Decentralized Web Search

by

Md Rakibul Haque

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Centrally controlled search engines will not be sufficient and reliable for indexing and searching the rapidly growing World Wide Web in near future. A better solution is to enable the Web to index itself in a decentralized manner. Existing distributed approaches for ranking search results do not provide flexible searching, complete results and ranking with high accuracy. This thesis presents a decentralized Web search mechanism, named DEWS, which enables existing webservers to collaborate with each other to form a distributed index of the Web. DEWS can rank the search results based on query keyword relevance and relative importance of websites in a distributed manner preserving a hyperlink overlay on top of a structured P2P overlay. It also supports approximate matching of query keywords using phonetic codes and n-grams along with list decoding of a linear covering code. DEWS supports incremental retrieval of search results in a decentralized manner which reduces network bandwidth required for query resolution. It uses an efficient routing mechanism extending the Plexus routing protocol with a message aggregation technique. DEWS maintains replica of indexes, which reduces routing hops and makes DEWS robust to webservers failure. The standard LETOR 3.0 dataset was used to validate the DEWS protocol. Simulation results show that the ranking accuracy of DEWS is close to the centralized case, while network overhead for collaborative search and indexing is logarithmic on network size. The results also show that DEWS is resilient to changes in the available pool of indexing webservers and works efficiently even in the presence of heavy query load.

Acknowledgements

All praise be to Allah, the Lord of the worlds, who guided me throughout this research and beyond.

I am deeply indebted to my supervisor, Professor Raouf Boutaba, for his financial assistance, continuous encouragements and precious advice to complete this thesis. I would also like to express my gratitude and appreciations to Dr. Khuzaima Daudjee and Dr. Bernard Wong for their insightful comments and constructive criticisms of this work.

A special thank to the members of pWeb project for their valuable discussions and feedback throughout this research. I want to mention, in particular, Dr. Reaz Ahmed who provided continuous support in my research and guided me in writing this thesis.

I would like to thank David R. Cheriton School of Computer Science, University of Waterloo for providing an excellent academic and research environment.

Last but not the least, my deepest thanks and greatest love to my father, Khademul Karim and mother, Soheli Pervin. Whatever I do in life, I would never achieve without what they have been accomplished. My true love also goes to my wife, Rifat Jafrin, for her patience and continuous encouragement to successfully complete my thesis.

Dedication

To my great parents and beloved wife ...

Table of Contents

List of Figures

Chapter 1

Introduction

There is no doubt that one of the biggest breakthroughs in the world of science and technology was the introduction of the Internet. With the Internet, we are able to connect with other people in real time even to those who are a thousand miles away from us. The Internet enables everyone to know about the latest in fashion, current events, politics, music, etc. Not only that, we can now make transaction with other businesses or other people in just a few clicks. Search engines (e.g., Google, Yahoo) bring the Web to the hands of peoples and present requested information on the fly with only few search keywords. However, it is difficult to crawl and index the whole Web by a centralized search engine as the Internet is growing on its own pace, which mandates a search engine for indexing and searching in distributed manners.

Few research works are performed on the direction of distributed ranking of search results based on both DHT (Distributed Hash Table) and non-DHT based overlay utilizing Google's PageRank or Information retrieval techniques. These techniques do not provide complete search results with high accuracy of ranking. For this reason, existing approaches can not be utilized for developing a decentralized search engine.

The focus of this thesis is to devise an efficient decentralized Web search engine. We present *DEWS* (Distributed Engine for Web Search), which provides independent indexing, flexible searching and complete results ensuring high accuracy of ranking in a distributed manner incurring low network and storage overhead. The concepts presented in this work have been validated through extensive simulation results. We used the standard LETOR 3.0 dataset [31] to drive input of our simulation.

The organization of the rest of this chapter is as follows. The demand for a decentralized Web search engine and requirements are presented in Section 1.1.

Section 1.2 explains the motivation behind this research. The contributions of this thesis are listed in Section 1.3. Finally, the organization of the thesis is outlined in Section 1.4.

1.1 Decentralized Web Search

Internet is the largest repository of documents that man kind has ever created. Voluntary contributions from millions of Internet users around the globe, and decentralized, autonomous hosting infrastructure are the sole factors propelling the continuous growth of the Internet. According to the Netcraft[1] Web Server Survey, around 34 million sites were added to the Internet in June 2012 making the total to 697.08 million.

Centrally controlled, company owned search engines, like Google, Yahoo and Bing, may not be sufficient and reliable for indexing and searching this gigantic information base in near future, especially considering its rapid growth rate. This explosion in the number of websites is accompanied by a proportional increase in the number of webservers to host the new content. If these webservers participate in indexing the Web in a collaborative manner then we should be able to scale with the searching needs in the rapidly growing World Wide Web.

Figure 1.1 shows the essential components of a search engine. In a centralized search engine, one or more crawlers fetch webpages from the Web and send them to a *Parser and Indexer* module. *Parser and Indexer* module extracts representative information from webpages and creates inverted index for each webpage. Inverted indexes are stored in the *Index*. *Query Processor* and *Ranking Module* interact with *Index* to process queries and rank the search results, respectively. *Ranking Module* computes rank of the webpages and search results. *Query Processor* is responsible for query optimization and evaluation with the help of the *Index* and *Ranking Module*. *Search Interface* interacts with the users. It receives queries from users and presents search results to the users with the help of *Query Processor*. In a decentralized search engine, these components should be implemented in a decentralized manner. A decentralized Web search engine should meet the following requirements:

- *Flexible searching:* Decentralized Web search engine should provide flexible searching. By 'flexible searching', we refer to a capability of a search mecha-

[1]Netcraft - http://news.netcraft.com/

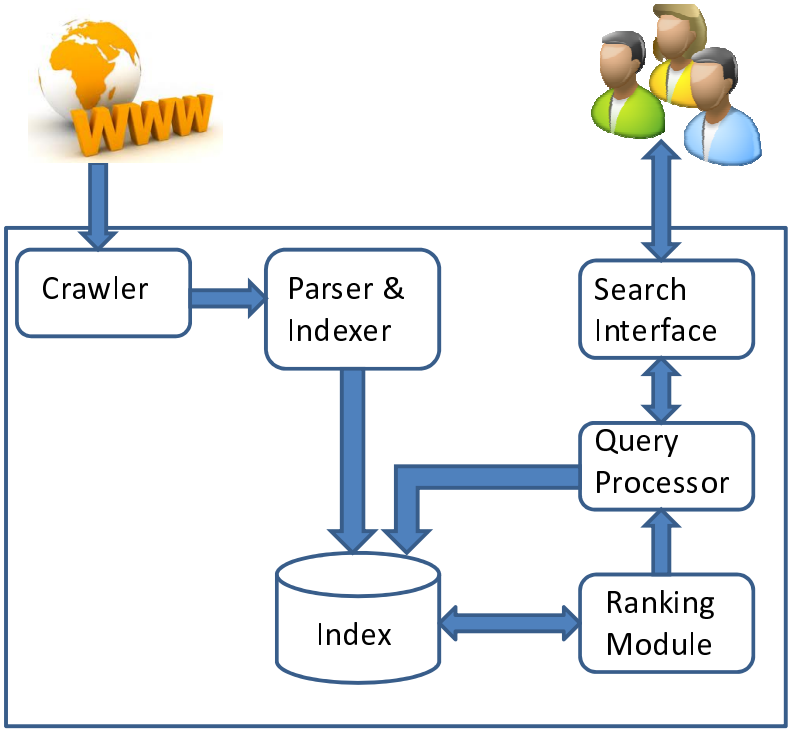


Figure 1.1: Components of a search engine

nism to handle misspelled and partially specified keywords in a query. Users may not exactly know the keywords for the requested information and may type misspelled or partially specified keywords. For this reason a search engine should provide flexible searching with high accuracy of search results.

- *High accuracy of search results:* A decentralized search engine should be able to provide requested information with most relevant results. The search engine should discover the rare or non-popular information as well.

- *High accuracy of ranking:* The search engine should present the most relevant information in the first few results. The overall performance of a search engine depends on how efficiently it determines the relevance of results with the query keywords and ranks them.

- *Distributed indexing:* In centrally controlled search engines, crawlers index the webpages in a single server or cluster of servers. It may not be possible to index the whole Web using this approach. In a decentralized search engine, each webserver should index the hosted websites independently in a distributed manner over the whole Web. This distributed indexing should require low storage per node and network bandwidth.

