Towards Collaborative Searching over an Overlay Network

Emmanuel Benazera, juban@free.fr

2 January 2006

There is a certain frustration in the searching the web these days. True, simple web interfaces let users search through a prescanned massive portion of the whole webshere and return results in microseconds. Carefully tailored search engines and their armies of zealot crawlers navigate the webshere, report, cache, classify and rank URLs and pages. The infrastructure that these systems necessitate has concentrated their development at a few powerful companies and network providers, the only players able to distribute powerful servers along the communication backbones in order to continuously serve an up-to-date reflection of the face of the websphere from their databases. However, if these infrastructures have changed and modified the use of the web in general, there is room for improvement in the way search results are presented, used and shared among users. Here we can pinpoint a few drawbacks and missing capabilities. First, adhoc multi-criteria ranking strategies have become so complicated that they leave the user with no intuition of why and how the list of results to his query has been concocted. Second, there is no possible direct feedback from the user to the search engine (for obvious reasons of treachery, etc...) so broken links and uninformative webpages regularly make it on the top of the results list of searches until crawlers remove them, or criteria have been fixed. Third, the search process through which the user manually narrows down on the collection of links of interests to him (through successive searches and elimination) is lost in the current range of web search services (but for the companies themselves, of course, that can match the users' IP addresses, query dates and keywords).

Notwithstanding the need for automated search engines and simple interfaces to their results, we argue that first, the concentration and power in the hands of the search giants through the devising of the ranking criteria and the massive amount of users information they can make use of is unhealthy for the future of the web, and does not beneficiate to the users (through the deshumanization of search results); second, there is for each user of a search engine a *missing social mass* that is the whole number of people that are conducting similar searches, within a certain time range, while not reporting on them (directly since they can't, or statically, e.g. through a dedicated webpage or blog).

A natural answer to these limitations[1] comes from the study of a now well established trend and technology for sharing on the Internet that is the spread of peer-to-peer (P2P) applications that let users search through

[1] There are other implicit limitations to the current search engines:

- So-called *crawlers* have to start from certain webpages, e.g. newly posted URLs. A certain part of the information can remain hidden for a time before it accesses the *conscious* (i.e. searchable) part of the websphere.
- Ranking must be extremely discriminatory (who does dig in beyond the first couple of result pages ?).
- There is no real-time search (it all comes from the archives).
- Updates are recurrent and may not be adapted to the different rhythms of changes of the different informational fields of the web.
- It is not easy to mix up the results of different search engines.

distributed databases for files and information fragments and exchange them through direct connections (i.e. not being routed through a dedicated server). We propose to build such an overlay network (i.e. a computer network built on top of the physical network) to relay and share ratings on the local results of users' searches through existing search engines as well as the web queries themselves. We argue that this would make part of the missing social mass appear thus opening new perspectives for the searching of the webshere. Users would be able to share and rate not only links, but whole search objectives and strategies. Thus the local websearcher would participate in real-time group searches digging in the massive and growing amount of information. In response to the three drawbacks given above, ratings could complement the current multi-criteria indexes, feedback would be made possible and shared among users, and each user's progression towards his closest interests would act as search suggestions to other users of related group searches.

The document is organized as follows: first we lay down a simple theory for both the collaborative filtering of URLs and the collaborative searching; second we study and adapt the best existing solutions to the building of an efficient overlay network; third, we give recommendation for a future implementation.

1 Decentralized Collaborative Filtering and Searching

Basically, the collaborative elimination of incorrect or unsuitable information and the fostering of useful sources or items is known as *collaborative filtering*. Collaborative filtering techniques are at the heart of all web-stores strategies for predicting personalized online consumer interests thus providing them with suggestions of suited articles. If we except those based on rules, there are two main types of ratings: user-based and item-based. User-based ratings use other users experience to predict a rating for a given user. Item-based ratings use past ratings from a user to predict his rating of non yet rated items. Clearly, the item-based rating is a local measure of prediction. Since our application builds on the *cumulative* experience of users searching through related parts of the websphere, we naturally choose a user-based rating here[2].

