"PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities"

F. M. Cuenca-Acuna, C. Peery,
R. P. Martin, and T. D. Nguyen

Write-up by Sergey Lugin
Saarland University, Department of Computer Science

2003

Content

1. Motivation

Peer-to-peer (P2P) systems have gained a world-wide popularity at the present days. In general, the P2P network unites the users, who want to have an opportunity of data exchange. Each user runs P2P software with equivalent functionality sharing some files in order to get an access to files shared by other users. The P2P systems are attractive due to:

1. the ability to leave shared, but distributed, data at their origin, rather than incurring the cost, privacy and safety concerns of collecting and maintaining them in a centralized repository,
2. ease of incremental scalability,
3. the possibility of scaling to extremely large sizes.

P2P systems can be divided into two common groups according to their architecture:

1. *Centralized systems* (e.g. Napster).
   A central server maintains an index of all files published in the network. All peer queries are directed to the server and it informs peers where they can download the data they are looking for.
2. *Decentralized systems* (e.g. Gnutella, Freenet).
   Peers use a point-to-point message exchange (message flooding) to locate the data they are looking for.

While decentralized systems have been tremendously successful and popular for music and video sharing communities, their search capabilities have been frustratingly limited. On the opposite side, the success of Internet search engines (e.g. Google, Yahoo) demonstrated that content based search and ranking is a *powerful model* for locating information across data collections exhibiting a wide range of sizes and content. One attempt to extend search capabilities for decentralized P2P systems using Internet search engine fundamentals was realized in a prototype system called *PlanetP* [1]. This paper describes fundamental principles of the architecture developed for PlanetP.

2. Introduction to PlanetP

PlanetP is a content addressable publish/subscribe service for unstructured P2P communities [1]. It is the first approach to provide content search and ranking for unstructured P2P networks. PlanetP is simple because each peer must only agree to perform a periodic, randomized, point-to-point message exchange with other peers and is powerful for two reasons:

- it can propagate information in bounded time in spite of the uncoordinated communal behaviour,
- it maintains a globally content-ranked data collection without depending on centralized resources or the on-line presence of specific peers.

The service has the following advantages:

1. The service provides **content search and ranking**.
   This system allows peers across the entire community to locate specific documents based on their content.
2. The service does not require central management.
3. The service provides resilience to rapid membership changes.
4. The service scales easily to several thousands peers.

3. Architecture

3.1 Local and global indexes

PlanetP is a typical peer-to-peer service. Peers share files that will be available eventually to all peers in the community. The main feature of the developed service is content search and ranking. In order to support it, PlanentP uses two data structures:

- *local index*, it describes *only* the precise content of documents published locally by a peer,
- *global index*, it describes all peers in a compact structure and is replicated everywhere in the community.

Let us consider these data structures in details:

Local index:

Peers publish documents (video, music) in PlanetP by providing *XML snippets* containing pointers to the appropriate files (if text documents have to be published then these documents should be used instead of XML snippets). A set of published snippets constitutes a peer *local index*.

PlanetP leaves the shared files in place but runs a *simple web server* to support retrieval of these files.

Global index:

PlanetP uses the *global index* to describe all peers and their shared information in the community. The index is replicated everywhere. To reduce the bandwidth usage, each peer summarizes the set of terms in its local index in a *Bloom filter* [2].

The Bloom filter is a bit-array that allows to quickly test a membership in a large term set using hash functions. The filter is computed by using *n* different hash functions to compute *n* indices for each term and setting the bit at each index to 1. Given a Bloom filter, we can ask, if some term *t* is a member of the set by computing the *n* indices for *t* and checking whether those bits are 1.

The global index is a table in which each row corresponds to a single peer in the community (see the table 2.1).