- *Low network bandwidth consumption:* Search engine should search queries on the Web with low network overhead. It is possible to route the queries to the selected webservers utilizing optimal routing paths. Efficient selection of webservers and determination of routing paths can reduce the overall network bandwidth and improve response time.

- *Incremental retrieval:* Users expect to find their queried information within first few presented search results and do not care about the subsequent results if they found their queried information. A search engine can optimize the searching procedure using this searching behavior. A search engine can find the most relevant information by selecting a smaller set of more important results. It can find and present more search results if the users are not satisfied with the presented results. This approach significantly reduces network bandwidth.

- *Scalability:* A decentralized search engine should be scalable as the number of websites is increasing day by day. Scalability should not affect flexible searching, accuracy of search results, ranking, network bandwidth and search response time.

- *Robustness:* The search engine should be robust to webserver failure. If few webservers fail, the search engine should be able to route the queries to alternative webservers with low network overhead and provide search results with high accuracy.

1.2 Motivation

Distributed indexing and decentralized searching of the Web are very difficult to achieve given the bandwidth limitation and response time constraints. In addition to indexing and searching, a distributed web search engine should be able to rank the search results in a decentralized manner, which requires global knowledge about the hyper-link structure of the Web and keyword-document relevance. Predicting such global information based on local knowledge only is extremely challenging in any large scale distributed system.

Link-structure analysis and keyword relevance are two widely used webpage ranking techniques in both centralized and decentralized systems. Existing approaches for decentralized ranking can be classified into three categories: (a) only

use link-structure weight, (b) only use keyword relevance, and (c) use both link-structure and keyword relevance. Approaches that belong to category c are better than approaches in other categories. Some approaches belong to categories a and c compute link-structure weight without preserving hyper-link structure which leads to an inefficient weight computation. The reason is that the webpages don't get their actual weight update as in centralized weight computation. Other approaches from those two categories (a and c) preserve hyper-link structure using non-DHT (Distributed Hash Table) P2P overlay, which increases network overhead for weight update and computation. Non-DHT based approaches do not have global knowledge on the number of webpages given a particular keyword which leads to an approximate value of keyword relevance. Existing decentralized approaches do not use the concept of incremental retrieval.

In this thesis, we present *DEWS*, a decentralized web search engine, where webservers collaboratively index their hosted websites, route queries and rank the search results. *DEWS* provides flexible searching with high accuracy of ranking and search results with low network overhead. *DEWS* uses a DHT-based P2P overlay of webservers and preserves the hyper-link structure to compute link-structure weights and keyword relevances efficiently. *DEWS* retrieves information incrementally to reduce network overhead and response time. It is also robust and scalable to meet the challenges of the growing World Wide Web.

1.3 Contributions

The contributions of this thesis can be summarized as follows:

- We propose a novel technique for enabling the Web to index itself. In our approach, no external entity is required to crawl and index the Web, rather webservers can collaboratively create a distributed index of webpages and respond to user queries.

- Unlike existing approaches for keyword search and distributed ranking, DEWS supports approximate keyword matching and complete ranking of webpages in a distributed manner without incurring significant network or storage overheads.

- We propose a new route aggregation protocol that extends the original Plexus routing protocol by adaptively combining routing messages from different

sources. Each node forwards incoming messages to selective next hop nodes towards the targets of the incoming messages. This approach significantly reduces the number of routing messages in the network.

- We also propose a distributed incremental retrieval technique that allows a user to limit his/her search to a small number of nodes. If additional results are required a user can progressively query additional nodes. Proposed mechanism does not incur excessive network overhead and uses structured routing to forward query messages to well-defined sets of target nodes where the webpages matching the query keywords are indexed.

1.4 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2: Background and Related Work

This chapter is divided into three parts. First part presents a discussion on the factors considered for ranking search results and existing approaches for Web search. Second part provides a brief discussion on existing decentralized search engines, and compares them with *DEWS*. Finally, in the third part, we present some preliminaries for the discussions in the subsequent chapters.

Chapter 3: Framework of *DEWS*

Chapter 3 presents a layered architecture of *DEWS* followed by the details of each layer in a bottom-up manner. This chapter focuses on distributed indexing, query routing and ranking methodologies in *DEWS*.

Chapter 4 : Evaluation

Chapter 4 presents performance evaluation of *DEWS*. We define some performance metrics and present simulation results to assess efficiency of searching, ranking, routing, indexing, and robustness.

Chapter 5 : Conclusion and Future Research

Chapter 5 presents summary of contributions and concluding remarks with future research directions.

Chapter 2

Background and Related Work

2.1 Introduction

This chapter introduces the web search ranking factors and presents both centralized and decentralized ranking mechanisms. It also briefly discusses some existing distributed search engines and compares them with our work. Background knowledge on linear binary code, list decoding, Plexus, and other necessary concepts required in subsequent chapters are briefly presented.

This chapter is organized as follows. Section 2.2 presents the factors for ranking Web search results. Centralized ranking approaches are discussed in Section 2.3. Section 2.4 presents the decentralized ranking approaches. We discuss the existing distributed search engines and compare them with *DEWS* in Section 2.5. Finally, preliminaries needed for understanding subsequent chapters are presented in Section 2.6.

2.2 Web search and ranking factors

This section presents the methodology behind the Web search and the factors considered for ranking of search results. A search engine crawls and indexes the webpages hosted in the Internet all around the World. It maintains inverted indexes generated from the webpages that are used during the query resolution using a few search keywords. In general, inverted index contains the information on keywords and their relevance to particular webpages. The Google search engine uses more than 200 factors for ranking search results [6]. The ranking factors include weight computed using hyperlink-structure analysis, keyword relevance to the webpages,

age of webpages, frequency of webpage updates, amount of content change in webpages, popularity of websites, type of contents in the webpages, size of the websites, and domain name extension. Among the different ranking factors, following two factors are well-discussed in the literature:

- **Hyperlink structure**
  Link structure analysis is very popular for ranking search results where a particular resource (e.g., webpage, website or document) gets higher rank if it is authorized (e.g., linked or referenced) by many other resources. The key factors for computing link weights include authority-ship from other websites, validity of internal links, intra-site links, anchor text for outgoing links, validity of outgoing links, etc.

- **Keyword relevance**
  Each webpage is represented by a few keywords and inverted index contains their relevance to the particular webpages. Keyword relevance is measured using *tf* (term frequency), position of a keyword in a webpage (e.g., in URL, head, body, anchor) and *idf* (inverse document frequency) of the webpage. *tf* is defined as the number of times a keyword appears in a particular webpage. *idf* is used to measure whether the keyword is common or rare across all webpages in the Web. *idf* for a keyword k is defined as follows:

$$idf = log \frac{|W|}{1 + |\{w \in W : k \in w\}|} \tag{2.1}$$

  In the above equation, $|W|$ is the number of webpages in the Web and $|\{w \in W : k \in w\}|$ is the number of webpages where the keyword k appears.

2.3 Centralized ranking approaches

This section presents representative centralized approaches in the literature for ranking Web search results.

(a) **PageRank and variations**

(i) Original PageRank
The most popular and effective link structure analysis algorithm is PageR-

ank [38]. Google uses PageRank to compute the authority of each crawled webpage. PageRank of a particular webpage A is computed as follows:

$$PR(A) = (1-d) + d\sum_{i=1}^{i=n} \frac{PR(B_i)}{Links(B_i)} \tag{2.2}$$

Here, d is a damping factor, which is usually kept as 0.85, B_i is the i^{th} webpage that links to page A and Links(B_i) is the number of out-going links of page B_i.

(ii) Personalized PageRank

Original PageRank algorithm [38] does not consider user preferences (bookmarks or preferred pages) during the computation of PageRank. User preferences are considered in PageRank by introducing a preference vector (represents the probabilities of the preferred pages) in Equation 2.2. PageRank with user preferences can be computed as follows [26]:

$$PR(A) = (1-d)\sum_{k=1}^{k=m} P_k + d\sum_{i=1}^{i=n} \frac{PR(B_i)}{Links(B_i)} \tag{2.3}$$

Here, m is the number of preferred pages and P_k is the probability of surfing the k^{th} preferred page from any page (defined in the preference vector).

