A Controlled Knowledge Base Evolution Approach for Query Hits Merging in P2P Systems

Rim Mghirbi[1,2], Hanen Majdoub[1], Khedija Arour[1] and Bruno Defude[2]

[1]*Faculty of Sciences of Tunis, Computer Science Department, Tunis, Tunisia*

[2]*Institute of Telecom and Management SudParis,
Computer Science department, Every, France*

rim.mghirbi@laposte.net, mejdoubhanen@gmail.com, khedija.arour@issatm.rnu.tn, Bruno.Defude@telecom-sudparis.eu

Abstract

Merging query-hits in large scale systems, like P2P, is challenging and potentially complex because results have to be ranked with respect to each other while sources are heterogeneous and with no centralized control. To solve this problem, we advocated in [10] a knowledge-based approach relying on users' profiles. A user profile includes information about past interests derived from the user past actions as well as information about peers from which results were obtained in the past for similar queries. Using a knowledge base can lead to the system obsolescence unless an effective approach is proposed to evolve this learned knowledge. Most used approaches for knowledge update are periodic and cannot react on user needs changes at the appropriate time. For this reason, we propose, in this paper, a controlled and distributed mechanism for knowledge evolution based on need observers. A need detector aims to detect the user new needs expressed in his queries, as well as the new resources. Experiments show a clear improvement of the system performance with our controlled mechanism.

Keywords: *IR, P2P systems, results merging, user profiles, KB evolution, update utility, Need detectors*

1. Introduction

In large scale systems such as the case of Peer-To-Peer systems (P2P), Information Retrieval (IR) is considered as an old problem that needs new issues to join the effectiveness of search to its efficiency. This latter factor concerns mainly the communication's cost that should be minimized in order to allow scaling and avoid the overhead in such systems. Two main problems have to be solved in P2PIR systems, the selection of the most relevant peers to answer a given query and the global ranking of all results returned by selected peers.

Global ranking is particularly challenging due to possible heterogeneity of peers in terms of collection size or IR model. For this reason, we have proposed in [10] a solution for global ranking based on the construction and use of a local knowledge-base (KB) on each peer. The solution aims to replace the use of peers' statistics by taking advantage of the user's past experience. It is worth mentioning that our knowledge base has the specific feature to be built automatically on each peer, and in a fully decentralized manner from the user behavior (non central coordination is needed). This latter is captured implicitly from the peers' log files and is processed to build knowledge. Our experiments show the effectiveness of this approach if the system is stables (no addition or deletion of resources, no changes in users' behavior). Of course in general, P2P systems are not stable at all and there is an inherent need to update the

KB. Most existing approaches for knowledge updating are periodic and so cannot guaranty that the KB evolves in a synchronous way relatively to the system evolution.

Based on these problems, our main goal in this paper is to define a controlled mechanism for the knowledge base evolution. This mechanism is based on need observers, located on each peer, observing user's queries and system answers and able to decide when a local KB has to evolve.

The main contributions of this paper are (i) a decentralized architecture for knowledge evolution, (ii) the definition of need observers and associated decision algorithms and (iii) a set of experiments validating our approach.

The remainder of this paper is organized as follows. Section 2 aims to position the problem in the context of information retrieval on large scale systems and focuses on semantic approaches using some king of knowledge base. In section 3, we review the main existing approaches for knowledge updating. We detail in Section 4 our proposal. We present the system architecture and several algorithms for this goal. An experimental validation is provided in section 5.

2. Problem Statement

Distributed Information Retrieval systems and more specifically P2P IR systems had specific problems. When the user issues his query on such a system, a selection phase is then to choose relevant peers that can be solicited to answer the query. Thus, each selected peer contributes to solve the query by sending a ranked results list to the initiator peer.

All these lists have to be merged into one final ranked list. This merging task is challenging due to the big heterogeneity of peers regarding their collection (type and size), their IR model and to the lack of reliable and global statistics.

Ideally, contributed systems should send content about their collections to allow ranking documents in a uniform manner. However, this strategy leads to a system overhead due to the communication's cost it incurs and does not respect autonomy of peers. In order to avoid this problem, we have proposed in [10], a profile-based merging algorithm (PBA) that aims to replace the use of global system's statistics by local information learned from user's behavior when he interacts with past queries' hits (see Figure 1).