Nickname	Status	IP	Bloom Filter
Bob	ON-LINE	10.25.125.10	BF[…….]
Gera	OFF-LINE	10.67.12.101	BF[…….]
…	…	…	…
Fred	ON-LINE	45.23.78.115	BF[…….]
Juri	ON-LINE	125.37.167.15	BF[…….]
Tan	ON-LINE	10.25.125.11	BF[…….]

Table 2.1 The global index structure.

A peer has:

- *nickname*,
- *peer status* ("ON-LINE" or "OFF-LINE", the field allows to reduce bandwidth usage),
- *IP address*,
- *Bloom filter*.

The global index is replicated everywhere in the community by *gossiping*.

3.2 Gossiping algorithm

PlanetP uses gossiping to replicate the global index across peers of the P2P community. All members agree to continually gossip about changes to keep the shared data structure updated and loosely consistent. Gossiping was chosen because it provides:

- robustness to the dynamic joining and leaving of peers,
- independence from any particular subset of peers being on-line.

PlanetP's gossiping algorithm is a novel combination of an algorithm previously introduced by Demers [3] and partial anti-entropy algorithm. The Demers's algorithm consists of a rumoring and anti-entropy algorithm.

Rumoring algorithm:

Purpose: The algorithm provides spreading of new information across a P2P community.

If a peer has a change of its data structure (for example: publishing of new files), the algorithm operates as follows:

- every T_g seconds, a peer P_x pushes this change (called a *rumor*) to a peer P_y chosen randomly from the global index,
- if the rumor is new information for the peer P_y, then it starts to push this rumor just like P_x,

- this process is repeated for following peers,
- the peer P_x stops pushing the rumor after it has contacted n consecutive peers that already heard the rumor.

Anti-entropy algorithm:

Purpose: The algorithm allows to avoid the possibility of rumors dying out before reaching everyone.

All peers perform the algorithm independently:

- every T_r seconds, a peer P_x attempts to pull information from a peer P_y chosen randomly from the global index,
- the peer P_y returns the summary of its version of the global index,
- then P_x can ask P_y for any new information that it does not have.

Partial anti-entropy algorithm:

Purpose: The algorithm allows to reduce the time for spreading new information.

The algorithm extends each push operation:

- when a peer P_x pushes a rumor to a peer P_y, the peer P_y piggybacks the identifiers of a small number of the most recent rumors,
- then P_x can pull any recent rumor that did not reach it.

The process requires only one extra message exchange in the case that P_y knows something that P_x does not.

To complete the gossiping description, it is necessary to make some remarks about membership changes. There are three possible situations:

- Joining of new peers
  PlanetP uses gossiping to inform all peers about an appearance of new peer and its shared files.
- Leaving of present peers:
  This case involves two possible situations: temporal and permanent leaving. A peer discovers that another peer is OFF-LINE when an attempt to communicate with it fails. It corresponds to temporal peer leaving. The peer status is marked as “OFF-LINE” and information in the global index is not dropped. If a peer has been marked as “OFF-LINE” continuously for time

T_{Dead} , it is assumed that the peer has left the community permanently. In this case all information about the peer is dropped from the global index.

- Rejoining of peers
  This situation corresponds to temporal peer leaving when the peer status was marked as "OFF-LINE". To spread information about rejoining, the service uses gossiping (the peer status is marked as "ON-LINE" everywhere).

3.3 Content search and ranking

PlanetP implements a *content ranking algorithm* that uses the *vector space ranking model* [4]. In this model, each document and query is abstractly represented as a vector, where each dimension is associated with a distinct term. The value of each component of a vector is a weight representing the importance of that term to the corresponding document or query. The *relevance* of each document for some query is calculated as the cosine of the angle between the two vectors using the following equation:

$$Sim(Q,D) = \frac{\sum_{t \in Q} w_{Q,t} \times w_{D,t}}{\sqrt{|Q| \times |D|}} \qquad (2.1)$$

where: Q is a query, D is a document, $|Q|, |D|$ are the number of terms in Q and D, $w_{Q,t}$ is the weight of the term t for the query Q, $w_{D,t}$ is the weight of the term t for the document D.