(iii) Topic-sensitive PageRank

PageRank [38] does not consider query contexts for ranking the query results. Topics of the queries are classified into several types (e.g., 16 categories from the Open Directory Project (ODP) [41]) in [25] to incorporate the query contexts in PageRank. PageRank vectors for each of the categories are computed off-line. During query processing, user queries are classified into the specific categories and PageRank vectors associated to those categories are applied. In [25], topic or category based PageRank is computed as follows:

$$PR(A^c) = (1-d)\sum_{k=1}^{k=m^c} P_k^c + d\sum_{i=1}^{i=n} \frac{PR(B_i^c)}{Links(B_i)} \tag{2.4}$$

Here, $PR(A^c)$ is the PageRank of page A as a category c, m^c is the number of predefined pages for category c, and P_k^c is the probability of k^{th} page in category c.

(iv) Other variations of PageRank

Weighted PageRank [37] groups the URLs into clusters and assign weights to the clusters, which is very similar to the topic sensistive PageRank. Original PageRank algorithm does not consider the possibility of browsing visited webpages using back button. BackRank ([14], [35]) modifies the original PageRank algorithm by adding the possibility of return to the earlier page by back button. Parallel PageRank approaches (such as [29], PETSc PageRank [22], and MIKElab PageRank [34]) compute PageRank in parallel to converge quickly. Approximate PageRank algorithms (such as BlockRank [27], the U-Model [16], and HostRank/DirRank [19]) use higher-level formations such as the inter-connection/linkage between hosts, domains, servers' network addresses or directories to compute approximate PageRank values fast.

(b) **HITS**

HITS (Hypertext Induced Topic Search) [28] is another well-known ranking algorithm based on link-structure analysis. HITS employs two scores for each page: hub score and authority score. The computation of these scores is also an iterative process similar to PageRank computation. An authority is a page with many in-links and a hub is a page with many out-links. The intuition is that a good authority is pointed by many good hubs and a good hub points to many good authorities. Given a broad query q, HITS finds a set of pages called 'root set' and computes another set of pages called 'base set' following the *in* and *out* links of the root pages. Pages with high authority and hub scores are selected first as the query results. HITS performs ranking considering query contexts. HITS requires more query resolution time than PageRank as HITS computes the root and base sets during the query evaluation phase.

(c) **Hilltop**

Hilltop [12] maintains a set of expert documents, which allow to provide query specific pages in search results. Expert documents are the subset of the crawled pages, which are topic specific and have links to many non-affiliated (e.g., from different domains) pages. Key phrases (title, header, anchor, etc. containing at least one URL) are extracted from the expert documents and maintained in an inverted index along with unique identifier, type (e.g., header, title, etc.), offset of the query keyword within the phrase, and URLs to match the query efficiently and compute the related pages. For a given query q, Hiltop looks up expert

documents and finds target pages from the expert documents. Hiltop assigns expert scores based on the number of query keywords each expert document contains and select k number of expert documents having high expert scores. From the selected expert documents, the Hiltop algorithm determines the target score for each URL in the selected expert documents and provides the high scored target pages as query result. Similar to HITS, Hilltop is slow compared to the standard PageRank algorithm as it determines the expert documents and target pages during the query evaluation phase. Hiltop may provide better query specific results compared to HITS and topic-sensitive PageRank. Hilltop may perform poorly if adequate expert documents are not available for a specific query.

(d) **SALSA**
SALSA [30] computes ranks of web pages combining the approaches of HITS [28] and PageRank [15]. For a given query, SALSA computes a set of pages using a search engine (such as Alta-Vista) similar to HITS, which is called *base set*. From the *base set*, another set of pages (*super set*) is identified following the links of the *base* pages. *super set* can be represented as a bipartite graph G whose two parts correspond to the hubs and the authorities, where an edge between hub r and authority s means that there is a hyper-link from r to s. SALSA employs two random walks for hubs and authorities, respectively where PageRank employs one random walk. SALSA ignores the intra-domain links. It provides query specific ranked results similar to HITS. As SALSA uses another search engine (e.g., AltaVista) for computing *base set*, the quality of results from AltaVista has a direct impact on the quality of ranking in SALSA.

2.4 Decentralized ranking approaches

This section presents existing decentralized approaches for ranking Web search results and compares them with our proposed approach *DEWS*.

(a) Sankaralingam *et al*. proposed a distributed PageRank algorithm in [43] for ranking (HTML) documents available in P2P networks. This approach works with both DHT and non-DHT based P2P networks. It is assumed that documents are pointed by other documents as webpages in the Web. At the beginning of PageRank computation, each peer assigns an initial PageRank score to the documents hosted by them and sends rank update messages to

the peers hosting out-linked documents. Suppose peer p_i hosts a set of documents D_i=$\{d_k\}$ which have out-links to other documents $\{d_{ij}^{out}\}$ hosted by the peers $\{p_{ij}^{out}\}$. p_i computes PageRank for each document in D_i and sends rank update messages to $\{p_{ij}^{out}\}$. When a peer receives an update message, it computes PageRank for the targeted documents. If new PageRank score for that document differs beyond a predefined threshold value from the old value, then update messages are sent to the peers containing out-linked documents. In this way, after hundreds of iterations, PageRank algorithm converges. When a new document is inserted in the network, it is assigned a random score and the hosting peer sends rank update messages to its out-linked documents. In this way, incremental PageRank computation is performed to avoid computation for the whole Web. In this approach, each peer caches addresses of the peers hosting the out-linked documents so that peers can send messages to them directly which helps to compute PageRank efficiently in a distributed manner. Our proposed approach, *DEWS*, computes PageRank similarly to this approach but differs from it as follows: (a) we apply a structured P2P network named 'Plexus'; (b) Plexus routing is modified and used for routing ranking and advertisement messages in *DEWS*; (c) *DEWS* provides flexible searching; (d) the concept of incremental retrieval is different, and (e) similar to this approach, *DEWS* caches other peers' addresses as 'soft-links'.

(b) Shi *et al*. proposed *Open System PageRank* in [44] based on structured P2P networks where each peer can communicate and view other peers' webpages. In this approach, webpages are divided into *pagegroup*s using hash code of the websites. If the system has n peers (*ranker*s) participating in the ranking, the whole Web is partitioned into n *pagegroup*s and assigned to the *ranker*s. Each *pagegroup* has internal links (IL) within the webpages belonging to that *pagegroup*. All the links (say EL_i) from webpages in a *pagegroup* (say PG_i) to the webpages in other *pagegroup*s (PG_i^{out}) are known as external links of PG_i. The computation of PageRank has two phases: a) each ranker computes PageRank for the *pagegroup*s assigned to it, b) each peer sends update messages to the peers responsible for the external linked *pagegroup*s. For example, peer P_i responsible for PG_i computes PageRank for PG_i using the links IL_i initially and sends update messages to the peers responsible for PG_i^{out} using EL_i. After a number of iterations, this algorithm converges. The computation of PageRank in this approach is similar to the computation in the original PageRank but

performed in a decentralized manner. Similar to this approach, we assume that webservers can communicate through server-to-server communication in a structured P2P overlay (specifically *Plexus*) and rank at the granularity level of website. However, the algorithm [44] does not mention on how the *pagegroup*s are assigned to peers and peers communicate with each other. In *DEWS*, websites are uniformly distributed over the network based on DHT. We use soft-links so that peers can communicate directly with other peers without overwhelming the network. *DEWS* provides flexible searching and incremental retrieval. The above approach only applies PageRank scores where *DEWS* uses both PageRank scores and keyword relevance.

(c) In Juxtaposed Approximate PageRank (*JXP*) [39], approximate PageRank is computed in a decentralized manner based on non-structured P2P networks. *JXP* does not use any specific webpage-to-peer matching technique and may assign a webpage to multiple peers. Each peer constructs a local graph based on the intra-links within the webpages assigned to them. An additional node (known as *world node*) is attached to the local graph of a peer to represent the knowledge of webpages that do not belong to them. The PageRank for the *world node* (say, PR_{wn}) is computed as PR_{wn} = 1-PR_{lg} where PR_{lg} is the summation of PageRank score of all the webpages in the local graph lg. A peer updates its *world node* when it meets with another peer. *JXP* computes PageRank in two phases: a) each peer constructs *local graph* including *world node* and initializes PageRank for all the webpages belonging to the graph, b) peers meet together and update PageRank of their webpages and *world node* by exchanging and merging their local graphs. *JXP* uses statistical synopses (light-weight approximation technique for comparing data between two peers with out exchanging their contents) to select promising peers to meet. After a several hundreds of meetings, each webpage gets an approximate PageRank score. This algorithm has a few problems: a) PageRank scores do not converge to the centralized PageRank scores due to the lack of global knowledge, b) correctness of the PageRank scores depend on the number of peer meetings and choice of peers, c) if the number of peers grows, it requires a large number of meetings and convergence time which is not a scalable solution, d) same webpage hosted by different peers may have different PageRank scores, e) use of non-structured overlay may lead to large network overhead for peer-to-peer communication. In contrast, we use structured overlay and soft-links which help

to compute PageRank score very close to the centrally computed PageRank but in a decentralized manner and compute keyword relevance to rank the search results. *DEWS* provides flexible searching and incremental retrieval which are not offered by *JXP*.

