Make it Fresh, Make it Quick — Searching a Network of Personal Webservers

Mayank Bawa
Computer Science Dept.
Stanford University
Stanford, CA 94305
bawa@db.stanford.edu

Roberto J. Bayardo Jr.
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
bayardo@alum.mit.edu

Sridhar Rajagopalan
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
sridhar@almaden.ibm.com

Eugene J. Shekita
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
shekita@almaden.ibm.com

ABSTRACT

Personal webservers have proven to be a popular means of sharing files and peer collaboration. Unfortunately, the transient availability and rapidly evolving content on such hosts render centralized, crawl-based search indices stale and incomplete. To address this problem, we propose *YouSearch*, a distributed search application for personal webservers operating within a shared context (e.g., a corporate intranet). With *YouSearch*, search results are always fast, fresh and complete — properties we show arise from an architecture that exploits both the extensive distributed resources available at the peer webservers in addition to a centralized repository of summarized network state. *YouSearch* extends the concept of a shared context within web communities by enabling peers to aggregate into groups and users to search over specific groups. In this paper, we describe the challenges, design, implementation and experiences with a successful intranet deployment of *YouSearch*.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems, information networks, Performance evaluation (efficiency and effectiveness); H.4.1 [**Office Automation**]: Groupware; H.5.4 [**Hypertext/Hypermedia**]: Architectures, User issues

General Terms

Algorithms, Performance, Human Factors

Keywords

Web search, Intranet search, peer-to-peer(P2P), decentralized systems, information communities

1. INTRODUCTION

The simplicity of WWW protocols (e.g., HTTP) and the maturity of those of the Internet (e.g., DNS) have combined to make web tools the preferred method of sharing information on both public and private networks. The commoditization of personal computer hardware and the ubiquity of the web-browser interface have made essentially universal content access possible. These facts have driven the adoption of personal webservers for sharing files and peer collaboration. As an example, almost 1,500 people within IBM use the YouServ [14] personal webserving system every week.

Critical to group collaboration is the ability to find desired content. While the Web offers URLs for accessing content from a named location, the popular method for locating content on the web is through search. We believe personal webservers are used in a manner which makes search even more preferable:

- Content on personal webservers has highly transient availability (even in systems such as YouServ which offer simple means of improving availability through site replication). One would be ill-advised to rely on locating content by URL when the referenced location may be inaccessible.
- Content on personal webservers is typically poorly arranged, rendering location by navigation ineffective.
- Content on personal webservers consists of a much larger fraction of non-HTML documents than the typical web. These documents typically are not cross-linked like HTML, further reducing the effectiveness of location by navigation.

Inadequacy of Web Search Tools

Despite the importance of search over content hosted on personal webservers, searches offered by the Web's existing search tools are inadequate. Any search index that relies on web crawling will omit any host that is unavailable at the time of crawling. Also, hosts that do manage to get indexed may be down at the point of querying, leading to dead results. Finally, a common usage of personal webservers is to share a document during authoring so that feedback can be gathered and suggested changes incorporated into the document. This causes documents to have intentionally small life-cycles which cannot be captured by crawling. Results are therefore bound to be stale and incomplete.

Addressing the Challenges

In this paper, we propose and describe *YouSearch*, an easy to use, low cost application that seamlessly integrates with a personal webserver. A YouSearch-enabled webserver provides a true web-based content sharing system by allowing fresh, fast and complete searches over personal content.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

Motivated by the desiderata of freshness, we opted for a Peer-to-Peer (P2P) paradigm in designing YouSearch. In this architecture, each webserver is enhanced with a search component consisting of a content indexer and a query evaluator. The indexer regularly monitors shared files, immediately updating its local index when changes are detected. Queries are forwarded to local indexes for evaluation at run time. Only clients that are available at query time respond, ensuring live results.

The need for speed and completeness led us towards a hybrid architecture in which the P2P network is augmented with a light-weight centralized component. Peers maintain compact site summaries (in the form a Bloom filter [16]) which are aggregated at a centralized registrar. These summaries are queried so that searches target only the relevant machines. YouSearch does not rely on query flooding or other routing-based schemes that are subject to network fragmentation and limited search horizons. Peers help reduce query load on the system by caching and sharing query results. They also cooperate to maintain freshness of the summary aggregation. This minimizes the role of centralized resources for low cost and graceful scaling.

Enhancing the Shared Context of Users

Users in a private network often organize themselves into groups with a shared goal (e.g., project teams) or a shared interest (e.g., music). YouSearch exploits this shared context to enhance search through inter-user cooperation. Web hosts can be aggregated into overlapping (user-defined) groups. Searches can be restricted to content shared by members of a group. Search results within groups can be made persistent and hand edited by users. When a query is re-run within a group, the group specific persistent version of results is returned. Thus, cooperation among users is used to improve both the efficiency and accuracy of YouSearch.

Deployment Experience

YouSearch was architected to be extremely simple to use to enhance its acceptance by users. YouSearch was released within the IBM intranet in mid-September 2002. Since it does not require the deployment of expensive server hardware, roll out has been practically cost-free. At the time of writing, in less than 2 months, the system has been adopted by nearly $1{,}500$ users and the number is steadily increasing. Our experiences in such a real-life active usage scenario show that YouSearch is fast and efficient, and most importantly, satisfies users' need for search on personal webservers.

Paper Overview

We begin with a survey of related work in Section 2. Section 3 describes YouSearch from an end user's perspective. Section 4 provides YouSearch's technical details. Section 5 evaluates performance for YouSearch in the IBM corporate intranet. As an interesting aside, we discuss our attempts at easing performance tuning of a distributed system in Section 6. We conclude in Section 7.

2. RELATED WORK

YouSearch is a deployed application that combines several ideas to provide an effective and coherent solution for web searching over peer-hosted content. This section discusses the most closely related systems and proposals.

P2P File-sharing Systems

Like YouSearch, current P2P file-sharing networks (Gnutella [3], KaZaa [7]) support search over transient, peer-hosted content. We eliminated these schemes as potential solutions to our problem since they suffer from inefficiencies that limit their scalability in providing a *fine grained* search capability. Gnutella [3] performs searches by flooding each query to all the hosts in the network reachable within a fixed *horizon* imposed by a time-to-live parameter on query packets. Since searches rarely reach every machine when the network is large, query results are often incomplete. In addition, its bandwidth and compute requirements are excessive. Another cause of search incompleteness in Gnutella is due to to partitioning of the overlay topology as peers join and leave the network. KaZaa [7] improves on this basic scheme by delegating most of the work to *super peers* – machines in the network with high bandwidth and large compute resources at their disposal. Each super-peer in KaZaa has to assume query processing responsibilities for a large number of ordinary peers. Though reduced, problems of fragmentation and limited horizon are not entirely solved by this approach.