1.1 k Nearest Neighbors, Prediction and Recommendation

In a typical scenario for collaborative filtering (CF) is that of m users $\mathcal{U} = \{u_1, \cdots, u_m\}$ and a list of n items $\mathcal{L} = \{l_1, l_2, \ldots, l_{m'}\}$. Each user u_i has a list of items L_{u_i}, which the user has expressed his/her opinions about. There exists an active user u_a for whom a CF algorithm must predict a rating (or likeness) for an item l_j. There are two complementary forms for expressing this prediction:

- *Prediction* is a rating $P_{a,j}$ whose computation is based on the existing ratings drawned from a certain group of users. The prediction uses a similarity measure ω among users to weight their respective ratings of the item l_j.
- *Recommendation* is a list of k items being recommended to the user a.

Both forms rely on the finding of k users of interest that would constitute the group from which the prediction or recommendation is computed. This problem is refered to as k Nearest Neighbors (KNN) problem. In our case however the rating problem is twofold:

- Users perform local searches with existing search engines whose results are to be rated (i.e. predicted) according to the experience of other fellow searchers[3].

[2]There would be several more options to be studied, such as log-based ratings[8].

[3]The use of bots is possible. Here users and bots are interchangeable.

- Users progress within their searches by performing new but related searches. These searches are shared and rated according to other users' ratings.

Therefore, the user-based rating strategy must be adapted to two different types of information: URLs and searches (i.e. sets of keywords). We refer to the first as *collaborative filtering* (formally, a task of prediction), and to the second as *collaborative searching* (formally, a task of recommendation). However, it must be noted that there is an obvious correlation between a search and the rating of the URLs that are its results. This means that for different searches whose results' intersection is non null, it is to be expected to observe a high variance on the ratings for the same URLs.

1.2 Similarity Measures

So before moving forward to the basic solving of the KNN and its adaptation to our application, two similarity measures are needed: one among the user ratings; the other among the searches. Both measures rely on a metric associated to the respective spaces: the space of all possible user ratings; the space of all possible searches. In both cases we list a few useful similarity measures.

1.2.1 Similarity measure among user rankings

We refer to the rating of a user i over an item (a URL) j as $r_{i,j}$. The Pearson's correlation coefficient is a measure of the linear correlation between two populations. It is expressed as the ratio of the covariances with the standard deviations. In our case it can be expressed as follows:

$$\omega(a,i) = \frac{\sum_{j \in L_{ua} \cap L_{u_i}} (r_{a,j} - \bar{r}_a)(r_{i,j} - \bar{r}_i)}{\sqrt{\sum_{j \in L_{ua} \cap L_{u_i}} (r_{a,j} - \bar{r}_a)^2 \sum_{j \in L_{ua} \cap L_{u_i}} (r_{i,j} - \bar{r}_i)^2}}$$

where $\bar{r}_i$ is the mean over all of user's i ratings. Note that if users have no rated items in common, the correlation is null.

Similarity can also be measured in term of 'distance' from one user to the other. This necessitates defining a distance D over the space of user's rankings. Here we can either use an Euclidian distance or a normalized Euclidian distance -(Mahalanobis). Then we can express ω as:

$$\omega(a,i) = \frac{\exp -D(a,i)}{\sum_i \exp -D(a,i)}$$

Note that a given group of users is required for computing the normalizing factor.

1.2.2 Similarity measure among searches

Similar measures can be used among searches. For computing the Pearson's correlation coefficient, either a rating is computed for each search based on the rating of its result URLs, either users can rate a whole search. However, distance-based rating seems more promising in this case because using the Hamming distance among keywords (number of different bits or characters) allows to not rely on another set of ratings. The implicit assumption here being that similar sets of keywords would lead to similar results from the search engines[4] and thus similar user ratings. For two searches s and s' in the space of all possible searches $\mathcal{S}$, we $\phi(s,s')$ the measure of the similarity between s and s'.

[4]Of course this is often not true, e.g. the case of mistyping a popular keyword.

1.3 Distributed Collaborative Filtering

Considering our application, it is natural to consider that the k users of interest are the users that belong to a same *search group* (SG). While the building of SGs is formally described later on, in the following we will refer to users that belong to a given SG sg as the $u_i \in sg$. This must clearly been understood as a heuristic for the solving of the KNN problem in our domain of application. Also we note $sg(s)$ the SG that corresponds to search s. We will see later that it is advantageous not to have a bijection between the space of all searches and the space of search groups.

