poncelet. HDM: A client/server/engine architecture for real time web usage mining. *Knowledge and Information Systems*, 5(4):439–465, October 2003.

[6] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, Berkeley, California, 2003.

[7] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. Prefixspan: mining sequential patterns efficiently by prefix projected pattern growth. In *Proceedings of the International Conference on Data Engineering (ICDE 01)*, Heidelberg, 2001.

[8] FIMI Repository. Workshop on frequent itemset mining implementations (FIMI 04). http://fimi.cs.helsinski.fi/fimi04.

[9] R. Agrawal R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE 95)*, Tapei, Taiwan, 1995.

[10] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 03)*, pages 363–370, Melbourne, Florida, 2003.

[11] B. Yang and H. Garcia-Molina. Designing a superpeer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 03)*, 2003.