We believe that the success of Gnutella and KaZaa in spite of these problems is due to their use for music and video sharing. In such networks, the most popular songs and files are both widely replicated and the target of most queries. Documents in corporate environments do not follow such replication or query patterns. In such settings, we believe that their inherent limits on performance will prove a stumbling block.

Napster [11] provided search over peer-hosted music files by adopting a hybrid scheme that is similar to YouSearch. Napster required the entire song list and song meta data from each peer to be centralized for indexing. This overhead forced fragmentation of Napster into islands of servers whose peers could not communicate. Such centralization of the term index becomes even more infeasible when terms arise from both file names as well as document contents, as in YouSearch.

Perhaps more important than the technical details is the philosophy of use behind current P2P systems. These systems form closed communities of users. Even people who wish to merely access or search shared content must install special purpose software that uses a proprietary protocol. As a result, each such protocol creates a partition of shared data that is inaccessible to users of the others. YouSearch, in contrast, is at its core web-compatible, and closely mimics the existing user experience of conducting a web search.

P2P Research Proposals

There have been many recent proposals for P2P systems and architectures that have yet to progress into deployment and use by a significant number of users. One such class of systems includes structured overlay networks called Distributed Hash Tables (DHTs) [24, 19, 26]. DHTs provide efficient key-value lookups across peers, but DHTs alone do not support content search.

PlanetP [18] is a research project that proposes another searchable P2P network. As in YouSearch, each peer constructs bloom filters to summarize its content. PlanetP peers then gossip with each other to achieve a loosely consistent directory of every peer in the network. In contrast, YouSearch peers exploit a designated lightweight registrar as a "blackboard" for storing network state. The advantage is that registrar-maintained state is much more consistent and can be more efficiently maintained. The registrar also performs other useful functions like locating result caches that would be impractical to provide via a gossiping scheme.

Webserving Tools

Several organizations provide webserver software that users can install on their machines to serve their own websites [1, 10, 8]. However, most do not provide an integrated search facility, let alone one capable of spanning many transient sites. Solutions for endowing such webservers with a search facility include ht://Dig [6] and SWISH-E [12]. The former is unsuitable in the presence of transience since it is crawl-based, the latter provides for indexing and searching of content over just the local site running the webserver.

One webserver supporting search across transient peers is BadBlue [2]. BadBlue can be used to search over a (private) network of hosts running BadBlue or other-Gnutella compliant software. In relying on the Gnutella protocol for searching the network, BadBlue suffers from the problems already outlined above.

Collaborative Applications
Collaborative work tools such as XDegrees [13] and Groove [5] provide secure file sharing among knowledge-workers. Groove does not enable search over files shared by a large number of peers. A press release at the XDegrees website mentions the ability to search over shared content but its technical details are not available.

Bharat in [15] proposed SearchPad as an extension to the search interface at a client to keep track of chosen "interesting" query results. The results are remembered and used in isolation by each individual client. The tool was found to be helpful for users in a study conducted in a corporate setting. YouSearch enables users to record and share results with other members in their groups. We believe that such a shared context among group members makes this functionality even more useful.

Caching of Resources at Peers
There have been several proposals for caching resources at peers. CoopNet [23], Backslash [27] and PROOFS [28] are proposals for a P2P caching scheme to address flash crowds at a host. Squirrel [21] and MangoSoft's CacheLink product [9] propose organizing a P2P cache for web objects. YouSearch exploits peer resources to cache query results. The distributed search interface of YouSe arch provides a natural means for distributing caching. Its use of a centralized registrar makes the discovery of cache locations simple and effective.

3. END-USER PERSPECTIVE

In this section we discuss the experience of a typical end user of the YouSearch application as it is currently deployed within the IBM intranet. We highlight how the usage characteristics of YouSearch motivate our design choices.

The Web Search Metaphor
The primary interface to the world wide web for many users is the search form offered by any of a number of web search engines. Because of its ubiquity, expectations of function for such an interface over personal-webserver hosted content is set by what is available on the web at large. The YouSearch interface preserves these expectations by closely mimicking familiar web search interfaces in both appearance and function as we discuss below.

The Search Interface
Each YouSearch-enabled peer provides a search form both in its standard "folder listing" template as well as within a dedicated search page (Figure 1). From the dedicated search form, the user can specify whether the search is to go across all hosts or be restricted to the local host. In keeping with user expectation, the semantics of the search form is modeled after Google [4], with `phrase match` (by way of enclosing phrases in quotes), an `exclusion (-)` operator, a `site` operator for restricting searches to a designated host, and so on.

Notice that unlike web search engines, YouSearch offers no single centralized query form. Instead, each participating host offers its own web-accessible search interface. Thus, our solution is an amalgam of a P2P execution infrastructure and the user expectation of a "universal search form."

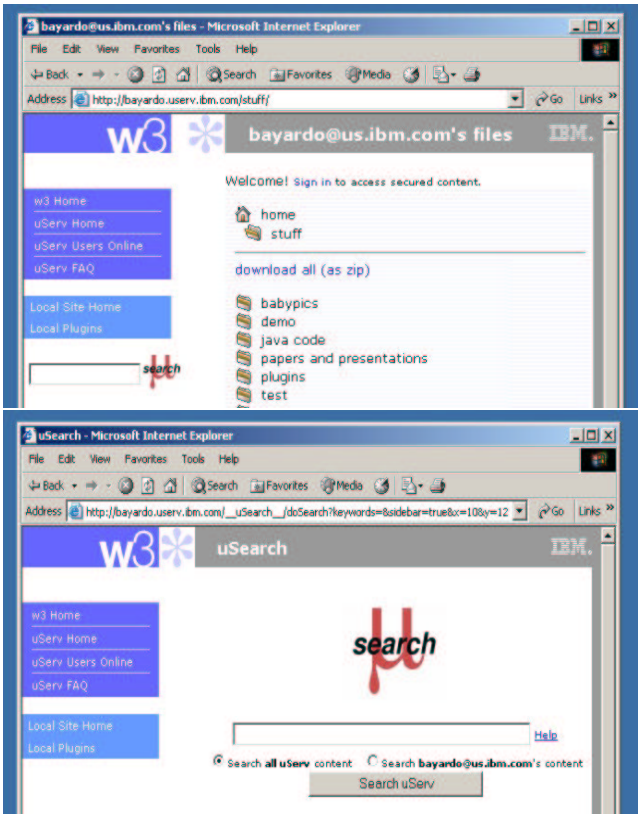

Figure 1: *[YouSearch interface]* A user browsing a YouSearch enabled website can issue queries using the above form-based search interfaces.