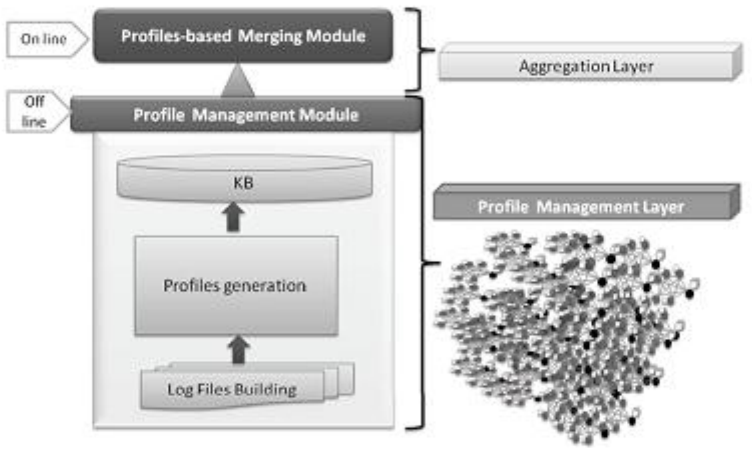


Figure 1. Knowledge Based Merging Approach

A user profile is deduced from the association between queries terms, contributed peers and documents. Formally, a profile Pr can be represented as a tuple: Pr= {Qt,Cp, Cd} where Qt={q1, q2,…, qm}, is a set of queries terms shared between similar Contributed Peers Cp and similar Contributed documents Cd. The set of all profiles stored in a peer is called a knowledge base. Thus, our merging algorithm can recommend the ranking of returned documents in the positions that best fit the user's need. However, the knowledge base may provide obsolete recommendation if the system or the user's need evolve.

Here, we give some realistic scenarios for the **observed** inconsistency between the system evolution and the user need on one side and the KB state on the other side. We recall that a peer's knowledge base is created when a user asks questions and chooses answers by clicking, accessing or downloading documents. A user often poses needs that are semantically close during a given period, either to refine his query or to consult close needs of interest.

Exampley posing queries with the theme RID, P2P, the IR process, the distribution problems... the KB, therefore, includes themes that are close and complementary. The documents set in the database are, from the user perspective, the most interesting for his queries.

If a user asks questions again about a theme already seen, the aggregation program may recommend him, starting from the results returned, those that interested him in the recent past: the KB will help to guide towards documents that are viewed or downloaded for this theme, this allows classifying these results in the first positions.

Once the user enters a query"car purchase" for example, we notice that the searched themes cut with what has been learned comparing to the learned terms. Hence, the query is considered as a new need that requires a new learning effort. If the peer continues to run queries of the same theme, the utility of regenerating the KB while including these new needs becomes insistent. Indeed, without prior knowledge, the classification will be randomly done and there is no way to exploit some knowledge.

A second situation that requires a use of learning is that when a user posing a learned query receives by one of the contributor peers an answer that doesn't appear among the previous similar queries answers. This situation can be interpreted by the accommodation of new documents that may interest the user while they are not learned. If the needs to this document increase when submitting other queries, this document must be included in the KB.

From the cited example, we notice that an important challenge in this context is to keep the KB data up-to-date with respect to the system evolution and the user need. The KB evolution goal is to enhance the retrieval effectiveness and this without paying high maintenance costs. Hence, any system using a KB has an incentive to update it as frequently as possible to increase the degree of freshness and efficiency of served results.

Unfortunately, given the large size of the knowledge base, it may take a long time to be updated at any time. However, as the user's needs evolve, there is a need to validate knowledge base and to enhance results quality with respect to this need. One trivial way of achieving freshness is having indicators that give notice whenever the knowledge base is updated.

For these reasons we aim to set a mechanism for knowledge base evolution based on the detection of users' needs changes. The proposed solution prioritizes the knowledge base 'entries to refresh according to a heuristic (updating Utility) that combines the rate new needs' changes and the rate of new queries and without need of any cooperation. To do this, it seems important to ask three main questions:

- Which changes are concerned with the update?

- Who is responsible for the knowledge base update decision? Is there any centralized authority that decides to globally update the network, or may this decision be local?
- Is the monitoring of network changes done in a real-time or changes are rather perennial?

Before answering these questions that will guide our proposal, we review some approaches for knowledge update.

3. Related Works

The main title (on the first page) should begin 1 3/16 inches (7 picas) from the top edge of the page, centered, and in Times New Roman 14-point, boldface type. Capitalize the first letter of nouns, pronouns, verbs, adjectives, and adverbs; do not capitalize articles, coordinate conjunctions, or prepositions (unless the title begins with such a word). Please initially capitalize only the first word in other titles, including section titles and first, second, and third-order headings (for example, "Titles and headings" — as in these guidelines). Leave two blank lines after the title.