1.3.1 Collaborative filtering

For now, noting s the search being performed by an active user u_a, and formalizing a simple KNN problem, we have:

$$P_{a,j} = \bar{r}_a + \sum_{u_i \in sg(s)} \omega(a,i)(r_{i,j} - \bar{r}_i)$$

This relation says that the prediction for user a on item j relies on the mean deviation on this particular item of other users (in the same search group) from their mean rating over other items. Obviously, a good prediction relies on both a decent number of users and a decent number of ratings per user (so the deviation bears a good significance).
Therefore it makes sense to base the prediction on a stream of recent and past ratings to increase its accuracy. However, given the fast changing space of certain portions of the websphere (news sites, blogs, wikis, ...) it is likely to see ratings progress or decay over time. The prediction can include a geometric recessing horizon factor for lowering the importance of old ratings. Here:

$$P_{a,j} = \bar{r}_a + \sum_{u_i \in sg(s)} \gamma^{d_{i,j}} \omega(a,i)(r_{i,j} - \bar{r}_i)$$

where $\gamma \in [0,1)$ and $d_{i,j}$ a measure of the time since rating $r_{i,j}$, e.g. in number of days.
Now, as stated before, the ratings of URLs are conditionned upon the search they've been obtained from. This means that a rating r depends on a search s and therefore should be noted $r(s)$. Taking this into account, we have:

$$P_{a,j} = \bar{r}_a + \sum_{u_i \in sg(s)} \phi(s,s')\gamma^{d_{i,j}} \omega(a,i)(r_{i,j}(s') - \bar{r}_i)$$

where for each $r_{i,j}$, s' is the search it has been drawned from.

1.3.2 Distributed computation

There has been at least one work on distributing a CF computation over a P2P network [5]. What distribution implies is that first, the search group $sg(s)$ (this is the object of the next section); second, the $r_{i,j}$, $\bar{r}_i$, $d_{i,j}$ and s' must be retrieved from the peers on the network.

1.4 Distributed Collaborative Searching

Here we express collaborative searching (CS) as the action of predicting a rating on a whole search (i.e. represented as a set of keywords). We treat it in a similar manner as CF (however, it is possible there are

better solutions). We have:

$$P_{a,s'} = \bar{s}_a + \sum_{u_i \in sg(s)} \phi(s, s')\omega(a, i)(s_{i,j} - \bar{s}_i)$$

where s' is a search being recommended to user a and s his currently performed search.

$$\bar{s}_i = \frac{\sum_{s \in S_{u_i}} \frac{1}{|s|} \sum_{j \in s} r_{i,j}}{|S_{u_i}|}$$

where S_{u_i} is the set of searches having been performed by user i (this can be reduced to the searches performed within a given search group). Obviously this model is of poor value because new searches, i.e. searches that have no strong presence in the user records will obtain ratings that have poor accuracy. What is needed here is a probabilistic model for predicting the relevance of a search given the set of past searches performed by the user. This would be a whole lot different, so we leave it at this for now. However, we'll see in the remaining of the paper that there are ways of making simple recommendations (i.e. answering a KNN problem over searches) with a properly built network structures.

2 Adapting a DHT-based overlay network

In this section we give reasons for and develop a structured distributed index for the storing of users' local search activities. We adapt network nodes data structures for our purpose and extend very recent research on locality sensitive index generation for use in a distributed structured framework.

Existing search engines map links to queries, i.e. they index documents and links. In our case the mapping is to users (i.e. computers) for collaboration. So we're offered two choices, either map the queries or the URLs[9] to the computers that have used or visited them. Now, under the hypothesis that for each URL, as indexed by a search engine, there exist at least one query that leads to it[5] (i.e. the URL is reachable) then it is easily seen that the space of all queries is larger than the space of all URLs (since they are queries that return no results). However, in general a URL does match less queries than a query matches URLs. Follows a trade-off between the number of available computers and the mean storage available on each computer: mapping URLs to computers leads to a high distribution of the information with very light load on most of the computers on the network; mapping queries to computers allows to store more information on less computers with a higher required storage capability. In consequence, given the fact the storage space is not the predominant problem these days, and that we can't expect our application to become a too popular service quickly, the second option seems to be the right one, and in the following we explain how to efficiently map queries on computers on the network.