Sharing Searchable Content
Traditional web search engines index only content that is reachable via URLs. Most web servers allow "hiding" the URL's of some web-accessible files by placing an "index file" (`index.html`) in the directory containing them, since the index file will be served in place of a directory listing. Such *hidden files* are thus not indexed by web search engines, and many people expect such files to remain hidden to all but those who know the exact URL. YouSearch peers by default will preserve this expectation by indexing only publicly shared, non-hidden content. Files are searchable not only via the keywords that appear in their filenames/URLs, but also through the content of the files. While access controlled content is not currently searchable, we plan to allow searches by authenticated users over such content in the future. Peers update their local index every two minutes to ensure freshness. For peers which share huge amounts of content, this update interval can be increased to reduce processing load.

Viewing Search Results
After executing a search, a results page (Figure 2) is displayed as soon as enough results are obtained from the underlying network. While the user views the available search results, the system continues to gather additional results in the background. This provides quick response times while preserving the web oriented interface. While a search is underway, the results page will indicate the total number of results found up to that point. The precise number of results is displayed once the search runs to completion.

The hits from each individual host are appropriately ranked using standard text search metrics. Though not currently implemented,

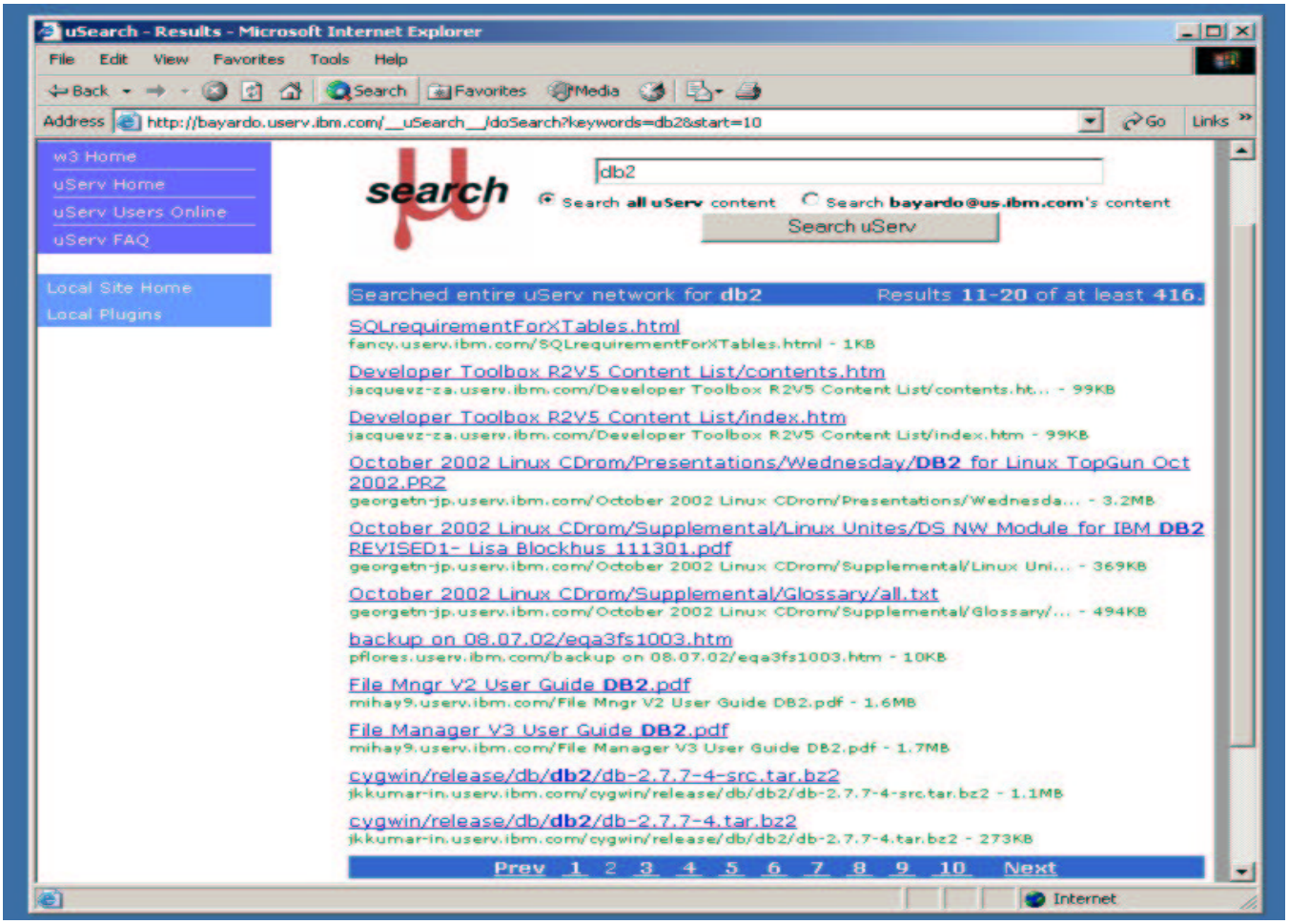

Figure 2: ***[YouSearch results display]*** **The results to a query are displayed as they are being gathered. Notice the "of at least" message at the top right corner of the results page.**

background rank aggregation of inter-host results [20] could cause the most relevant (unviewed) results to percolate upwards in the ranking even if discovered later. The ranking of already-viewed results, however, should be frozen to preserve back and forward semantics of web browsing.

Searching within a Shared Context

Users who share some context often know "where" information is, but not which of a number of documents it is in. YouSearch leverages this in two ways. First, by allowing search restricted to local hosts, and second, by supporting user defined "groups." Group restriction is enabled via a `group` operator that designates a user defined subset of hosts that are to be considered (or excluded if preceded by "-") when searching. For example, the query *pdf group:YouSearchTeam* will identify all pdf files (and files containing the term *pdf*) that are shared by peers in the *YouSearchTeam* group. Peers are free to define and manage groups of their choice. By leveraging the existing IBM corporate system for managing user group memberships, users have immediate access to many group definitions that have already been defined for other applications.

Usage Characteristics

YouSearch's low barrier to conducting searches might be seen as problematic by users of other P2P content sharing systems, where users who do not share content are often barred from conducting searches and/or downloads to encourage sharing by all participants. YouServ, however, differs significantly in its usage characteristics than these other networks. In YouServ, users are identified in the URL of every file they share. This accountability, particularly when deployed within a work environment, discourages the use of YouServ for illicit purposes such as copyright infringement. YouServ is instead regularly used for constructive purposes, such as sharing of work-related documents and web content, sharing family photos, creating informative web-hosts for various work tasks, and so on. With YouServ, users typically *want* others to use their hosts both to execute searches and to download their content, whether or not these others are actively sharing content themselves.