A similarity of 0 means that the document does not have any term in the query while a 1 means that the document contains every term in the query.

There are some methods for term weight assignments. *TFxIDF* is the most popular method from them. The method combines:

1. *Term Frequency* - ($f_{D,t}$)
   This metric refers to "how often the term t appears in the document D".
2. *Inverse Document Frequency* - (IDF_t)
   It is "the inverse of how often the term t appears in the entire collection".

This technique allows to balance:

- the fact that terms frequently used in a document are likely important to describe its meaning
- terms that appear in many documents in a collection are not useful for differentiating between these documents

Using these metrics the weights are calculated as:

$$w_{D,t} = 1 + log(f_{D,t}) \qquad w_{Q,t} = IDF_t \qquad (2.2)$$

There are some problems to compute TFxIDF for P2P community. Documents are distributed across a P2P community and the network bandwidth is restricted to use. To overcome these problems, PlanetP approximates TFxIDF by breaking them into two sub-problems:

1. Ranking peers according to their likelihood of having relevant documents
2. Deciding on the number of peers to contact and ranking the identified documents

PlanetP's content search consists of two steps:

1. *Ranking peers.*
   The query that can contain one or more terms is evaluated for all peers in the community. For this purpose the Bloom filter is used. If a peer P_x has larger number of query terms than another peer P_y then the peer P_x has a larger rank value. After ranking the peer list is sorted.

2. *Querying the most relevant peers.*
   In this step, PlanetP queries the most relevant peers (from top to bottom of the ranking). Each peer returns a set of document URLs together with their ranks. PlanetP sorts received URLs and presents them for the user.

The figure 2.1 shows the PlanetP's content search schematically.

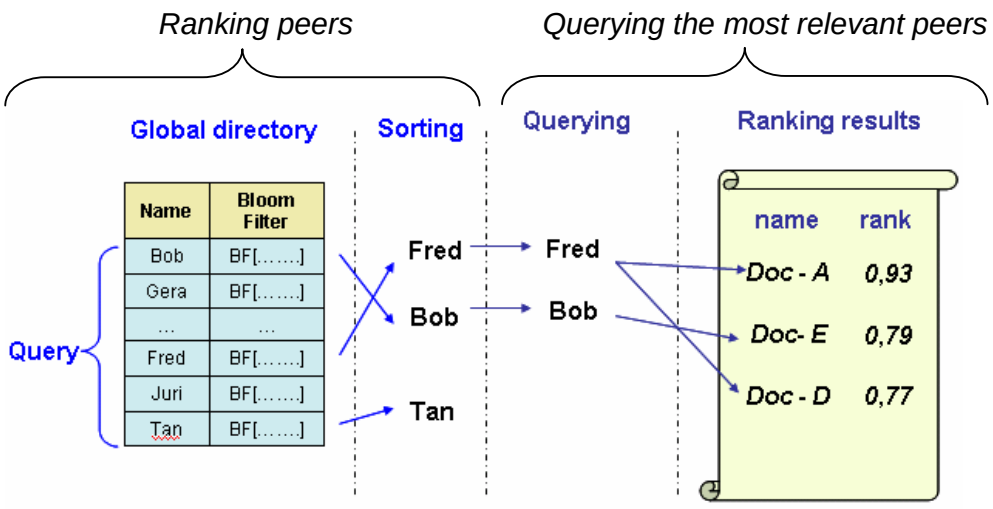


Figure 2.1 The PlanetP's content search.

Let us consider all steps in details:

Ranking peers:

PlanetP introduces a new measure similar *IDF* - *Inverse Peer Frequency* (*IPF*). The idea is that a term that is present in the index of every peer is not useful for differentiating between the peers for a particular query. For a term *t*, IPF_t is computed as:

$$IPF_t = log\left(1 + \frac{N}{N_t}\right) \tag{2.3}$$

where *N* is number of peers in the community, N_t is the number of peers that have one or more documents with term *t*.