(d) Wang *et al*. [47] proposed a distributed ranking approach where webservers crawl and store only a portion of the Web. Links between webpages stored on different servers are discarded, which restricts each server to a partial view of the global link structure. A server computes PageRank (named *Local PageRank*) only for the webpages that are stored locally based on the partial link structure. Query results are ordered based on the pre-computed *Local PageRank* and *ServerRank*. *ServerRank* of a webserver is computed as the maximum or the summation of the *Local PageRank* in that server. Computation of *ServerRank* in this way may result in assigning higher ranks to the irrelevant pages just because they are stored in the highly ranked servers. In contrast to this algorithm, we compute PageRank on the whole Web and keyword relevance in decentralized manners. We also provide flexible searching and incremental retrieval.

(e) SiteRank [49] proposed a decentralized system architecture [48] to compute ranks of Webpages. This approach computes PageRank in three steps: a) computation of *siterank*, b) computation of local rankings of webpages, c) combination of the ranking scores using the algebra specified in [9]. In this approach, a *sitegraph* is defined using the collection of websites and their internal links. The assumptions regarding the computation of *sitegraph* are as follows: a) the size of the whole Web is only of the magnitude of a dozen of millions, b) it is possible to compute *siterank* using the *sitegraph* even in a low-end PC, c) Web does not change dramatically so that it is possible to exchange the *siterank* vector among the webservers. These assumptions make this approach inappropriate towards a search engine for gigantic Web because computing global *SiteRank* in a centralized manner will not scale with current size of the Web. In contrast, we compute the PageRank at the granularity level of websites in a decentralized manner. We also use keyword relevance and rank the websites using both the PageRank and keyword relevance.

2.5 Web search engines

This section presents existing decentralized Web search engines and compares them with *DEWS*.

(a) **MINERVA**

MINERVA [11] is a DHT-based (Chord [45]) decentralized Web search engine. In MINERVA, every peer is autonomous and maintains a local index. Each peer acts like a crawler and posts (using DHT) a small amount of metadata corresponding to the representative keywords of documents. The peer, indexing a particular term, maintains a *PeerList* of all postings for that term from across the network. Posts contain contact information about the peer who posted this summary together with statistics to calculate ranking score for a term. If a query is initiated in a peer, it retrieves the *PeerList* for all the query terms by DHT lookups. It selects and contacts with top-k peers from each *PeerList* using a distributed top-k algorithm [36] so that all the selected peers can be queried in parallel. Global document frequency *gdf* is computed using the Hash sketch technique [20]. In that approach, every peer includes a hash sketch representing its index list for the respective term when publishing its (term-specific) Post, so that a directory peer can compute an estimate of *gdf* for the terms it is responsible for (as the hash sketch synopses representing the index lists of all peers for a particular term are all sent to the same directory peer). The querying peer collects the *gdf*s as piggybacked information when retrieving the *PeerLists* from the directory peers and includes *gdf* values when sending the query to peers selected in the query routing phase. These remote peers can use the *gdf* estimates on-the-fly (as weights during index scans) to compute their local query results, to produce globally comparable scores. The differences between MINERVA and *DEWS* are as follows: a) *DEWS* computes and uses PageRank and BM25 scores whereas MINERVA only employs document frequencies, b) *DEWS* provides flexible searching which may not be possible in the Chord overlay in MINERVA.

(b) **ODISSEA**

ODISSEA (Open DIStributed Search Engine Architecture) [46] was a proposal for a P2P search engine for different applications including searching P2P networks, large intra-net environment and the Web. This system proposed two tiers: *lower layer* and *upper layer*. *lower layer* maintains overlay and index. It

also serves queries and performs ranking on the documents. *upper layer* manages the documents (e.g., insert, delete, update) and designs optimized query execution plan. This proposal includes PageRank, term frequencies and other approaches for ranking documents, but there is no specific algorithm provided for them. Authors proposed a Bloom filter based protocol for optimizing query execution plans. In contrast with ODISSEA, we present a complete Web search engine with specific implementation.

(c) **CHORA**

CHORA [24] is not a standalone search engine rather it enhances the current centralized search engines to incorporate the users' browsing history into search results. The framework of CHORA consists of two search components: a) a traditional search engine and b) a desktop search engine. When users search queries, both the central search engine and CHORA (with the help of desktop search engine) compute search results which are re-ranked and presented to the users. During the user registration with CHORA, a summary including URL of the computer, location, bandwidth, and a set of related keywords on that computer is computed and stored using openDHT [42]. CHORA aggregates users browsing history by computing click graphs. Click graph organizes user webpages based on the connectivity implied by their clicks and summary statistics describing their interaction with each page. Webpages are ranked using the amount of time spent on the webpages by a particular user. The query processing in CHORA involves two steps: a) selecting peers using DHT, b) routing queries to the selected peers and retrieving webpages using the desktop search engine. The motivation behind CHORA is to reflect the users browsing history in the search results returned by the traditional search engines. In contrast, our motivation behind *DEWS* is to develop a standalone decentralized Web search engine.

(d) **COOPER**

The motivation behind COOPER [50] is similar to CHORA. It works with centralized search engines and incorporates users searching experiences on the search results in a Peer-to-Peer fashion . Instead of using PageRank, COOPER proposes *PeerRank* with an assumption that human recommendations constitute a better measure of relevance than link structure weight. It has four components: a) User agent which interact with users through search interface, b) Web-searcher agent which performs the users' searching using traditional search

engines and builds the repository of users' searching experience, c) Collaborator agent which performs the users' real-time collaborative searching, and d) Manager agent that coordinates and manages other types of agents. When a user submits a query, Web-searcher agent fetches and gives the requested webpages to the User agent. At the same time, Collaborator agent shares the user's new result to the whole network. The major disadvantage of this approach towards a scalable Web search engine is that it floods the queries in its Gnutella [1] network which requires a large network overhead for query processing.

(e) **AlvisPeers**

AlvisPeers [33] is a full-text P2P retrieval engine. It uses P-Grid [8] as the underlying network. The framework of this system has three layers: a) DHT layer which stores global index, b) *HDK* (Highly Discriminative Keys) layer for building the key vocabulary and corresponding posting lists, and mapping queries to keys during retrieval, and c) Ranking layer that implements distributed document ranking. It is assumed that each peer runs a Web service to accept queries and documents from remote hosts. *HDK* is used for minimizing the number of keywords to be indexed in DHT. During indexing, keywords are categorized into two groups based on it document frequency (DF): non-discriminative (if its DF is greater than a pre-defined threshold DF_{max}) and discriminative (if it's DF is less than DF_{max}). A keyword is discriminative if it is a strong candidate for representing a document. It uses tf(term frequency)*idf(inverse document frequency) of the queried keywords to measure the rank of the documents. There are few differences between AlvisPeers and *DEWS* as follows: a) we use both PageRank and BM25 scores (as keyword relevance), but AlvisPeers only uses tf*idf for document ranking, b) we provide searching results with high accuracy even in presence of misspelled or partially specified queried keywords where AlvisPeers only supports full-text search, c) we provide a mechanism for distributed websites indexing which is not present in AlvisPeers, d) incremental retrieval is not supported in AlvisPeers.

(f) **YACY**

YACY [7] is a fully decentralized open source Web search engine based on a P2P network. Users become *participating peers* by installing the YACY software in their machine. It employs thousands of crawlers to index webpages and stores in the network in multiple peers (*replica*) using DHT. User requested information is not censored or blocked as there is no central authority which is also our

motivation behind *DEWS*. The ranking of search results are performed on the users' machines based on a set of users' preferences. YACY does not have a ranking algorithm like PageRank so that most relevant information may not be presented to the users [4]. In contrast with YACY, all participating webservers in *DEWS* index their hosted websites only. Another major difference is that *DEWS* computes PageRank in a decentralized manner to present most relevant information to the users which is not done in YACY.