Deployment Status

At the time of this writing, YouSearch has been deployed for almost 2 months. Over the last week, nearly 1,500 people have made their content available to the search system. The full set of functionality described in this paper is being gradually rolled out. For example, the initial deployment of YouSearch provided conjunctive queries, exclusion and `site` operators. A month later we provided users with the `group` operator. The feature allowing sharing of user-recommended results among group members is implemented but not currently deployed. We intend to wait for users to familiarize

themselves with the `group` operator before its release.

4. INTERNALS OF YOUSEARCH

In this section we detail the YouSearch architecture and its implementation. We overview the various components and then discuss how they interact to provide the features discussed in the previous section. We conclude with a discussion of the community's role in maintaining network consistency.

4.1 System Overview

The participants in YouSearch include (1) *peer nodes* who run YouSearch-enabled clients, (2) *browsers* who search YouSearch-enabled content through their web browsers, and (3) a *Registrar* which is a centralized light-weight service that acts like a "blackboard" on which peer nodes store and lookup (summarized) network state.

The search system in YouSearch functions as follows. Each peer node closely monitors its own content to maintain a fresh local index. A bloom filter content summary is created by each peer and pushed to the registrar. When a browser issues a search query at a peer p, the peer p first queries the summaries at the registrar to obtain a set of peers R in the network that are hosting relevant documents. The peers in R are then directly contacted by p with the query to obtain the URLs for its results.

To quickly satisfy any subsequently issued queries with identical terms, the results from each query issued at a peer p are cached for a limited time at p. The peer p notifies the registrar of any cache entry it maintains. This allows peers other than p that happen to receive the same query to locate and return the cached results instead of executing the query from scratch.

Peers in YouSearch use the centralized registrar as a blackboard by posting the state of their node content for other peers to query. This way, the registrar can be used to avoid costly and often ineffective flooding of queries to irrelevant peers, and also for efficient location of relevant result caches. Though its role is important, the registrar remains a mostly passive, and hence light-weight entity. The peer nodes perform almost all of the heavy work in searching.

4.2 Indexing Shared Content

We now describe the indexing process in detail. An indexing process (Figure 3) is periodically executed at every peer node. The process starts off with an *Inspector* that examines shared files in accordance with a user specified *index access policy*. Each shared file is inspected for its last modification date and time. If the file is new or the file has changed, the file is passed to the *Indexer*. The Indexer maintains a disk-based inverted-index over the shared content. The name and path information of the file are indexed as well. Our implementation uses the engine described in [17] for indexing local node content, which provides sub-second query response times, efficient index builds, and excellent results ranking.

The *Summarizer* obtains a list of terms T from the Indexer and creates a bloom filter [16] from them in the following way. A bit vector V of length L is created with each bit set to 0. A specified hash function H with range $\{1, ..., L\}$ is used to hash each term t in T and the bit at position $H(t)$ in V is set to 1.

Notice that bloom filters are precise summaries of content at a peer. Suppose we want to determine if a term t occurs at a peer p with a bloom filter V. We inspect the bit position $H(t)$ in V. If the bit is 0, then it is guaranteed that a document with term t does not appear at p. If the bit is 1, then the term might or might not occur at p since multiple terms might hash to the same value thus setting the same bit in V. The probability that such conflicts occur can be reduced by increasing the length L.

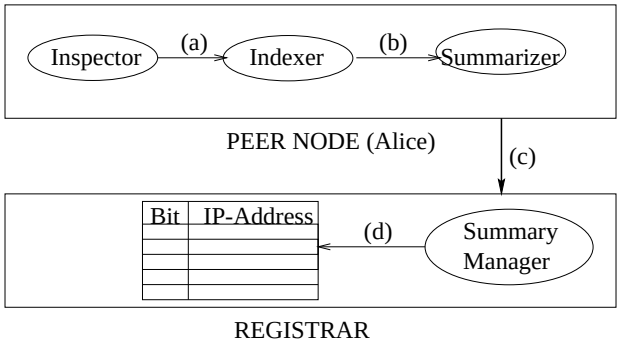

Figure 3: *[Indexing shared content]* Peer processes are run periodically at intervals that can be set by the end user.

Conflicts can also be reduced by using k independent hash functions $H_1, H_2, \ldots, H_k$ and inspecting that bit positions $H_1(t), H_2(t), \ldots, H_k(t)$ are all set to 1. Conventionally, the k hash functions are used to set bits in a single bit vector. However, in YouSearch we construct k different bloom filters, one for each hash function. It can be shown that such a design incurs a slight increase in false positives. Nevertheless, we opt for multiple bloom filters to enable further decentralization of registrar. As the network grows and the load on registrar increases, we can easily distribute load by placing the k bloom filters across multiple servers.

The k bloom filters are sent to the registrar when a peer becomes available, and whenever changes in its content are detected. The *Summary Manager* at the registrar aggregates these bloom filters into a structure that maps each bit position to a set of peers whose bloom filters have the corresponding bit set.

4.3 Querying Indexed Content

We now describe the querying process in detail. Any browser (Bob) visiting a YouSearch enabled website (Alice's peer) has a query interface from which he can search the network.

4.3.1 Querying Global Content

Suppose Bob wishes to search all of YouSearch enabled content (Figure 4). Bob's query is received by Alice's peer node via a web interface and is forwarded to a *Canonical Transformer* that converts the query into a canonical form consisting of sets of terms labeled with the associated modifier. For example, a query of *pdf group:YouSearchTeam* will be converted to $\{\langle keywords, \{pdf\}\rangle, \langle group, \{YouSearchTeam\}\rangle\}$. The canonical query is forwarded to the *Result Gatherer*.

The Result Gatherer sends the canonical query to the registrar where the *Query Manager* computes the hash of keywords to determine the corresponding bits for each of the k bloom filters. The registrar looks up its bit position to IP address mapping and determines the intersection R of peer IP address sets. The set R is then returned to the querying peer (Alice).

The Result Gatherer at Alice's peer obtains R. If the query contained special modifiers (e.g., `site, group`), R is further filtered to contain only peers that satisfy the modifier. It then contacts each of the peers in R and obtains a list of URLs U for matching documents. The results are then passed to *Result Display* which then appropriately formats and displays U. In order to reduce the latency perceived by Bob, Result Display shows its results even as Result Gatherer is collecting them.