A literature review shows that the knowledge update problem was often treated based on push approaches [8, 5, 15, 2]. Push policy is specific to real-time systems where usually a centralized server updates the knowledge bases in a uniform manner and periodically [11, 3]. The problem with push approaches and periodic updates is that of considering the user as a passive entity whereas these updates are especially user-centered. Besides, the problem with a periodic update is twofold:

- Rebuilding the knowledge base while the system has not evolved;
- Spending some (possible large) periods without update while data are stale.

Moreover, adopting centralized server update imposes a server overhead and seems to be not suitable for our prior choices of scaling [10]. Intuitively, it is obvious that this cannot be efficient and effective in large scale systems due to the amount of data that would have to be transferred.

More recently pull approaches [4, 5] have been proposed and run the update on users' demands. Adopting a pull approach requires the user to directly ask for update, which Moreover, adopting centralized server update imposes a server overhead and seems to be not suitable for our prior choices of scaling [10]. Intuitively, it is obvious that this cannot be efficient and effective in large scale systems due to the amount of data that would have to be transferred.

More recently pull approaches [4, 5] have been proposed and run the update on users' demands. Adopting a pull approach requires the user to directly ask for update, which imposes on the one hand a cognitive overhead in addition to his service usage, and the recourse to some heuristics for estimate the next update on the other hand. The estimation task is generally judged as not evident.

In this work, we first argue that the real problem in semantic aggregation for large scale information retrieval is the ability to cope with changes to the KB in the presence of dynamic user behavior and data updates. For these reasons, many applications try to study the user behavior to understand from his specific intensions when to refresh. Based on this assumption, we propose new algorithms for controlled evolution mechanism that guides the next update at the suitable moment and this, in a completely decentralized manner.

4. A Controlled Knowledge Evolution Mechanism

4.1. Overview

To present the main features of our proposed solution, we begin by answering the questions asked in Section 2. As concerns the changes to detect, it is important to know that in our context of P2PIR systems, three kind of changes can be observed: changes in users 'queries and in system resources (peers or documents). For any kind of changes, the KB has to be updated. Peers join and leave the system frequently; their relative documents are naturally added or removed from the system. Besides, new documents can be added or removed from existing peers while the knowledge base is not refreshed to capture these changes. These peers and documents can be useful to the user and can suit his needs while they are not yet learned in his knowledge base.

Updating a knowledge base may rely on a centralized authority [11] which is able to decide what and when updating. However, using a centralized authority to update a system of many hundred of peers will incur a system overhead and does not respect peer's autonomy.

For this reason, we have chosen a decentralized solution. Therefore a local decision is runon each peer to update its knowledge base. However, it is worth remarking, that being local, the peer's decision to evolve its KB is not antagonistic with others peers' decisions. Indeed, a peer that evolves its KB evolves indirectly changes in the system. Thus, the main contribution of this paper is the incremental aspect of the system refresh based on decentralized decisions to run local updates.

The third answer concerns the update delay. This latter should take into account the nature of changes. It would be ineffective to make updates continuously even for small changes. To ensure system stability, we do not use a real-time approach. One can notice that update will be done in an off-line manner.