2.1 Why a Distributed Hash Table (DHT) ?

In our application each node of the network corresponds to a computer along with a user querying a search engine in order to find a set of webpages whose content fits his expectations or needs (answer to a question, lookup for a portal to precise information, lookup for a local service, gathering information for an exhaustive coverage of a past or ongoing event, testing the existence of a theory, ...). The idea is that with the geographical spread of the Internet it is natural to expect several individuals to be digging the websphere with

[5] Given the current search algorithm available to the public, this is a reasonnable assumption for text-based documents only. Documents made of pictures or equations only may not be reachable.

similar objectives in mind at the same time. As they progress through their search they implicitly discard uninterested links, read and/or bookmark the interesting ones. As these information related to these activities can more or less easily be encapsulated and shared with CF as described in the previous section, remains the geographical sharing of this information among computer nodes. This can naturally be done through a P2P overlay network. The idea here is that the structure of the network is used to locate and regroup user's ongoing searches (e.g. sets of keywords) while their interests on the resulting URLs are shared through direct connections among the peers. Thus an answer to a user query is a set of users that belong to the same search group.

Following recent surveys [7], robustness analysis [2] and current trends in the development of filesharing applications over P2P networks [6], *structured* routing along with *decentralized* indexes seems to offer the best performances, routing a query in an overlay network of N nodes in $O(\log N)$ hops (i.e. nodes in the path between the source of the query and the node with the best answer). The archetype of these system is a Distributed Hash Table (DHT).

Basically a DHT is a mapping from the space of queries to the space of computers in the network. As in a standard hash table, the mapping function is a hash function. In the distributed case however, it must be clearly understood that the type of hash function determines the structure of the overlay network. This is of primary importance since that in application to searching it is very much welcome that results to similar queries are grouped together: this stems from the fact that most queries are best answered with a set of results (usually the k best results). When this applies, the search problem can be identified to a k Nearest Neighbors problem (KNN).

2.2 Implicit groups vs. keyword searches combinations

Before detailing the basic structure of the overlay network, we study two options in the choice for the representation of the space of queries. As mentionned above, the hash function of the DHT represents a mapping from the space of queries to the space of computers in the network, i.e. each computer is given a key and becomes a node of the overlay network. We note $\#N$ the key for node N. We've thought of two simple representation of query space: (i) queries are single keywords only; (ii) queries are chains of characters (including spaces) with an imposed maximum size.

Let's study the implications of option (i). First, it presents the advantage that looking for keywords we can expect that the pool of users returned and thus the search groups formed would be relatively large, which in general is a factor of increase in the prediction's accuracy (see previous section). Second, it implies that the load over the network would be very unbalanced: nodes that correspond to popular keywords would easily be overloaded. Third, complex queries (i.e. made of more than a single keyword...) would have to be assembled in the network, i.e. users ids fetched from each node are combined through intersections and unions with nodes content (through direct connections) before the results are sent back to the user. Alternatively, all results can be returned and the user node would then do the job, while obvioulsy unecessarily clogging the network. This problem has been studied in the litterature for a simple reason: it is equivalent to distributing a regular search engine. Thus, observing the rapid expansion of search engines use, several groups of people started looking into combining information retrieval (IR) with P2P architectures. However, it appears that solutions fare poorly in general w.r.t. centralized search architectures. Thus [6] shows that an index partition by documents is more efficient than a partition by keyword even if they choose to improve on the latter. The rational behind this is in fact that partition by keyword leads to both expensive communications among peers for intersecting the results (the case of a conjunction of keywords) and that some keyword results flood the

[6] Even the popular bittorrent protocol is moving towards a DHT based implementation.

network with gigabits of data. This can be mitigated but the overall complexity remains within an order of magnitude to feasibility. Partition by document does not really apply to us as in our case nodes are not storing and sharing any file fragments[7]. Therefore option (i) doesn't look too good for us.

Now, let's review option (ii). Here queries are chains of characters and users that used them are stored as such in the DHT. Now, this presents the obvious disadvantage that search groups are made smaller. However, the load over the network nodes is more well balanced. No more reduction operations are needed to be performed on the network neither. In fact, this option is the easiest one to start building a basic application on. Later we look into enlarging the user database to each node by regrouping similar queries.