(g) **FAROO**

FAROO [5] is a decentralized search engine based on a P2P network. When a user browses a webpage, FAROO indexes the webpage instantly using DHT. If the number of users is small and the users do not browse webpages a lot, search results of FAROO will be poor. FAROO does not use PageRank or other link-structure weight, which makes it difficult to present the most relevant information to the users. In contrast to FAROO, all webservers index their hosted websites in *DEWS*. *DEWS* also computes PageRank and keyword relevance to present most relevant information to the users.

2.6 Preliminaries

In this section, we provide preliminaries on coding theory specially Reed-Muller code, list decoding, Bloom filter, edit distance, BM25 and phonetic algorithm.

2.6.1 Linear binary code

A linear binary code $\mathcal{C}$ can be represented as $< n, k, d >$ where n is the number of bits (0 or 1) in $\mathcal{C}$, k is the dimension of the generator matrix of $\mathcal{C}$, and d is the minimum Hamming distance between any two codewords in $\mathcal{C}$ [17]. All the codewords of a particular linear binary code can be represented by a minimal set of codewords, which is known as generator matrix. A generator matrix G_C of a linear binary code $\mathcal{C}$ has k rows. XORing any number of rows from G_C produces another codeword and in this way it is possible to generate all the 2^k codewords of $\mathcal{C}$ using G_C.

$$G_C = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix}$$

2.6.2 Reed-Muller Code

Reed-Muller code is a linear binary code. The r^{th} order Reed-Muller code is denoted by RM(r,m), which is a vector subspace of length $n = 2^m$ over F_2^n, for some positive integers r and m. Minimum distance d, between any pair of codewords in a RM(r,m) code is 2^{m-r}. Number of codewords in the code is 2^k, where $k = \sum_{i=0}^{r} {}^mC_i$ is the dimension of the code. For example, minimum distance for RM(2,6) code is 16, dimension of the code is 22, number of bits in each codeword is 64 and number of codewords is 2^{22}.

The generator matrix of Reed-Muller code can be constructed in the following way. The number of bits in each row of the matrix of RM(r,m) code is 2^m. We can consider a bit vector (X_0) of length 2^m containing 1 in all the bits, which is the first row of the matrix. The second row (X_1) can be defined as 1 in first 2^{m-1} bits and 0 in the last 2^m bits as depicted in Figure 2.1. In this way, rows X_0 to X_r are computed. Then m_{C_r} rows are computed using the rows X_0 to X_r. For example, generator matrix for RM(2,3) contains the rows X_0X_1, X_0X_2, X_1X_2 where 'X_1X_2' is the dot]

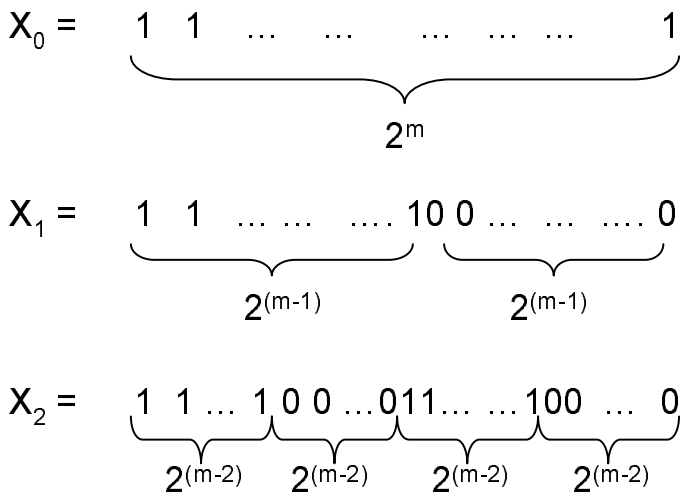


Figure 2.1: Construction of generator matrix

$$G(2,3) = ss \begin{bmatrix} X_0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ X_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ X_2 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ X_0X_1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ X_0X_2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ X_1X_2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

2.6.3 List Decoding

Let C be a linear binary code $< n, k, d >$ and x be a binary pattern of length n, then a list decoding of C provide a set of codewords $X = \{X_1, X_2, \ldots, X_m\}$ where $X_i \in C$ and Hamming distance from x to each X_i is at most ϵ as follows:

$$X(x) = \{X_i | X_i \in \mathcal{C} \wedge d(X_i, x) \leq \epsilon\}$$

In literature, there are few sophisticated list decoding algorithms including [18], [23], [21], and [40]. Algorithm 1 presents a straight-forward list decoding algorithm, which computes a list of codewords upon receiving a binary pattern where all the codewords are within a pre-specified Hamming distance ϵ (line 10) and the number of codewords in the list is bounded by a pre-specified number γ (line 13). The complexity of Algorithm 1 is $O(2^k)$ where k is the dimension of the code. Although its complexity is high, it offers simplicity of list decoding. For this reason, we employ this algorithm in our simulation.

List decoding plays an important role in approximate matching. A set of codewords computed from a binary pattern are numerically close to each other. We determine the target nodes using the list decoded codewords during keyword advertisement and searching, which increases the chance of finding common nodes even if the search keyword is partial or approximate to the advertisement keyword. The concept of keyword matching using list decoding is discussed in Section 2.6.4. Figure 4.2(b) shows the effectiveness of list decoding technique.

2.6.4 Plexus

In Plexus [10], keywords are mapped to patterns (or bit-vectors) and a Hamming distance based routing technique derived from the theory of *Linear Binary Codes* is used. The keyword to pattern mapping process retains the notion of similarity

Algorithm 1 $ListDecode(\rho, \epsilon, \gamma)$

1: Inputs:
   ρ: Binary pattern need to be decoded
   ϵ: List decoding radius
   γ: Maximum list decoding size
2: Internals:
   k: Dimension of the linear code
   G: Generator matrix of the linear code
   g_i: i^{th} row of the Generator matrix G
   $\delta(a, b)$: Numeric distance between a and b
   $\psi(\Omega)$: Number of elements in a list Ω
   $\Upsilon(\zeta_{i-1}, i)$: Compute grey code corresponding to i
   $\Pi(g_{i-1}, i)$: Compute the bit position where ζ_{i-1} and ζ_i differ
   Ω: List of codewords
3: $c_i \leftarrow null$
4: $\Omega \leftarrow null$
5: $\zeta_{-1} \leftarrow 0$
6: **for** i=0 to 2^k-1 **do**
7: $\kappa \leftarrow \Pi(\zeta_{i-1}, i)$
8: $\zeta_i \leftarrow \Upsilon(\zeta_{i-1}, \kappa)$
9: $c_i \leftarrow c_i \ \oplus \ g_\kappa$
10: **if** $\delta(c_i, \rho) \leq \epsilon$ **then**
11: $\Omega \leftarrow c_i$
12: **end if**
13: **if** $\psi(\Omega) \geq \gamma$ **then**
14: **return** Ω
15: **end if**
16: **end for**

between keywords, while Hamming distance based routing delivers deterministic results and efficient bandwidth usage.

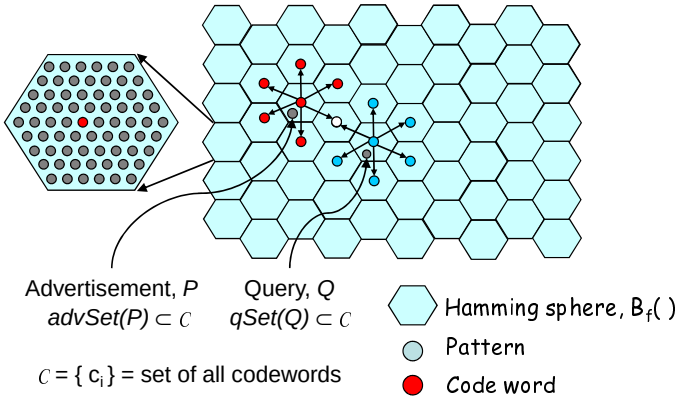


Figure 2.2: Core concepts in Plexus

As explained in Figure 2.2, a linear binary code $\mathcal{C}$ (for example, Reed-Muller code) partitions the entire pattern space $\mathbb{F}_2^n$ into Hamming spheres, represented by hexagons in the figure. A codeword ($c_i \in \mathcal{C}$) is selected as the unique representative for all the patterns within its Hamming sphere. An advertised pattern P is list decoded to a set of codewords $\mathscr{A}(P) = B_s(P) = \{c_i | c_i \in \mathcal{C} \wedge \delta(c_i, P) \leq s)\}$. Similarly, a query pattern, say Q, is list decoded to $\mathcal{Q}(Q) = B_t(Q) = \{Y | Y \in \mathcal{C} \wedge \delta(Y, Q) \leq t)\}$. It has been shown in [10] that there will be at least one codeword in $\mathscr{A}(P) \cap \mathcal{Q}(Q)$ if the Hamming distance between P and Q, $d(P, Q) \leq s + t - 2f$, where f is the covering radius of $\mathcal{C}$. In Plexus network, each peer is assigned for one or more codewords. The pattern P is advertised to the peers assigned for the codewords in $\mathscr{A}(P)$ and the pattern Q is queried to the peers assigned for the codewords $\mathcal{Q}(Q)$. Thus, the peers assigned for the codewords in $\mathscr{A}(P) \cap \mathcal{Q}(Q)$ serve the query pattern Q.