Using this measure the peer rank is computed as:

$$R_i(Q) = \sum_{t \in Q \cap t \in BF_i} IPF_t \tag{2.4}$$

where
Q is the query, R_i is the relevance of peer *i* to *Q*, BF_i is the Bloom filter of peer *i*.

The equation assigns the rank for each peer according to the number and relevance of query terms that it contains.

Querying the most relevant peers:

The community can have a large number of peers that contain at least one term of the input query. In this case, it becomes infeasible to contact a large number of peers for each query. To solve this problem, the user has to specify a limit *K* on the number of potential documents that should be presented.

Planet executes following sub-steps:

- the service contacts peers from top to bottom of the ranking,
- each contacted peer returns a set of document URLs together with their relevance calculated by following equation:

$$Sim(Q, D) = \frac{\sum_{t \in Q} IPF_t \times (1 + log(f_{D,t}))}{\sqrt{|D|}} \tag{2.5}$$

  It is necessary to note that there are only two main discrepancies from the TFxIDF equation. IPF is used instead of IDF and the number of query terms is not used because it is a constant value.
- the service stops to contact peers when the documents identified by *p* consecutive peers fail to contribute to the top *K* ranked documents.

Simulation results indicate that *p* should be a function of the community size *N* and the number of document requested *K* as follows:

$$p = C_0 + \lfloor C_1 N \rfloor + \lfloor C_2 \sqrt{K} \rfloor \tag{2.6}$$

where: C_0, C_1, C_2 are constant values. The simulations results showed that the tuple (C_0, C_1, C_2)=(2,1/300,1/2.5) is a good initial values for equation 2.6.

This metric was called as “a stopping heuristic”.

4. Performance

The performance study involves evaluations of an efficacy for the content search and time/bandwidth usage for the gossiping algorithm. It is necessary also to notice that the performance study was based on a developed simulator. The simulator was validated against measurements taken from a prototype (up to several hundred peers).

4.1 Content search and ranking algorithm

To evaluate the efficacy of the content search and ranking, two accepted information retrieval metrics were used, *Recall (R)* and *Precision (P)*:

$$R(Q) = \frac{no.\_relevant\_docs.\_presented\_to\_the\_user}{total\_no.\_relevant\_docs.\_in\_collection} \quad (3.1)$$

$$P(Q) = \frac{no.\_relevant\_docs.\_presented\_to\_the\_user}{total\_no.\_docs.\_presented\_to\_the\_user} \quad (3.2)$$

R(Q) captures the fraction of relevant documents that the algorithm is able to identify and present to the user, *P(Q)* describes how much irrelevant material the user may have to look throughout to find the relevant material.

Text documents extracted from *Associated Press (AP89)* [5] were placed at two different document-to-peer distributions:

- *Uniform*
  It is the worst case for a distributed search.
- *Weibull*
  This distribution is caused by fact that 7% of the users in the Gnutella community share more files than all the rest together.

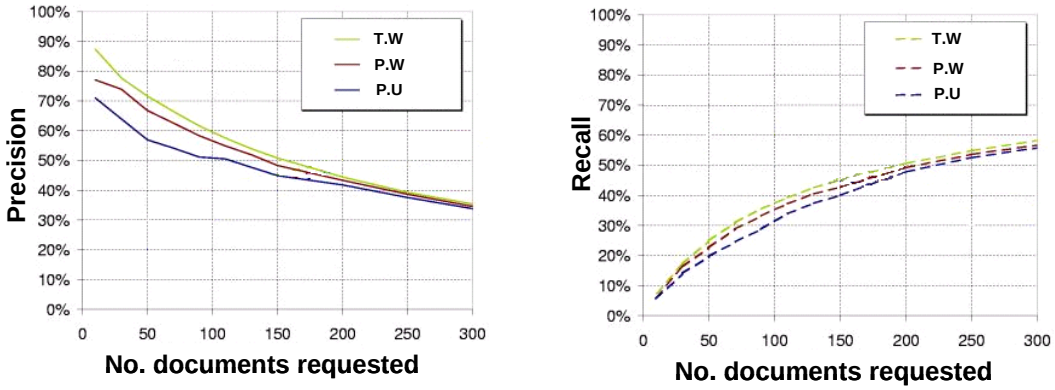
The simulated results for *PlanteP’s search engine* at two different document-to-peer distribution and the *centralized implementation TFxIDF are* shown on the figure 3.1.