The decision to restrict query processing at the registrar to keywords was motivated by the desire to reduce processing load at the

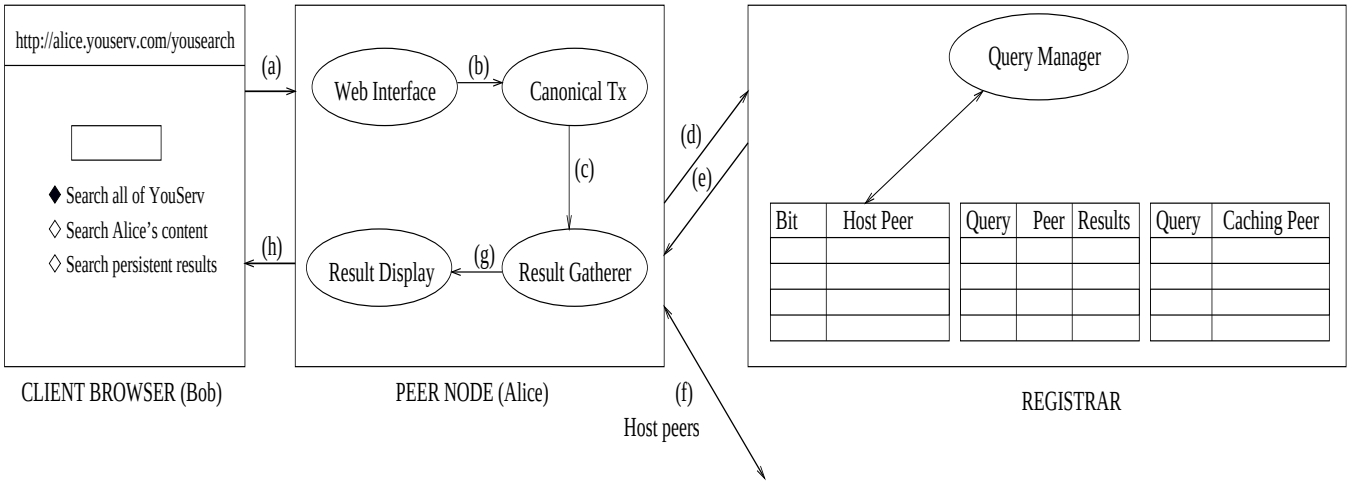

Figure 4: ***[Querying global content]*** **The registrar uses its global summary to determine a set of peers that could have documents matching the query. Alice's peer then contacts each of these peers to obtain the results.**

registrar. The trade-off involves increased bandwidth usage as R could be further filtered by query modifiers. However, the `group` modifier in particular is an expensive filter as it involves a remote procedure call to the intranet group server. By delegating modifier processing to querying peers, the incurred cost is distributed across the community. Moreover, we have now allowed a possible extension of having groups defined locally at each peer. The registrar can then be made transparent to maintenance costs as groups evolve over time.

Note that the use of bloom filters as summaries of shared content implies that R is guaranteed to contain all peers that have a matching document (no *false negatives*). This ensures result completeness. However, R can have *false positives* in which case Alice will receive 0 answers from some of the peers it contacts. The chances of such false positives can be reduced by increasing the length L of a bloom filter or the number k of bloom filters used.

4.3.2 Querying Local Content

Suppose Bob wishes to limit his search to Alice's node alone. The query is received by the web interface at Alice's peer, transformed into a canonical form and forwarded to Result Gatherer as before. The Result Gatherer recognizes the query to be a local query and looks up its local index to find documents that match the query. The index returns a list of ranked URLs U for the matching documents. The ranked list is sent to Results Display which formats and displays results as before.

4.4 Caching Query Results

As the network matures, a significant fraction of the queries will be repeated. Indeed, it has been widely reported that queries have a zipfian distribution and individual queries are temporally clustered [29]. Caching search results enables a search solution to reduce costs by reusing the search effort.

Since YouSearch has abundant resources (the computing and storage resources of all peers in the network) at its disposal, it is extremely aggressive in its use of caching. Every time a global query is answered that returns non-zero results, the querying peer (Alice's peer node) caches the result set of URLs U. The peer then informs the registrar of the fact. The registrar adds a mapping from the query to the IP-address of the caching peer (Alice) in its cache table.

Each cache entry is associated with a (small) lifetime that is monitored by the caching peer node. The caching peer itself monitors and expires entries in its cache, and informs the registrar of any such changes.

Suppose Bob asks a global query at Alice's peer node that has been cached at other peer nodes in the network. The Result Gatherer at Alice's node sends the query to the Query Manager at the registrar. The Query Manager at the registrar looks up its cache mapping for the query and determines the set of peers that are caching the query. The registrar picks one of them (Ted) at random and asks the Result Gatherer at Alice's node to obtain the cached results from Ted's peer node.

4.4.1 Querying Recommended Results

Suppose Bob searched for a query that he expects others in his group to be interested in (e.g., *how to install a printer, location of weekly meetings,* etc.). Since peers in a group share a context, Bob expects that results they are interested in will be similar to his own interests. It would be convenient for them if Bob could persist his efforts in inspecting the results to determine the most relevant answer to his own query.

YouSearch provides a mechanism by which such information can be shared by members in the group. Each global query result is displayed with a check box that allows Bob to indicate if he found the result relevant and would like to recommend it. If Bob opts to do so, the query and the selected result are sent to the registrar who maintains such mappings from query to the recommended URL. If Bob is signed in, the result is stored as a recommendation of Bob, otherwise it is recorded as an anonymous recommendation.

Bob can also query the recommended information. The query is evaluated as before except that the Result Gatherer obtains the result set of URLs directly from the registrar. The results displayed by Result Display are grouped into three categories: recommended by Bob himself, by anonymous users, and by identified users. Bob can also delete a result from the recommended information that was set by Bob himself or by an anonymous user. Thus the information pool is editable, allowing peers to modify and augment shared information.

4.5 Role of the Community

Peers take it on themselves to keep the network state consistent

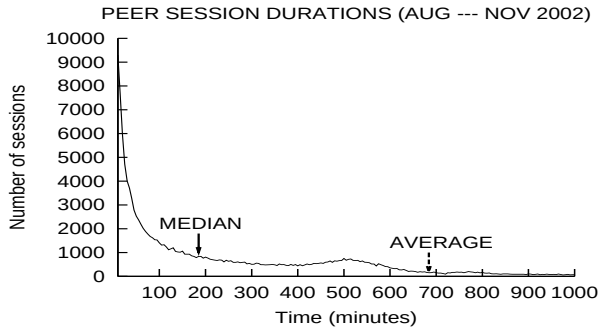

Figure 5: Distribution of session durations of YouServ peers.