In Plexus, each peer maintains $k + 1$ routing entries in its routing table, where k is the dimension of $\mathcal{C}$. These $k + 1$ routing entries contain links to the peers responsible for the codewords $X_1, X_2, \ldots, X_{k+1}$ computed as follows ($\oplus$ refers to the bitwise XOR operation):

$$X_i = \begin{cases} X \oplus g_i & 1 \leq i \leq k \\ X \oplus g_1 \oplus g_2 \oplus \ldots \oplus g_k & i = k + 1 \end{cases} \tag{2.5}$$

Using these routing links, Plexus route a message from any source peer to any target peer in less than or equal to k/2 routing hops. The routing mechanism in Plexus is based on the linear code and generator matrix. Suppose peer X (*i.e.*, ,

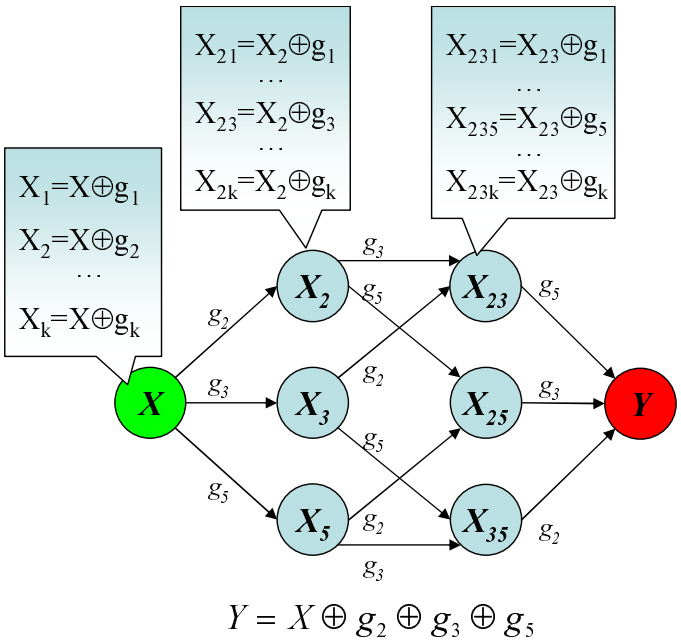


Figure 2.3: Plexus routing

the peer responsible for codeword X) wants to route a message to peer Y which is shown in Figure 2.3. According to the properties of linear codes, any codeword (say Y) can be obtained by XORing some combination of generator matrix rows (g_i) with any given codeword (say X). For example, codeword Y can be expressed as $Y = X \oplus g_2 \oplus g_3 \oplus g_5$. Since peer X has routing links to peers $X_2 = X \oplus g_2$, $X_3 = X \oplus g_3$ and $X_5 = X \oplus g_5$, peer X can forward the message to any of these three peers in one hop. Suppose, peer X forwards the message to peer X_2. Similarly, peer X_2 can route the message to any of the peers $X_{23} = X_2 \oplus g_3$ or $X_{35} = X_2 \oplus g_5$ in one hop. Suppose, peer X_2 forwards the message to peer X_{23}. Finally, peer X_{23} can route the message to peer Y since peer X_{23} will have a routing link for $Y = X_{235} = X_{23} \oplus g_5$.

2.6.5 Bloom Filter

A Bloom filter [13] is a space-efficient probabilistic data structure used to represent a set. Bloom filters support set membership test operations with a small probability of false (erroneous) positives. An empty Bloom filter is a bit array of m bits, all set to 0. There must also be k different hash functions defined, each of which maps or hashes some set element to one of the m array positions with a uniform random distribution.

Figure 2.4 presents a Bloom filter which represent a set {a, b, c} where m=16

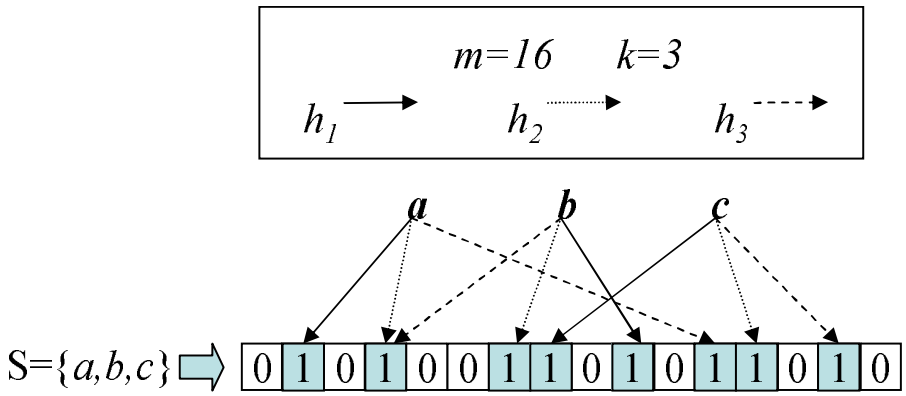


Figure 2.4: Bloom Filter

and k=3. To add an element, all the k hash functions are used to hash on it to get k array positions which are set to 1. To test membership of an element, all the k hash functions are used on it. If any of the bits on the resultant positions of the array are 0, the element is not in the set. If all are 1, then either the element is in the set, or the bits have been set to 1 during the insertion of other elements, which gives a false positive result.

2.6.6 Double Metaphone Encoding

The Double Metaphone encoding algorithm attempts to detect phonetic ('sounds-alike') relationships between English words. Double Metaphone works by producing one or possibly two phonetic keys from a given word. For example, the string 'Jhon Abraham' produces two phonetic words-JNPR and ANPR. The primary Double Metaphone key represents the American pronunciation of the source word. All words have a primary Double Metaphone key. The secondary Double Metaphone key represents an alternate, national pronunciation. For example, many Polish surnames are 'Americanized', yielding two possible pronunciations, the original Polish, and the American. For this reason, Double Metaphone computes secondary keys for some words. The vast majority (roughly, 90%) of words will not yield a secondary key, but when a secondary key is computed, it can be pivotal in matching the word. To compare two words for phonetic similarity, one computes their respective Double Metaphone keys, and then compares each of the following combination:

$$\begin{cases} \text{Primary key (word 1)}, & \text{Primary key (word 2)}; \\ \text{Primary key (word 1)}, & \text{Secondary key (word 2)}; \\ \text{Secondary key (word 1)}, & \text{Primary key (word 2)}; \\ \text{Secondary key (word 1)}, & \text{Secondary key (word 2)}. \end{cases}$$

Depending upon which of the above comparisons match, matching strength is computed. If the first comparison matches, the two words have a strong phonetic similarity. If the second or third comparison matches, the two words have a medium phonetic similarity. If the fourth comparison matches, the two words have a minimal phonetic similarity. Depending upon the particular application requirements, one or more matching levels may be excluded from the matching results.

2.6.7 Edit Distance

Edit distance is a metric for measuring the level of differences between two strings. The edit distance (also known as *Levenshtein distance*) between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation may be an insertion, deletion, or substitution of a single character [2]. For example, edit distance between 'kitten' and 'sitting' is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

$kitten - sitten$ (substitution of 's' for 'k');
$sitten - sittin$ (substitution of 'i' for 'e');
$sittin - sitting$ (insert 'g' at the end).

2.6.8 BM25 weighting scheme

BM25 [3] is a probabilistic weighting scheme to measure weight of a particular document using some statistics of the documents including frequency of the document in the collection, frequency of the keywords/terms in that document, number of relevant documents. Search engines use BM25 to rank a set of documents based on the search keywords appearing in each document.