4.2. Proposed Architecture

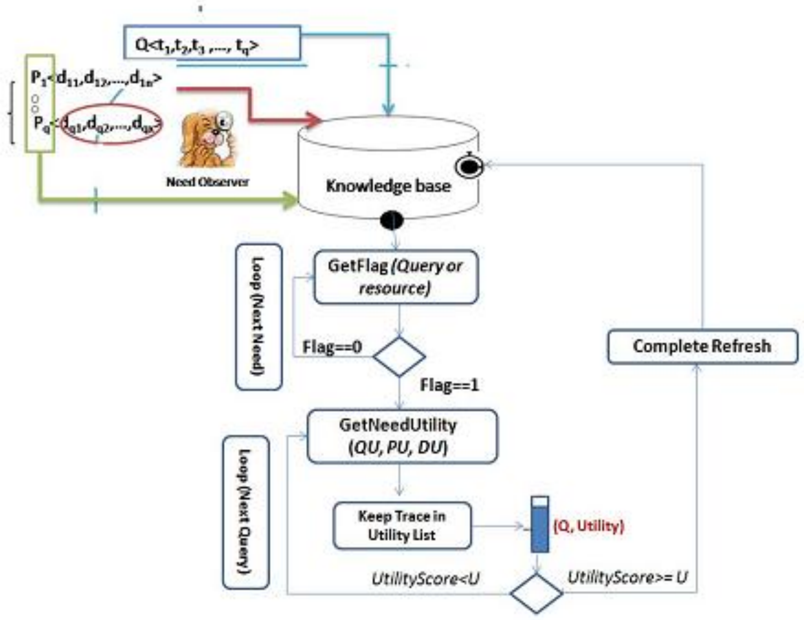


Figure2. Knowledge base Update Module

Figure 2 presents the architecture of the controlled evolution mechanism run on a given peer. This architecture is based on an activity diagram that describes the main steps performed for refreshing the knowledge base. It is obvious from the Figure that the process is run since the user had received the results of his submitted query from several contributor peers. A set of need Detectors (depending on the kind of need), running on the application background, as a daemon, try to evaluate three kinds of needs (topics, documents and peers) against information stored in the knowledge base (or profiles base). Need detectors use Flags to claim that the need is ”new” compared to the KB state (so it’s set to 1) or already learned (flag value =0) as presented in Algorithm 1 and 3 for respectively topics and resources.

When a Need Flag is set to 1, the need utility will be computed as presented later in Algorithm 4. In this step, automatically the query (with its hits) and utility scores are added to an utility list.

When the utility score reaches a fixed threshold U, a warning is turned on and the KB can be regenerated with respect to the new needs.

4.3. Updating Process

This section presents the main steps to perform a KB evolution. Several algorithms will be presented to detect new queries and resources in the P2P system (Documents and/or nodes), to compute the utility and to update the KB. In all these algorithms we will consider the following notation table:

Table1. Notation Table

Symbols	Notations
q	User query
N	Initiator peer
Q	The queries set submitted from an initiator peer
KB	knowledge base
T	the set of queries’ terms existing in the profiles of KB
$\Im$	a similarity threshold between queries and profiles termes that checks the system coherence with the KB
PR	The set of profiles in KB similar to q
R	resource (peer or document)
Need	User need (query, peer ,document)
$Flag$	a new Need detector
QU	the new queries’ utility
PU	the new contributor peers’ utility
DU	the new documents’ utility
$\mathcal{U}$	Utility threshold value to trigger the update
$MaxUtility$	max(QU,PU,DU)
max	the maximum similarity score between q and all the profiles in PR
score	coherence score
Nbq, Nbd	respc the queries number in Q and the documents number

4.3.1: New queries detection: Algorithm 1 gives the main steps to detect that a query q is a new need compared to the KB’s state. Therefore, a coherence score between the system and the KB’s state is computed (see Algorithm 2). If the score is below a similarity threshold £, a detector Flag for q is set to 1, the query is then considered as a new need.

```
Algorithm 1: GetQueryFlag
input  : KB:knouwledge Base
         ℑ:Coherence threshold system with KB
         Q:User query
         N: Initiator peer
output: Flag: Indicator value specifying the new need
1 begin
2     $Flag = 0$
3     $Score(Q) = GetNeedProfileSimilarity(Q, KB)$
4     if $Score(Q) < \Im$ then
5         Flag=1
6     return Flag
7 end
```

```
Algorithm 2: GetNeedProfileSimilarity
input  : KB:Knowledge Base
         q: User query
         T: All terms in a profile
output: Score: coherence score
1 begin
2     max=0
3     PR=GetQuerySimilarProfiles
4     foreach $pr$ $in$ $PR$ do
5         Score = $\frac{|q.termes \bigcap pr.T|}{|Q.termes \bigcup pr.T|}$
6         if Score >= $max$ then
7             max = Score
8     return Score
9 end
```

4.3.2. New Resources Detection: The main goal of this detection is to check whether new resources (R) (peers or documents) join the system. This detection will be based on progressive comparison between the new queries' answers and the answers in the KB for similar queries. The notion of similarity between new queries' terms and terms in the profiles PR from KB is based on Slaton's semantic similarity, given in Algorithm 2. Algorithm 3 sets a detector Flag to 1for each resource that not exist in the Profiles of the query q and so decides that R is a new need compared to the KB's state.

```
Algorithm 3: GetResourceFlag
input  : PR: Profiles base
         q:user query
         R: Resource (peer or document)
output: Flag: Indicator arrival of new resource;
1 begin
2     Flag= 0
3     if !Exist $(R,PR)$ then
4         Flag=1
5     return Flag
6 end
```