Following abbreviations were used:

T.W is a search engine using TFxIDF (centralized implementation),

P.W is the PlanteP’s search engine (Weibull distribution of documents),

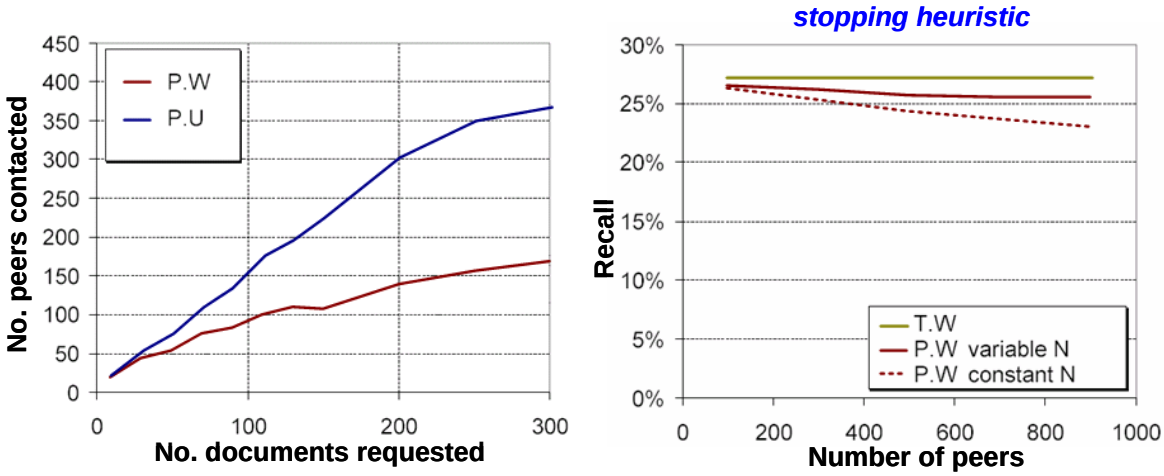
P.U is the PlanteP’s search engine (Uniform distribution of documents).



(a) (b)

Figure 3.1 Average recall and precision for the AP89 collection when distributed across 400 peers.

It is interesting to note that PlanetP tracks the performance of the centralized implementation TFxIDF closely.



(a) (b)

Figure 3.2 (a) Average number of peers contacted in community of 400 peers vs. number of documents requested, (b) Average recall as a function of community size.

The figure 3.2 (a) shows the average number of peers contacted in a community of 400 peers vs. the number of documents requested at two different document-to-peer distributions (Weibull and Uniform). The figure shows that the developed stopping heuristic allows PlanetP to search more widely among peers when documents are more widely distributed preserving recall and precision independent of document distribution. It is obvious, that for uniform document distribution the number of peers contacted is larger than for Weibull distribution.

The figure 3.2 (b) shows the average recall as a function of community size. We observe from it that PlanetP's recall would degrade with community size if the stopping heuristic were not a function of community size. The developed stopping heuristic allows to maintain recall close to a constant value independently of community size.

The presented results yield three important observations:

1. *PlanetP tracks the performance of the centralized implementation closely.*

   a) Performance is independent of how the shared documents are distributed.

   b) PlanetP's recall and precision is within 11% of TFxIDF's implementation.