Given a query Q containing keywords $q_1, \ldots, q_n$, the BM25 score of a document D is:

$$BM25(D,Q) = \sum_{i=1}^{n} IDF(q_i).\frac{f(q_i,D).(k_1+1)}{f(q_i,D)+k_1.(1-b+b.\frac{\|D\|}{avgdl})}$$

$IDF(q_i)$ = Inverse Document Frequency computed as $log\frac{N-n(q_i)+0.5}{n(q_i)+0.5}$
N = total number of documents in the collection
$n(q_i)$ = number of documents containing q_i
$f(q_i, D)$ = q_i's term frequency in the document D
$\|D\|$ = length of the document D in words
$avgdl$ = average document length in the collection
k_1 = constant $\in$ [1.2,2.0]
b = constant 0.75

We compute BM25 scores for the search keywords with the resultant websites to measure keyword relevance and rank the websites to present the most relevant websites in the first few results in *DEWS*.

2.6.9 *n*-gram

n-gram is a subsequence of *n* items from a given sequence. The items in question can be phonemes, syllables, letters, words or base pairs according to the application. An *n*-gram of size 1 is referred to as a 'unigram'; size 2 is a 'bigram' (or, less commonly, a 'digram'); size 3 is a 'trigram'; and size 4 or more is simply called an 'n-gram'. For example, the sentence 'the quick red fox jumps over the lazy brown dog' has the following character level tri-grams: 'the', 'qui', 'uic', 'ick', 'red', 'fox', 'jum', 'ump', 'mps', 'ove', 'ver', 'the', 'laz', 'azy', 'bro', 'row', 'own', 'dog'. We use 1-gram to match partially specified search keywords with the advertised keywords.

2.7 Summary

This chapter presented an overview of Web ranking mechanisms and background knowledge towards the context of subsequent chapters. PageRank, a link-structure analysis technique, is the most used technique in existing centralized and decentralized ranking algorithms. On the other hand, keyword relevance is popular for information retrieval purposes. We found that some of the existing decentralized techniques use both PageRank and keyword relevance together to rank their Web search results. We also found that existing decentralized approaches either compute the approximate values of PageRank due to the lack of global knowledge on hyper-link structure or utilize non-structured overlay which requires large volume of ranking messages for the algorithm to converge. We also found that decentralized techniques using non-DHT based overlay compute keyword relevance with a partial

knowledge of the Web. Few distributed Web search engines were briefly discussed and compared with our proposed search engine, *DEWS*. We have also presented some background on linear binary codes, Reed-Muller code, list decoding, Bloom filter, phonetic encoding, n-grams, BM25 to provide a context for the chapters to follow. We used Plexus, a structured P2P overlay, in *DEWS*. The core concept of Plexus and its routing mechanism were briefly presented in this chapter, as well.

Chapter 3

Framework of *DEWS*

3.1 Introduction

In this chapter we introduce *DEWS* as a solution to the problem of searching the Web in a distributed manner. The novelty of the proposed approach lies in the implementation of Google's PageRank in a distributed manner on a structured P2P overlay. It also computes BM25 scores in a distributed manner and allows approximate search with incremental retrieval. A structured overlay allows computation of PageRank similar to the centralized computation of PageRank, preserving the hyper-links structure of the websites. *DEWS* uses phonetic encoding and n-grams for approximate search and retrieves search results incrementally using the list decoding feature of the underlying linear binary code. *DEWS* ranks the Web search results efficiently using PageRank and BM25 scores.

The rest of this chapter is organized as follows. Architecture of *DEWS* is presented in Section 3.2 in a bottom-up manner. We present the Plexus overlay and modified Plexus routing algorithm in Section 3.3. Section 3.4 presents the mapping of hyperlink structure to the Plexus overlay and advertisement of distributed inverted index. Website advertisement, distributed ranking and retrieval are discussed in Section 3.5. We present the technique for incremental retrieval in Section 3.6.2. Finally, we summarize the concepts and findings of this chapter in Section 3.7.

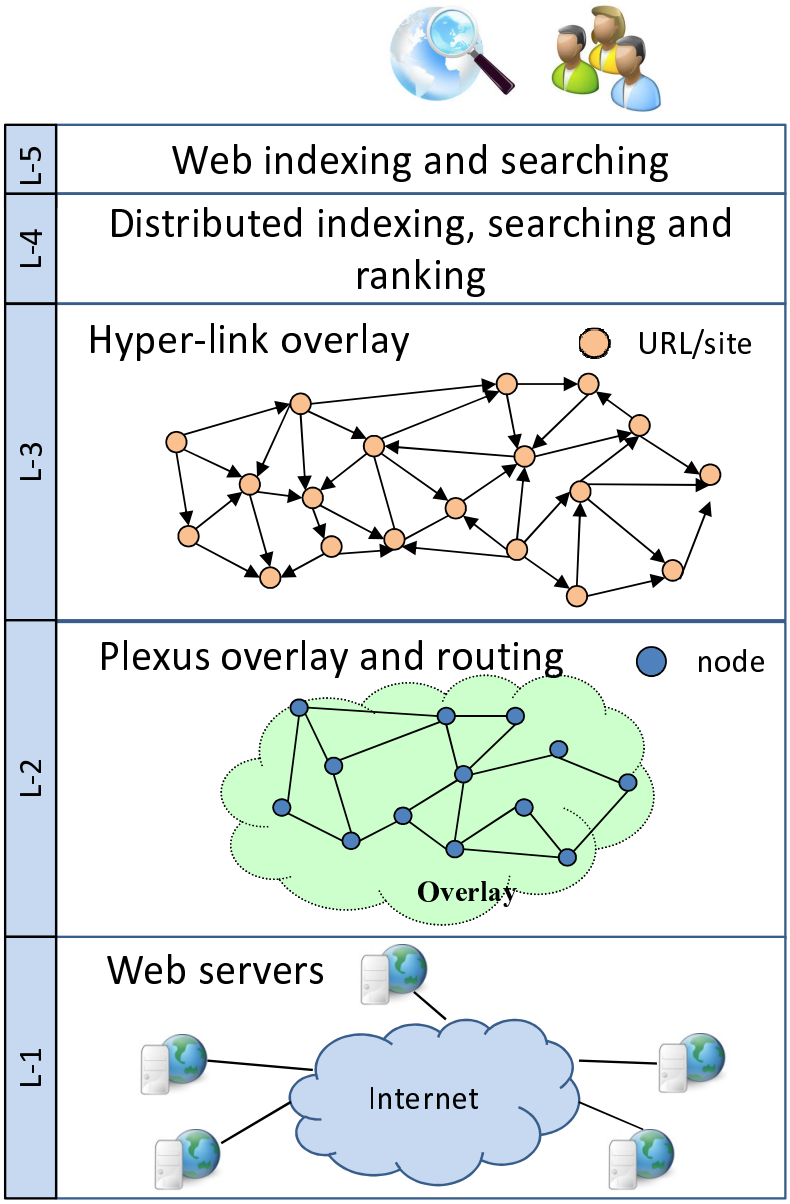


Figure 3.1: Architecture of *DEWS*

3.2 System architecture

The architecture of *DEWS* is composed of five different conceptual layers as shown in Figure 3.1. The higher layers are dependent on the functionalities provided by the lower layers. The five conceptual layers are as follows:

- **Web servers (Layer L_1)**
  We assume that web servers distributed all over the world connected by Internet collaborate on distributed indexing and searching the Web. Instead of crawling and indexing by a single server or a cluster of servers, webservers index their hosted websites collaboratively using server-to-server communication. In this approach, it is possible to index the whole Web. These webservers collaboratively resolve the user queries and rank the search results to present most relevant websites to the users.

- **Plexus overlay and routing (Layer L_2)**
  We assume that web servers are organized into a structured overlay network. Distributed Hash Table (DHT) based solutions have been proven to be efficient in information lookup (e.g., in $O(log\ n)$ hops) in very large networks. Hence we opt to use a DHT mechanism for indexing and search. In addition to efficient lookup, we need to perform approximate matching between query keywords and webpage keywords. In general DHT mechanisms offer exact matching and do not support approximate matching. We have chosen Plexus [10] to organize the webservers for the following reasons:

  - Plexus provides a DHT-based structured overlay which can preserve the hyperlink structure on top of it to compute the PageRank and keyword relevance efficiently.
  - Plexus has an efficient multicast routing protocol which requires reduced network bandwidth for routing to multiple peers. The maximum number of hops required to route a message from a source to a destination is bounded by $k/2$ where k is the dimension of the used linear binary code (for example, k is 22 for RM(2,6) code).
  - In Plexus, patterns having less Hamming distances are mapped to the nearby (or same) peers which allows efficient approximate matching between the advertised and query keywords.