4.3.3. Update Utility Computing: The controlled evolution of our knowledge Base is based, as already cited, on default needs' detection (or new needs) represented by Flags. However, to run the update, it's not sufficient that a Need is considered as new. In fact, the decision of running the updating step is related to an utility value for each kind of need. This utility depends on which detector is used, or on all of them (the query, or resource detector). In fact, we consider one utility by each kind of need (QU, DU and PU) and needs whose utility value is higher than 1, are added to the Needslist (see Algorithm 4). A complete trace of the needs (resources and submitted queries) and their respective utilities is injected in the NeedsList.

```
Algorithm 4: GetNeedUtility
entrees: KB: knowledge Base
         q: requte utilisateur
         Q: set of queries from an initiator peer
sortie  : NeedUtility: list containing the queries considered new needs
begin
    Nbq=0
    Nbd=0
    QU=0
    PU=0
    DU=0
    foreach $q$ $in$ $Q$ do
        Nbq++
        $QU = QU$ + GetQueryFlag foreach $p$ $in$ $q.P$ do
            foreach $d$ $in$ $P.D$ do
                Nbd++
                if GetPeerFlag =$1$ then
                    $PU = PU$ + GetDocFlag
                $DU = DU$ + GetDocFlag
                if GetQueryFlag$!=0$ || GetPeerFlag$!=0$ || GetDocFlag$!=0$ then
                    NeedUtility.add(Q,QU,PU,DU,Nbq,Nbd)
    Return NeeddUtility
end
```

4.3.4. The Knowledge Base refresh: The utility score computing for each need, as presented in Algorithm 4, is the necessary condition for performing the update. Indeed, when utility score is above a given utility threshold (see Algorithm 5), the update can be realized. We consider here one utility score (of the query or the resources) depending to the kind of the used flag. When all Flags are used, the maximum utility of the three is compared to the utility threshold.

To refresh the KB, two approaches can be used: the complete or the incremental refresh approach. In our case, we choose to achieve a complete update by the regeneration of information gathered from the NeedsList and the knowledge base as presented in Algorithm 5. It is important to notice here that even a complete refresh is done in a purely local manner with respect to the initiator peer.

```
Algorithm 5: Complete Refresh
  input  : KB:knowledge Base
           $\mathcal{U}$:Threshold value to trigger the update
           Need:User need (query and/or peer and/or document)
  output: KB: New base
1  begin
2      ListOQ = getLearnedQueries (KB)
3      NeedsList = GetNeedUtility ()
4      T = NeedsList.size()
5      MaxUtility = NeedsList.get(T,UQ)
6      Nbq = NeedsList. get(T,Nbq)
7      if $\frac{\text{MaxUtility}}{\text{Nbq}} > \mathcal{U}$ then
8          ListNQ = GetQueries (NeedsList)
9      GenerateBase (ListOQ,ListNQ,KB)
.0 end
```

Algorithm 5 presents the different steps for the complete KB refresh and determines the conditions to trigger it.

5. Experiments

5.1. Simulation Environment and Parameters

To validate our approach, we have developed a peer-to-peer simulator based on XML files describing the system peers and the documents they contain, as well as queries which will be launched on the system. Our simulator uses a module for data distribution and replication (uniform, random, etc.), the Benchmarking Framework for P2PIR [16], developed in our team. This framework is configurable, it allows user to define some parameters such as the system size, the documents and queries' distribution methods and replication rate, etc. In the present study, we used a centralized dataset which is a selected subset of the DMOZ [6] web directory. It contains over 28182 documents and 4246 topics.

This subset of documents is obtained from 810 sites which share more than 10 documents. During simulations, each site is considered as a peer. Queries follow a zipF law distribution [1] and are replicated three times. Documents in our experiment are distributed based on a classification method i.e. the documents distribution is done in a way that springs naturally from the collection via document URL [9]. URL domain is the data relied upon by the classification. No replication is considered within documents to consider our approach performances in the worst case.

For evaluation metrics, we use the same metrics presented in [10], i.e the precision at a given document cutoff (p@k) [13], the Mean Average Precision (MAP) calculated by averaging the precision p@rank(d) at the cutoff rank(d) for all relevant documents, the Relative precision (RP) proposed in [14] that traduces the probability of relevance of a document estimated as inverse rank in reference ranking and the SimilarPositions@k, developed in the context of our work [10] and aiming to compute the percentage of positions left in their place, compared to the centralized rank list.

In this set of experiments, we applied the Profile-based aggregation algorithm (PBA) [10] to conduct our global ranking model. We compare the results obtained by our approach to those obtained by a centralized IR system. It's worth mentioning that our simulator considers both global and local similarities to rank, instead of using only global similarities. Global similarities are those used by the centralized collection, while local similarities are relative to the contributor peers. It implies that the values we get for the different IR metrics are relative ones and not absolute ones (for example a precision of 1 indicates that our approach gives the same result than a centralized one).