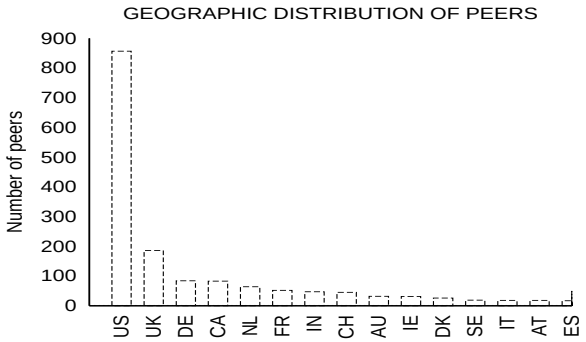

Figure 6: The top 15 of 43 countries with YouSearch users. Although USA constitutes the dominant fraction, the YouSearch enabled web is distributed across the world.

by absolving the registrar from actively having to maintain its store of bloom index and cache mappings. Each peer informs the registrar of any changes in its index or cache entries. Whenever a peer leaves the network, it asks the registrar to remove its entries from the index and cache tables.

If a misbehaving peer (Carol) neglects to inform the registrar of its changes, the mappings at the registrar become inconsistent. In such a case, the registrar would include Carol's peer node in its set of hints of peers to contact for a particular query. The peers are designed to handle and report such inconsistencies. A querying peer (Alice) will try to retrieve answers from Carol who is no longer available causing the request to time out. Alice then informs the registrar that Carol is unreachable. The registrar verifies the information by contacting Carol and subsequently removes entries for Carol from its mappings. Thus, the registrar in YouSearch serves just as a repository of state, changing the state at the explicit direction of the peers themselves.

5. PERFORMANCE ON A REAL-WORLD DEPLOYMENT

YouSearch has been deployed as a component of the YouServ personal web-hosting application within the IBM corporate intranet since September 16, 2002, with a limited beta release preceding it a week before on September 9, 2002. This section provides a look at the usage trends of YouSearch compiled to date. We show how these trends support YouSearch's design principles and our assertion regarding the importance of search within such a network.

5.1 Peer Lifetimes

YouServ was originally deployed within the IBM intranet in July 2001. A peer enables content sharing by starting the YouServ client, and disables it when the client stops. The interval between start and stop of a YouServ client is defined as a YouServ *session* for that peer. Figure 5 shows the distribution of session durations observed among the 3,055 YouServ peers that were active between July 1, 2002 and November 9, 2002. As can be seen, most of the sessions are short, with half of the sessions being less than 3 hours long. The dotted arrow in the figure points to the average session duration of 684 minutes.

YouServ offers the ability for replicating content across trusted peer nodes so that peer transience need not necessarily translate into content availability transience. The idea is that as long as some peer hosting the content is online, the content remains available, and through the same URLs. Although users are aware of this feature, only a small fraction (171 of the 3,055) of the peers made use of it. Thus, result freshness remains an important concern.

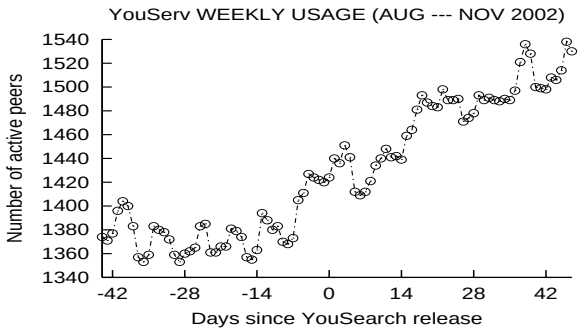

Figure 7: The number of YouServ peers that are active during a week has been increasing rapidly since YouSearch was released.

5.2 Geographical Layout of Peers

Figure 6 shows the distribution of YouSearch-enabled peers within the IBM intranet. Although the dominant fraction of peers is located within the USA, YouSearch is being used actively in 43 countries across the world. The different time zones of its users combined with short session durations causes the YouServ web to be in constant churn, rendering centralized crawling impractical.

5.3 Perceived Value-Add of YouSearch

The proliferation of YouServ peers on the IBM intranet shows that people want a simple and effective means of sharing content using the web. Recent usage statistics indicate the value-add of YouSearch. Figure 7 shows the number of unique users who actively used YouServ sometime during the week before each plotted point. Note that a significant growth in this usage metric coincides with the day YouSearch was released. Another sign of demand for search is the fact that the *beta* release alone (interval $[-7, 0]$ in Figure 7) was downloaded and used by nearly one hundred users.

5.4 Characteristics of Bloom Filters

In our IBM intranet deployment, the length of each bloom filter is $L = 64$ Kbits and the number of bloom filters is set to $k = 3$. The three hash functions H_1, H_2, H_3 are computed as follows. An MD5 [25] hash of each term at a peer is determined. An MD5 hash is 16 bytes long from which three 16 bit hash values are extracted and used as keys in the bloom filter. Note that the use of MD5 ensures that the hash is strongly random and that the resulting bloom filters are independent.

As mentioned in Section 3, only publicly shared files at each peer

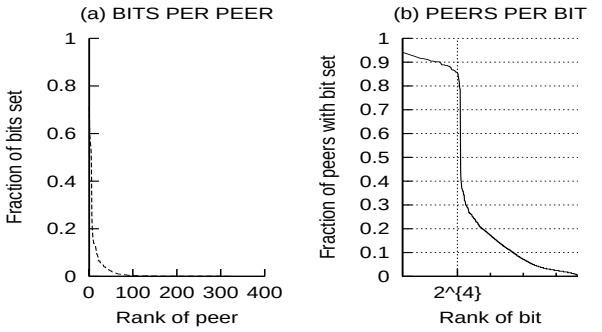

Figure 8: Characteristics of bloom filters from approximately 340 peers.

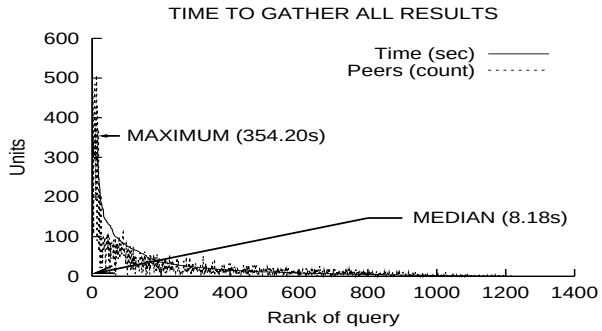

Figure 9: Time taken to gather all results. Note that the response time seen by a browser is a small fraction of this time.

are indexed. Users can also explicitly declare specific directories to be indexable or not. For each indexable file, the Indexer indexes keywords that appear in its URL. In addition, if the file is of type HTML or text, the Indexer indexes keywords within the file itself.