2. *PlanetP scales well for communities of up to 1000 peers, maintaining relatively constant recall and precision.*

3. *PlanetP's stopping heuristic allows to maintain the close recall and precision independently of how documents are distributed.*

4.2 Gossiping algorithm

Firstly, to study the gossiping algorithm, we measure the time required to propagate a single Bloom filter throughout stable communities. Measuring propagation time is important because it represents the window of time where peer's directories are inconsistent, so that some peers may not be able to find new documents.

The figure 3.3 shows the propagation time for two scenarios. In both cases peers are connected by 45 Mbps links and the Bloom filter contains 1000 terms (3000 bytes). Following abbreviations were used:

LAN-AE: Peers use only push anti-entropy: each peer periodically push a summary of its data structure. The target requests all new information from this summary. The algorithm has been successfully used to synchronize communities.

LAN: Peers use PlanetP's gossiping algorithm. Some parameters used in the simulation are shown on the table 3.1

Parameter	Value
Base gossiping interval	30sec
Message header size	3 bytes
1000 terms BF	3000 bytes
BF summary	6 bytes
Peer summary	48 bytes

Table 3.1 Constants used in simulations of PlanetP's gossiping algorithm.

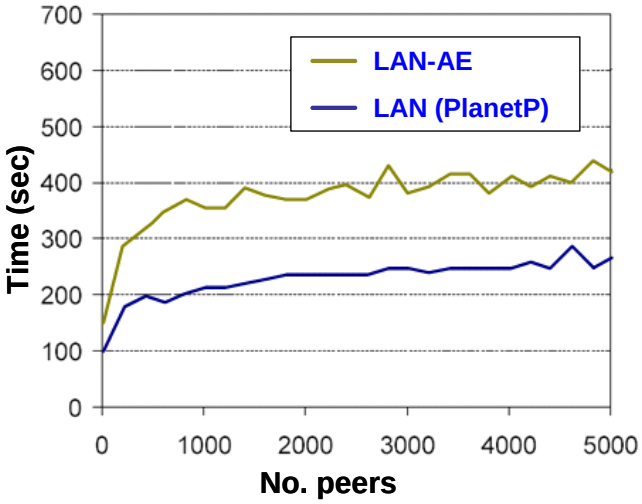
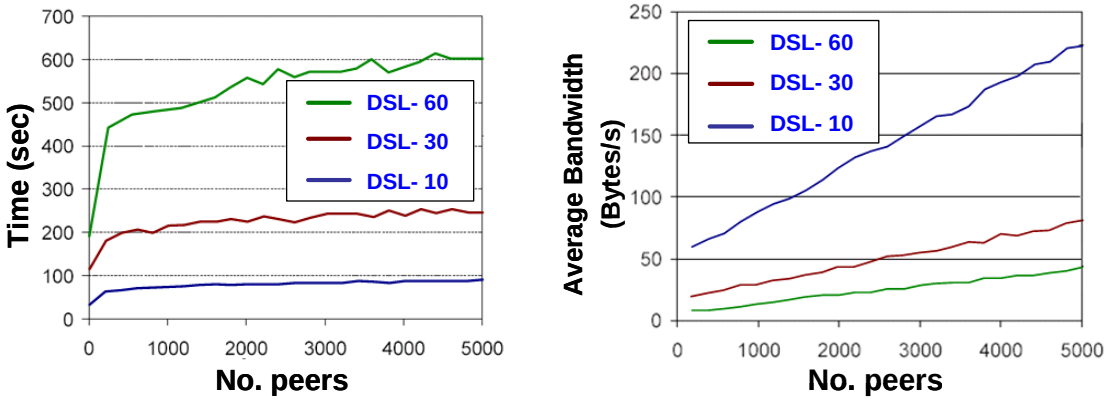


Figure 3.3 Time required to propagate a single Bloom filter containing 1000 terms everywhere vs. community size.