In these experiments, we do not introduce heterogeneity in IR models. We only use cosine distance between queries and documents either in global or local similarities. The evaluation of the proposed approach for knowledge base evolution is based on the use of PBA with and without updated knowledge base. To evaluate our updating method for the PBA algorithm, we first focused on the learning step. We reserved 1/3 of the queries in each peer for training. Moreover, we consider the training only for 450 peers and not for all of the 810 peers. This allows the simulator to send as query hits new resources that are not learned to see our potential to observe queries defaults as well as resources defaults.

During the training phase, profiles are extracted from logs and constructed using formal concept analysis (more precisely using the Godin's algorithm [7]), implemented in the platform Galicia V 3 [12].

5.2. Test Scenarios

Different test scenarios can be achieved to detect a knowledge default: we cite in the following:

- **S1 Arrival of new queries:** We consider in this case the change of users' needs. More precisely, we need to detect the change of users' interests. We know that users' interests change over time. The need of updating the knowledge base is necessary. Hence, our goal is to dynamically track their changes as the user interacts with the system.
- **S2 Arrival of new documents:** In this case, we suppose that the system is enhanced by new documents
- **S3 Arrival of new peers:** In this case, we suppose that new peers have joined the system currently, we only simulate and evaluate the change of users' needs which is associated directly to him (so only the query Flag is used). PBA algorithm starts in any peer with the simulation parameters described in the following table:

Training and test		#Queries
B0	initial Knowledge base	2700
T0	initial test on all peers (450)	1100
B1	# New queries detection from T0	275
T0z	Zoom test on T0 (10 peers)	———
T1	new queries for test	4300
B2	updated Knowledge base	2700+275
ulterior tests	T0/B2 , T0z/B2, T1/B2	———

Table 2. Test scenarios

Two conditions for update are considered:

- The threshold $\mathfrak{J}$=50% is the condition to decide that a query is a new need.
- The utility threshold >= 30% of all submitted queries number in a given peer is the condition to recommend the update.

5.3. Experimental Results

We begin with the test T0 described in the table 2 where we launch 1100 queries from 450 peers. The first step of this experiment is to evaluate the capability of our approach to detect defaults. The algorithms 1 and 2 for queries defaults detection is run on T0 and T0z and provides the results depicted in Figures 3, 4 and 5. Figure 3 presents an overview of the average non-satisfaction rate of the queries aggregated at the system level, and shows that the number of default queries (which present a non satisfaction rate greater than the threshold) is ascending over the time.

In order to track the reason of this failure, we proceed by doing a zoom on some peers. More precisely we focus on the first 10 peers in the system (see Figure 4). We notice from the results depicted in Figure 4 that 6 peers among the ten present a knowledge default since their non satisfaction rate exceeds the fixed threshold.

We followed the same strategy to see more precisely the reason of the failure in these peers and we focus on the queries launched from these peers. At this level, Figure 5, shows that in each peer, the number of default queries is more than 30% of the peers queries. By this condition we simulate that our default stack is full and then we can start the update. At this step, we run the update on each peer in failure. This update is based on the complete refresh approach and we present as follows the system performance improvement when updating the base by injecting the queries that present knowledge defaults. To detect the system improvement when applying the PBA, we run a test phase T0z before and after the update i.e on B0 and on B2 which is the updated base with respect to B1 (the set of injected failed queries). The test aims to see the system improvement when we submit among others, learned queries. Results of this test are depicted in Figures 6, 8 and 9.

All metrics show a clear improvement when PBA is run with update. However these results are individuals (relative to single peers), that is why we tried to have an overall view on the system performance when considering the MAP of all peers as presented in Figure 10. The result showed in this Figure confirms that the individual decision for knowledge update driven from each peer is not antagonistic and they contribute to improve obviously the overall system performance.

The last question we want to answer throw this experiment is to see the impact of the knowledge update on the performance of new queries in the system. This is the goal of test T1. The result of this test for the MAP metric is depicted in Figure 11 which presents the initial system performance with the initial base B0 and how this performance was improved when the knowledge based is updated.