Figure 8(a) shows that only a few peers have a large (two-thirds) fraction of their bits set. In addition to increasing network traffic, these peers could face high query processing loads. A relatively simple solution to this problem is to create partitions of content at such peers, with distinct bloom filters summarizing each partition instead of each node. Figure 8(b) shows that most of the bits in the bloom index are highly selective, though a few bits that correspond to the most frequently occurring words (about 2^4) are set by almost 80% of peers. YouSearch could be made to filter stop words to reduce this effect.

5.5 Time to answer queries

To evaluate overall query performance, we logged statistics for global queries asked at a YouSearch peer. We formed a *query set* comprised of the first $1,500$ global queries logged during the monitoring period (September 9, 2002 to November 9, 2002). The queries were sorted based on the time taken to gather answers at the querying peer. Figure 9 shows that more than half the queries were answered in less than 10 seconds. Nearly 10% of the queries took more than a minute to be answered. Note that these times are not the response times seen by the browser. As discussed in Section 3, the results are displayed to the user while they are being gathered.

We also plotted the number of peers that were contacted for answers to the corresponding query. Not surprisingly, the curve for number of peers contacted follows the time curve closely. The jitter

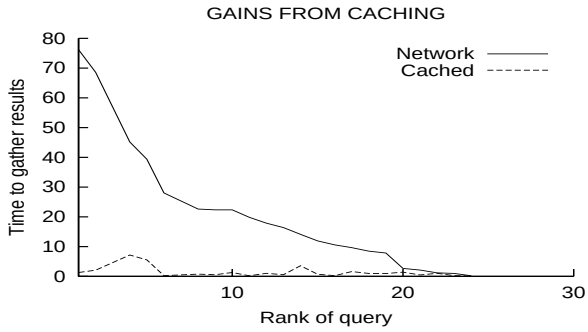

Figure 10: Time to gather results with and without caching.

in this curve can be attributed to the geographic distance between peers and the fact that even on the IBM intranet, nodes have vastly different bandwidth and latency characteristics (some connect via dialup VPN, for example). We note that the current implementation probes peers sequentially. Parallelizing such probes by contacting α nodes simultaneously will result in proportional improvements in gather times. A factor of ten in improvement is easily feasible and will bring the median gather time to a sub-second range.

Not surprisingly, the longest queries were also observed to have large answer sets. For these queries, the collection of results were in fact collected faster than the speed at which a user is likely to inspect them.

5.6 Query Characteristics

We examined queries in the query set discussed earlier in Section 5.5. A large fraction (94.47%) of queries were simple keyword queries with an average length of 1.22 keywords and standard deviation 0.58. The remaining 5.53% (83 of the $1,500$) involved an advanced feature like `site` or `group` search. About 70% of the queries had at least one answer with an average of 254.41 answers per query obtained from an average of 22.83 peers. Figure 9 plots the number of peers (size of R) that were probed to obtain answers.

We believe that the users are still adjusting to the availability of search, with a significant amount of content remaining unindexed due to YouSearch's default behavior of leaving content unindexed should it be hidden behind `index.html` files. As users become more familiar with YouSearch, more data will be made available for searching, and the fraction of successful queries will increase.

We observed that 17.38% of the queries had a false-positive peer in its result set. The average number of false-positive peers was 0.139 which corresponds to 0.6% of an average result set of size 22.83 peers. Most of these false positives are due to the few peers in Figure 8(a) that have a large fraction of their bits set.

Of the successful queries, 3.54% were served from peer caches. This value will increase as the system grows due to higher query loads. Additionally, we have been very aggressive in clearing caches: the default cache lifetime is set to 5 minutes. Increasing this default parameter will lead to improved cache hit rates, though with a slight penalty in result freshness. Indeed, nearly a third (31.31%) of all queries in our sample were asked more than once.

To better quantify the effect of caching on performance, we issued a sample of 25 queries at one peer, and then repeated these same 25 queries at different peers in the network. The second time a query was executed, results were gathered from a cache instead of gathered from scratch. Figure 10 shows the times taken in the two invocations. Clearly caching improves performance, often by an order of magnitude.

5.7 Load on Registrar

Recall that YouSearch utilizes a centralized registrar for providing an aggregated bloom filter index. In this section we analyze the communication and processing demands required of this component.

Suppose there are n peers participating in the network. Each of the n peers will send their k bloom filters of size l bits every T seconds if their content has changed, where T is the period at which crawls occur at each peer. Let f be the fraction of peers whose content changes in an interval of T seconds. The registrar thus has an average inbound traffic of $f \times n \times k \times l$ bits every T seconds.

In the current YouSearch deployment, $k = 3$, $l = 65{,}536$ and $T = 300$ seconds. We conservatively set the frequency of site changes, f, to 20%. With such settings, assuming the registrar has a T1-line bandwidth of $1.54M$ bits per second, of which 20% is consumed by networking overhead, the registrar could support $n = \frac{80\% \times 1.54Mbps \times T}{k \times l \times f} =$ **9,856** peers. Assuming a corporate private network with a T3 line capacity of $44.736Mbps$, the registrar could support $n =$ **286,310** peers. These are rather loose upper bounds given our conservative setting of site-modification frequency. Bandwidth could be further reduced by having peers send only changed bits [22] instead of re-sending the entire bloom filter with each change. The current design nevertheless easily supports the current and projected user bases within the IBM deployment.

Let us now consider the processing costs at the registrar. For any reasonable number of peers, the registrar can easily maintain the three mappings of Section 4 in main memory. For each query, the registrar performs a small number of simple lookups on these data structures which amount to easily-optimized bit-vector operations. This design permits even modest hardware to scale to tens if not hundreds of thousands of users and their queries.

6. TUNING PEER DEPLOYMENTS

We realized that tuning peer deployments for optimal performance would be a difficult job. The difficulty stems from the end user's reluctance in downloading new releases of software that is already providing a desired level of satisfaction. In this section we present a simple solution that we designed in response.

6.1 Challenges Unique to P2P Networks

While every client-side software deployment suffers from similar end-user inertia, the problem is especially severe for P2P networks. A P2P network draws its benefits from having *large numbers* of peers participate to form a *single community* of users. The large numbers of peers directly translates to a large number of individual software deployments that need to be tuned. More importantly, the single community constrains that such tuning be simultaneous across all peers: the success of the community relies on all peers using the same protocol.