2.3 Basic overlay structure for CF

Basically, an overlay network over a DHT distributes the storage of data by having a node hosting the data pieces whose keys are closest to this node's key (than those of the other nodes). In our case, the network address of a user that is performing a local query on a search engine is stored on the node whose key matches his local query the best.

2.3.1 Node data structure

Basic data structure for each of the network node follows. First, each node must store a certain number of his neighbors' addresses as this constitutes the mesh of the overlay network. We refer to it as the routing table. Second, it must store addresses to nodes whose data match their key. We refer to it as the data table. Both tables have rows of the form (#search_id, {ip_addr1, ..., ip_addrn}).

Additional local data structures (for sharing over direct connections) include a mapping of searches to links and ratings whose rows are of the form (#search_id, $\{(l_1, r_1), \cdots (l_n, r_n)\}$).

2.3.2 Routing

Now, since nodes are getting in & out of the network at all times, more robustness is needed and modern DHTs in fact store data on the p nodes whose keys are closest as that of the original target node[2]. This means that a node's data table does not forcibly reflect the addresses of nodes whose local queries are closest to this node's key. But this is of no importance given the DHT routing algorithm. Here a query from a node to the DHT is a lookup for a search group given the node's local query. At each node of the network, the query is routed by sending it to the neighbor with the closest key to the local query. Once no other node than the current node can be found, the target node grabs the data from his p (or less) neighbors and returns it to the query root node through a direct connection. The two main functions are $put(s)$ and $lookup(\#K, m)$ that respectively push a local query s into the DHT, and retrieve a search group (based on m nodes) from a DHT based on a hashed local query $\#K$. A lookup in the DHT corresponds to $O(\log N)$ RPCs.
In general in our application both a push and a lookup are performed over the same local query, so the two functions should be merged.

[7] Nodes store user ratings on URLs so it would be possible to build an index based on this information, such as in [5]; however, this only proves to be a heuristic to the solving of the KNN problem. We show later that it is possible to get closer to the optimal solution by using other methods.

1: $N = N_0$, where N_0 is the local node.
2: **while** $\#N$ is not the closest to $\#K$ **do**
3: $N \leftarrow$ closest node to $\#K$ in N's routing table.
4: return N's m neighbors whose keys are the closest to $\#N$ (from N's routing table).

Algorithm 1: $lookup(\#K, m)$

1: (make a local query s on a web search engine).
2: hash s into a key $\#K$ on the current node N_0.
3: $lookup(\#K, p)$.
4: **for** the p nodes **do**
5: Store $(\#K, ip(N_0))$.

Algorithm 2: $put(s)$

2.4 Enhancing the overlay structure with locality

The basic DHT structure as described above does look up for exact query matches: a lookup returns a certain number of computer IP addresses whose local user's machine connects to and asks for exact query lookups to their respective data tables. However, we can expect close or similar queries[8] to return very correlated results, and therefore be of interest for the same users. Intuitively, a search group should thus be based on a set of similar queries. So as stated before, this comes back to having an imperfect matching between the space of queries and the space of computers, i.e. several queries should refer to the same computer. In the DHT framework, this is a property of the hash function. What we express here as a desirable property is in fact refered to as the *collision* rate in the literature on hash tables and functions: for cryptography or efficient storage for example, it is highly desirable that no two hash values are the same for two different items. However, in certain applications like KNN, it can be of interest to group items together by having their hashed values collapsing for items within a certain bounded space. This is what we want to do here.

Now, let's imagine we have such a hash function that takes a query and returns a real value, and that this value would be the same for all queries within a certain range (i.e. according to a distance among strings, such as Hamming or Levenshtein distances). Such a partitioning would forcibly be adhoc because there would be queries that could have belonged to different search groups and that won't. So, what this means is that the hash function must be probabilistic: i.e. the same query should be distributed among the search groups in relation with its distance to other queries. In other words, the hash function's collision rate should reflect the closeness of queries by distributing them accordingly among computers on the network. In application that would mean that first, each query would have to be pushed several times in the DHT; second, that each query must be looked up several times. Results are a set of computers (i.e. search groups and users) ranked by their rate of occurence within the result list that is directly reflecting their respective distance to the looked up query.