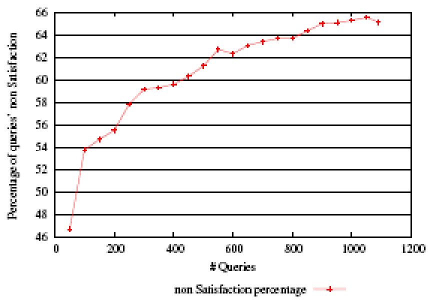
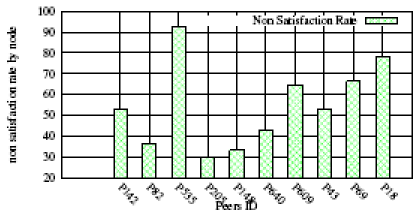


Figure 3. Average queries non satisfaction rate



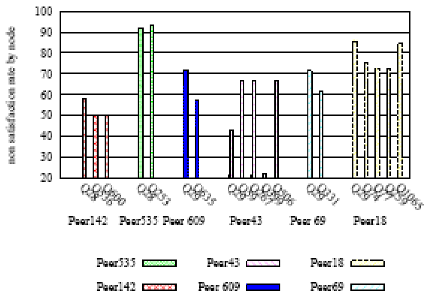


Figure 4. Queries non satisfaction rate by nodes

Figure 5. Zoom on individual queries

6. Conclusion

We proposed in this paper, a new approach for knowledge base updating based on detecting system evolution. Detection is made by local need detectors that look at three kinds of changes that are respectively changes on user topics, new added peers, and new documents. Architecture and several algorithms have been proposed for this purpose.

The main advantage of this approach is that it allows controlling knowledge evolution at a peer level since it is able to detect peers that have to evolve and when to update. Experimental results validate our approach and show that a fully decentralized decision to update independent bases can ensure a better behavior of the updated peers and of the entire system.

Even if our proposal has been validated on the global ranking of results, it can be also interesting for other process manipulating a knowledge base (selection of relevant peers to route queries for example). In the same manner, it can also be applied to other type of knowledge base (one just need to define new observers, depending on the structure and the content of the KB).

Moreover, we think that the proposed approach can be an interesting issue for another important problem in semantic P2P systems, that is the initial construction of the KBs. In fact, all semantic approaches need a learning step as preliminary. In our case, the evolution mechanism may allow to reduce this step or to allow a self-adaptation of a predefined KB used to initialize new peers.

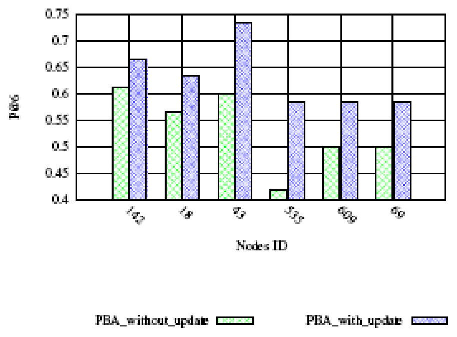


Figure 6. Impact of the update on Precision@k

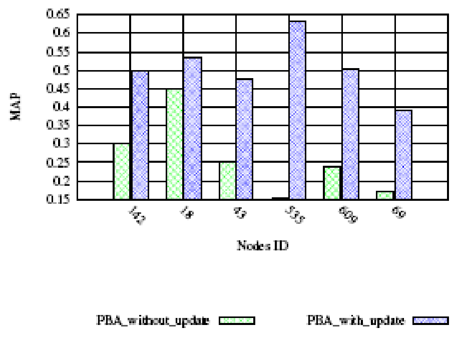


Figure 7. Impact of the update on MAP

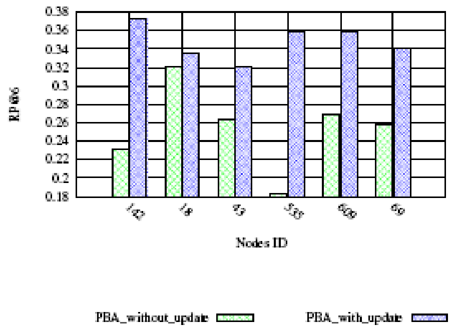


Figure 8. Impact of the update on RP@k

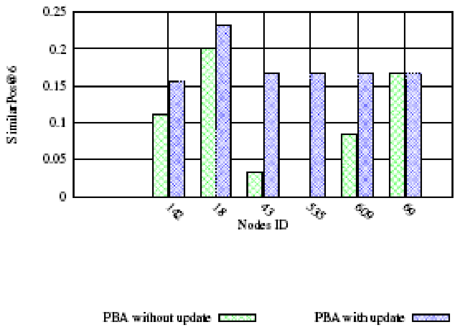


Figure 9. Impact of the update on Similar-Position rate

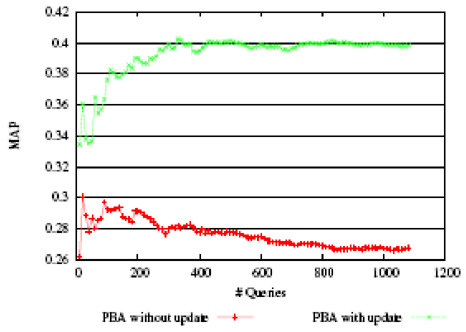


Figure 10. Global view of the system (MAP improvement)