EXAMPLE 6.1. *Suppose that we released YouSearch with the size of individual bloom filters as $L = 512b$. We might have found that a significant fraction of the community sets most of the bits. Each query would then be mapped to all the peers in the network causing the Result Gatherer phase to degenerate into a broadcast of the query over the entire network.*

The correct thing to do would be to increase the size of the bloom filters to 1024 *bits. However, the problem is not solved by just rolling out a new version of the software. If the new and the old versions co-existed, the same term would be hashed to a* different *bit position in both these versions. Thus the same query would be represented differently at different peers.*

While the problem could be temporarily solved by maintaining two different filters at the Registrar (one each for 512 *bits and for* 1024 *bits), the complexity of the code would increase. Further, each such tweak would result in more filters being created and maintained for backward compatibility.*

Similar to the size of bloom filters discussed in Example 6.1, there are several parameters in the code that need to be tweaked based on the usage patterns of peers (e.g., the number of bloom filters to create at each peer, the frequency of sending bloom filters to the Registrar from each peer, the duration of time-outs while contacting a peer to gather results, etc.) We label such parameters that arise from an implementation of the conceptual design *tunable parameters*.

6.2 Addressing the Challenges

We programmed YouSearch to allow distributed tuning. Each YouSearch peer is enabled with a *Tuning Manager*. The Tuning Manager works with a centralized *Administrator* to receive and affect the changes pushed out by the Administrator. The Manager creates a local state file on disk at the peer during the installation process. Signed messages received from the Administrator are interpreted, acted upon and persisted in the state file by the manager. The rest of the application reads the maintenance state to respond to the changes pushed out by the Administrator.

The various parameters that need to be tuned in the code are identified and set to values read from the maintenance state file at application launch. Changes to the values of these tunable parameters can be sent to the Maintenance Manager by the Administrator. The Manager merely overwrites the values for these parameters in the state file to affect the changes. Thus, we can simultaneously change the settings for tunable parameters of as many peers as we like, allowing us to experiment and tune the network as it evolves.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the challenge of providing fresh, fast and complete search over personal webserver-hosted content. Because of the transient availability of personal webservers and their rapidly evolving content, any crawl-based search solution suffers from stale and incomplete results. Our solution, *YouSearch*, is instead a hybrid peer-to-peer system that relies primarily on the webservers themselves to keep pace with changing network state. It scales gracefully and costs little since its centralized resource requirements are small.

YouSearch also enhances the shared context among its users. Personal webservers can be aggregated into overlapping, user specified groups, and these groups searched just as individual nodes. Any group member can persist result recommendations so that others can draw upon its knowledge.

Within two months of its deployment, YouSearch has already been adopted by nearly $1{,}500$ users. Our study of its usage in this real-life setting showed that YouSearch performs well and, most importantly, satisfies user needs.

Future work might consider allowing authenticated peers to search secured in addition to public content. Other useful extensions would include having a peer generate snippets of matching body text for its (cached) search results, exploit social networks (defined by existing group definitions) for personalized inter-host ranking, and even actively maintain cached results for the most popular queries (instead of simply timing them out to avoid staleness). Unlike purely centralized search architectures, the plethora of compute, storage, and bandwidth available to the set of YouSearch peers as a whole puts few constraints on further enhancement.

8. ACKNOWLEDGEMENTS

We thank Rakesh Agrawal for many insightful comments on this draft and YouSearch in general. We thank Dan Gruhl for designing the YouSearch logo. Finally, we acknowledge the many users of our internal YouSearch deployment for their valuable feedback. Mayank also thanks Amit Somani for introducing him to the YouServ project.

9. REFERENCES

[1] Apache's HTTP Server Project. `http://httpd.apache.org/`.

[2] BadBlue — The P2P File Sharing Web Server. `http://www.badblue.com/`.

[3] The Gnutella Network. `http://www.gnutella.com/`.

[4] Google — The Web Search Engine. `http://www.google.com/`.

[5] Groove Networks, Inc. Desktop Collaboration Software. `http://www.groove.net/`.

[6] ht://Dig – Internet Search Engine Software. `http://htdig.org/`.

[7] The KaZaa Media Network. `http://www.kazaa.com/`.

[8] Mac OS X: Personal Web Sharing. `http://www.mac3d.com/`.

[9] LAN-Based web caching for accelerated web access. `http://www.mangosoft.com/products/cachelink`.

[10] Microsoft's Personal Web Server and Peer Web Services. `http://www.microsoft.com/`.

[11] The Napster Company. `http://www.napster.com/`.

[12] SWISH-E: Simple Web Indexing System for Humans - Enhanced. `http://swish-e.org/`.

[13] The XDegrees Company. `http://www.xdegrees.com/`.

[14] R. J. Bayardo Jr., A. Somani, D. Gruhl, and R. Agrawal. YouServ: A Web Hosting and Content Sharing Tool for the Masses. In *Proc. 11th Intl. World Wide Web Conf. (WWW)*, 2002.

[15] K. Bharat. SearchPad: Explicit Capture of Search Context to Support Web Search. In *Proc. 9th Intl. World Wide Web (WWW) Conference*, 2000.

[16] B. Bloom. Space/time Trade-offs in Hash Coding with Allowable Errors. In *Communications of ACM*, volume 13(7), pages 422–426, 1970.

[17] D. Carmel, E. Amitay, M. Herscovici, Y. Maarek, Y. Petruschka, and A. Soffer. Juru at TREC 10 - Experiments with Index Pruning. In *Proc. 10th Text REtrieval Conference (TREC)*, 2001.

[18] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in Ad Hoc P2P Communities. In *Proc. International Workshop in Peer-to-Peer Computing*, 2002.

[19] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In *Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

[20] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *Proc. 10th Intl. World Wide Web Conf. (WWW)*, pages 613–622, 2001.

[21] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proc. 21st ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2002.

[22] M. Mitzenmacher. Compressed Bloom Filters. In *Proc. 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 144–150, 2001.

[23] V. N. Padmanabhan and K. Sripanidkulchai. The Case for Cooperative Networking. In *Proc. 1st Intl. Peer-to-Peer Systems (IPTPS) Workshop*, 2002.

[24] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A Scalable Content-Addressable Network (CAN). *Proc. of ACM SIGCOMM*, 2001.

[25] R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. Technical report, Network Working Group, 1992.

[26] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, 2001.

[27] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer Caching Schemes to Address Flash Crowds. In *Proc. 1st Intl. Peer-to-Peer Systems (IPTPS) Workshop*, 2002.

[28] A. Stavrou, D. Rubenstein, and S. Sahu. A Lightweight, Robust P2P System to Handle Flash Crowds. In *Proc. IEEE Intl. Conf. on Network Protocols (ICNP)*, 2002.

[29] Y. Xie and D. O'Hallaron. Locality in Search Engine Queries and its Implications for Caching. In *Proc. 19th IEEE Infocom*, 2000.