We observe from the figure 3.3 that the developed gossiping algorithm significantly outperforms the ones that use only push anti-entropy. Using rumoring enables PlanetP to reduce the amount of information exchanged between nodes while the mixture of pull (anti-entropy) and push (rumors) rounds reduces convergence time.

The figure 3.4 shows the propagation time and bandwidth usage for different gossiping intervals: 10 sec (DSL-10), 30 sec (DSL-30), 60 sec (DSL-60). In this case, peers were connected by 512 Mbps links.



(a) (b)

Figure 3.4 (a) Time and (b) average per-peer bandwidth required to propagate a single Bloom filter containing 1000 terms everywhere vs. community size.

It is interesting to notice that it is possible to trade off propagation time by increasing or decreasing the gossiping interval.

The presented results yield four important observations:

1. *The algorithm significantly outperforms ones that use only push anti-entropy for both propagation time and network volume.*

2. *Propagation time is a logarithmic function of community size.*

3. *Total number of bytes sent is very modest, implying that gossiping is very scalable.*

4. *We can easily trade off propagation time against gossiping bandwidth by increasing or decreasing the gossiping interval.*

5. Group extension

PlanetP has one significant limitation. The community can contain up to several thousands of peers maintaining recall and precision comparable with centralized implementation. One possible approach to overcome this limitation was also proposed by the authors of [1]. It is necessary to emphasize that it is only proposed approach and it does not have a real implementation.

Community:

The community with large number of peers have to be divided into a number of groups. Peers within the same group operate as described above. Peers from different groups will gossip an attenuated Bloom filter that is a summary of the global index for their groups. Using the single filter for entire group allows to compress amount of information that is necessary to exchange. Peers mostly gossip within their groups but, occasionally, will gossip to peers of other groups.

Search:

If a peer P_a (group A) tries to find documents that are relevant to a query Q and, for example, the Bloom filter of the group C contains relevant terms to a query Q, then the peer P_a would query a random peer P_{ci} from the group C. The peer P_{ci} would return a ranked list of peers in the group C. After that, the peer P_a can query peers of the ranked list and receive sets of document URLs with ranks as described before.

6. Related works

Numerous research efforts (*Tapestry, Pastry, Chord* and *CAN*) have produced highly scalable distributed hash tables (DHT) over P2P communities. In general DHTs spread (*key,value*) pairs across the community and provide retrieval mechanisms based on the key. This architecture makes it difficult to implement the content search. The problem is caused by the high cost of publishing thousands of keys per file.

Cori and *Closs* address the problems of database selection and ranking fusion on distributed collections. Both systems use servers to keep a reduced index of the content stored by other servers. The developed architecture yields two problems:

- The need for centralized resources.
- The possibility of a single point failure.
  If a centralized resource is crashed, then the community is not able to operate.

7. Summary

PlanetP is a content addressable publish/ subscribe service for unstructured P2P communities. It is the first work that supports content ranking for this kind of P2P communities. PlanetP tracks the performance of the centralized implementation TFxIDF by providing two features:

1. A gossiping algorithm that provides propagation of shared information everywhere and adds robustness to the dynamic peer behaviour.

2. Content search and ranking algorithm that provides the search capabilities comparable with centralized resources and operates independently of how documents are distributed throughout the community.

8. References

1. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. *PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities.* 12th IEEE International Symposium on High Performance Distributed Computing, 2003.

2. B. H. Bloom. *Space/Time Trade-offs in Hash Coding with Allowable Errors.* Communications of the ACM, 13(7):422–426, 1970.

3. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. *Epidemic Algorithms for Replicated Database Maintenance*. In Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, pages 1–12, 1987.

4. I. Witten, A. Moffat, and T. Bell. Managing Gigabytes: *Compressing and Indexing Documents and Images.* Morgan Kaufmann, San Francisco, second edition, 1999.

5. D. Harman. *Overview of the first TREC conference.* In Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1993.