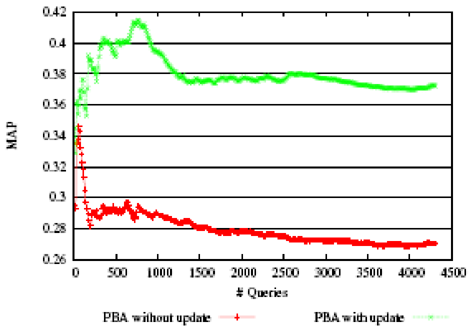


Figure 11. Global view of the system (MAP improvement) for new queries

References

[1] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet", Glottometrics, vol. 3, **(2002)**, pp. 143–150.
[2] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham and P. Shenoy, "Adaptive push-pull: Disseminating dynamic web data", IEEE Trans. Comput., vol. 51, no. 6, **(2002)** June, pp. 652–668.
[3] L. Bright and L. Raschid, "Using latency-recency profiles for data delivery on the web", In Proceedings of the 28th international conference on Very Large Data Bases, VLDB '02, VLDB Endowment, **(2002)**, pp. 550–561.

[4] V. Cate, “Alex – a global filesystem”, In Proceeding of the 1992 usenix file system workshop, **(1992)**, pp. 1–12, 1992.

[5] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham and P. Shenoy, “Adaptive push-pull: Disseminating dynamic web data”, In WWW10, **(2001)**, pp. 265–274.

[6] DMOZ, “DMOZ Open Directory Project”, http://olc.ijs.si/dmozReadme.html, **(2011)**.

[7] R. Godin, R. Missaoui and H. Alaoui, “Incremental concept formation algorithms based on Galois (concept) Lattices”, Computational Intelligence, vol. 11, no. 2, **(1995)** May, pp. 246–267.

[8] C. Liu and P. Cao, “Maintaining strong cache consistency in the world-wide web”, In Proceedings of the Seventeenth International Conference on Distributed Computing Systems, **(1998)**, pp. 445–457.

[9] J. Lu and J. P. Callan, “Content-based retrieval in hybrid peer-to-peer networks”, In CIKM, **(2003)**, pp. 199–206.

[10] R. Mghirbi, K. Arour, Y. Slimani and B. Defude, “A profile-based aggregation model in a peer-to-peer information retrieval system”, In Proceedings of the Third international conference on Data management in grid and peer-to-peer systems, Globe’10, Berlin, Heidelberg, **(2010)**, Springer-Verlag, pp. 148–159.

[11] D. Theodoratos and M. Bouzeghoub, “Data currency quality factors in data warehouse design”, In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW’99), Heidelberg, Germany., Heidelberg, Germany, **(1999)**.

[12] P. Valtchev, D. Grosser, C. Roume and M. R. Hacene, “Galicia: an open platform for lattices”, In Using Conceptual Structures: Contributions to the 11th Intl. Conference on Conceptual Structures (ICCS’03), Shaker Verlag, **(2003)**, pp. 241–254.

[13] E. M. Voorhees and D. K. Harman, “TREC: experiment and evaluation in information retrieval”, The MIT Press, **(2005)**.

[14] H. F. Witschel, “Global and Local Resources for Peer-to-Peer Text Retrieval”, PhD thesis, Der Fakultat fr Mathematik und Informatik der Universitat Leipzig eingereichte, **(2008)**.

[15] J. Yin, L. Alvisi, M. Dahlin and A. Iyengar, “Engineering server-driven consistency for large scale dynamic web services”, In Proceedings of the 10th international conference on World Wide Web, WWW ’01, New York, NY, USA, **(2001)**, pp. 45–57, ACM.

[16] S. Zammali and K. Arour, “P2PIRB: Benchmarking framework for p2pir”, In Proceedings of the Third international conference on Data management in grid and peer-to-peer systems, Globe’10, Berlin, Heidelberg, 2010, Springer-Verlag, pp. 100–111.