Strangely enough, it seems that only recent work on hash tables and functions has explored such functionalities [4].

[8] Intuitively there is a difference between close and similar queries: they can be close by the results while not similar in the keywords used. Here we in fact only refers to similar queries. Closeness is a more diffi cult problem related to information an ontologies.

2.4.1 Locality sensitive hash functions (LSH)

Recent work on LSH can be found in [3]. Formally, a hash function is locality-sensitive if the probability of the function colliding on two different queries is decreasing with the distance between these queries.

Basically, the LSH is built upon a specific family of probability distributions, the p-stable functions. The weighted sum of random variables that follow a p-stable distribution D has the same distribution as a single random variable with distribution D but weighted by the l_p norm of the weights. Using l_2 enforces a proportionality with the Euclidian distance.

In practice, the colliding rate is amplified within a certain range R, i.e. that the rate is increased within a ball of radius R thus amplifying the gap between this region and the rest of the space. This is very useful to create small pockets of very closed queries while keeping the property of having the same query belonging to several pockets. This is achieved by concatenating a 'family' of LSH for generating the key of a single point (or query). Now, this must be repeated with L families for each point (i.e. query): this comes to sampling and dispersing the points among different pockets to create local patterns of hashed values. Thus, a lookup in the LSH is made of L lookups, each returning a set of elements. It is prooved that the solution to the KNN problem is within the union of these sets.

In our application the benefit of using LSH functions could be huge, as L lookups for a query s would return the N closest queries on the network along with their associated users. However, this is to be more precisely studied because the computational load and the memory requirements for LSH are way higher than for regular hash tables.

2.4.2 Distributed LSH

There is no work on distributed LSHs that I know of. In a distributed LSH (DLSH) the major change compared to a DHT would be in the local data structures of each node. In fact, an additional data table that we refer to as the locality-sensitive table is made necessary. Each of its rows is of the form (#Key1, #Key2, {#search1, ..., #searchM}). The later column refers to data within the data table.

3 Appendix

3.1 Lexicon

Lexicon is in table 1.

3.2 Social trends and ideas

It is unlikely that users rate a high percentage of the visited URLs. However, it is easy to put a bot construction kits in their hand that they could program in a similar way as a spam filter.

Certainly in such an application users would try to foster their own websites and businesses by massively overating these URLs. One solution to mitigate this behavior is to maintain special ratings or even 'wiki' boards that would flag the obviously over-ranked information. Other techniques could be used here.

References

[1] An architecture for peer-to-peer information retrieval. Technical report, EPFL, 2004.

rating	vote of a user on a search result (URL) or a search proposition.
prediction	prediction of the vote of a user on a URL or a search, based on those of other users, and used as the likelihood of the usefulness of that URL or search for that user.
collaborators	the set of users a given user is directly connected to, and exchange URLs and searches with (local view).
search group	the whole set of users that are searching on a given (set of) queries (or topic) together.
local query	the query a user is performing on his machine through a standard search engine.
query	a query on the DHT network.
routing table	node's table of p closest (in the key space) neighbors' ip addresses.
data table	node's table of ip addresses to the nodes (related by their local queries).

Table 1: Lexicon

[2] J. Cates. Robust and efficient data management for a distributed hash table, June 2003 2003.

[3] M. Datar, P. Indyk, N. Immorlica, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *In Proceedings of the Symposium on Computational Geometry*, 2004.

[4] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *In Proceedings of the 25th VLDB Conference, Edinburgh, Scotland*, 1999.

[5] P. Han, F. Yang, and R. Shen. A novel distributed collaborative filtering algorithm and its implementation on p2p overlay network. In *Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia*, 2004.

[6] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. the feasibility of peer-to-peer web indexing and search. In *In 2nd International Workshop on Peer-to-Peer Systems (Berkeley, California, 2003).*, 2003.

[7] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia, September 2004.

[8] J. Wang, A. P. de Vries, and M. J.T. Reinders. A user-item relevance model for log-based collaborative filtering. In *European Conference on Information Retrieval (ECIR 2006)*, 2006.

[9] Kun-Lung Wu and Philip S. Yu. Latency-sensitive hashing for collaborative web caching. *Computer Networks*, 33(1-6):633–644, June 2000.