- List decoding of the linear binary code used in Plexus provides an efficient incremental search mechanism.
- Plexus is robust to the peer failures.
- Plexus has a scalable architecture which can meet the challenge of exponential growth rate of websites.

- **Hyper-link overlay (Layer L_3)**
  We preserve the hyper-links among the websites on the Plexus overlay in a structured manner, which forms an overlay of hyper-links. Suppose two websites u_i and u_j have links between them. In the hyperlink overlay, there should be a link between the two nodes representing u_i and u_j. We index websites in such a way that hyperlink overlay is preserved on the Plexus overlay. In PageRank algorithm, one node hosting u_i has to send its weights periodically to the node hosting u_j if there is a link from u_i to u_j. For this reason we preserve hyperlink overlay to compute PageRank efficiently. This layer is also responsible for creating *distributed inverted indexes*. Techniques for hyperlink overlay preservation and *distributed inverted index* construction are presented in Section 3.4.

- **Distributed indexing, searching and ranking (Layer L_4)**
  This layer provides all the functionalities related to website indexing, computation of weights and searching in distributed manner. Websites are indexed using their representative keywords on the Plexus overlay preserving the hyperlink structure. We have used two types of metrics for ranking search results- PageRank weight of each website and keyword relevance to websites. This layer computes PageRank in a distributed manner utilizing the hyperlink overlay preserved on Plexus overlay. A distributed searching technique is also provided by this layer. During searching, websites are retrieved based on their relevance to the query keywords. Query keyword relevance to the websites are computed in a distributed manner. Mechanisms for indexing, ranking and searching in this layer are explained in Section 3.5.

- **Web indexing and searching (Layer L_5)**
  This layer has two components: search interface and website indexer. Webservers implementing this layer index their hosted websites. Webservers first extract the representative keywords from websites and index them using the functionalities offered by the lower layers. The search interface allows users

to search using keywords and discovers the most relevant websites. If users are not satisfied with the presented results, search interface has the option to retrieve more relevant websites. This approach is known as incremental retrieval. Details on indexing and searching techniques used in this layer are discussed in Section 3.6.

3.3 Plexus overlay and routing

In this section, we describe how the webservers are organized into Plexus overlay and present the modified Plexus routing mechanism.

3.3.1 Plexus overlay

We assume that webservers are organized into Plexus overlay based on the second order Reed-Muller code RM(r,m) (explained in Section 2.6.2). The maximum number of webservers that can be allowed in the Plexus network is $2^{1+\binom{m}{1}+\binom{m}{2}}$ using $RM(2, m)$ codes. For example, we have used $RM(2, 6)$ for simulation which can support about four million (2^{22}) nodes in the overlay. We can extend the network size by incorporating higher order Reed-Muller codes. In a network with maximum allowed webservers, each webserver is assigned and responsible for an unique codeword of RM(r,m) code. If the number of codewords is much higher than the number of connected webservers in the overlay, multiple codewords are mapped to each webserver as follows:

RM(r,m) code has exactly 2^k codewords, where k is the dimension of the code and the generator matrix G of the code has k rows. Now, we define a k-bit ID to identify the 2^k codewords in RM(r, m). The i^{th} bit of the ID for a codeword will be 1 if the i^{th} row of G (*i.e.*, g_i) is required to construct that codeword. We use this ID to partition the codewords into a logical binary tree of height at most k. At the i^{th} level of the tree, partitioning is based on whether the i^{th} row of G (*i.e.*, g_i) is used for constructing the codeword.

Figure 3.2 shows an example of the above mentioned partitioning process along with the routing table entries for webserver X. Each webserver is assigned a leaf node in this tree and is responsible for all the codewords having that particular combination of g_is. For example, webserver X is responsible for all the codewords including g_1, g_3 and excluding g_2, g_4 from their construction. This concept is implied by the prefix $g_1\bar{g}_2g_3\bar{g}_4$, where $\bar{g}_i$ indicates the absence of g_i in the construction of a

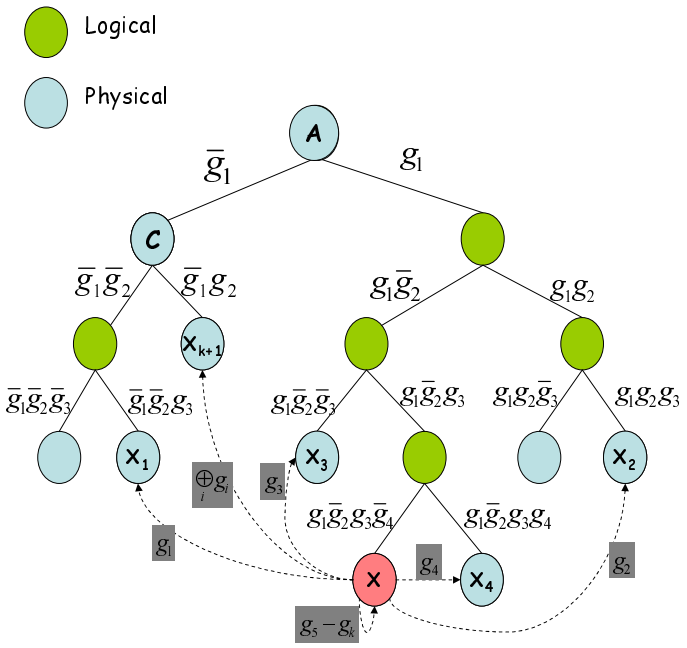


Figure 3.2: Mapping codewords to peers

codeword. The i^{th} entry in the routing table of webserver X points to the webserver responsible for the codeword $X \oplus g_i$ according to the partition tree as presented in Figure 3.2. A new webserver is organized in the Plexus overlay as the peer joining process in Plexus [10]. Although the rate of webserver failure is far less than that of peer failure in a P2P network, DEWS adopts the mechanism for handling peer failure as specified in Plexus.

3.3.2 Modified Plexus routing

We extend the Plexus multicast routing protocol by message aggregation. Our extension can be explained by the analogy of an airport. Each airport works as a hub. Transit passengers from different sources gather at an airport and depart on different outgoing flights matching their destinations. Similarly, we use each Plexus node as routing hubs. Default routing mechanism in Plexus is muticasting, since a few nodes have to be checked to allow approximate matching. As a result, each message arriving a node contains a number of target codewords.

Algorithm 2 presents the aggregate routing mechanism in DEWS. We expect each node to continuously receive messages, since Web queries from around the globe will be submitted and processed by the system. Instead of instantly forwarding the incoming messages, each node accumulates incoming messages in a message queue ($msgQ$) for very small period of time, around one second. Target codeword lists

Algorithm 2 *AggregateRouting*

Inputs:

$msgQ$: $\{< pl, \mathcal{Y} >\}$, where pl is message payload
and $\mathcal{Y}$ is target list for pl.

Internals:

k: Dimension of the linear code $RM(2, m)$

$X_1, \ldots, X_{k+1}$: $(k+1)$ neighbors of X $\{$*Eqn. (2.5)*$\}$

$\mathcal{Y}_m \leftarrow \bigcup_{m \in msgQ} m.\mathcal{Y}$

$\{$*find suitability of each neighbor as next hop*$\}$

$\mathscr{R} \leftarrow \{\mathcal{T}_1, \ldots, \mathcal{T}_{k+1} |\ \mathcal{T}_i \subseteq \mathcal{Y} \wedge$
$Y \in \mathcal{T}_i \implies X_i$ is on path$X \rightsquigarrow Y\}$

while $\mathcal{Y}_m$ not empty **do**

  $\mathcal{O} \leftarrow \phi$

  find s such that $\forall \mathcal{T}_i \in \mathscr{R}, |\mathcal{T}_s| \geq |\mathcal{T}_i|$

  for all $m \in msgQ$ **do**

    if $m.\mathcal{Y} \cap \mathcal{T}_s \neq \phi$ **then**

      $out \leftarrow \mathcal{O} \cup \{< m.pl, m.\mathcal{Y} \cap \mathcal{T}_s >\}$

      $m.\mathcal{Y} \leftarrow m.\mathcal{Y} / \mathcal{T}_s$

    end if

  end for

  $\mathscr{R} \leftarrow \mathscr{R} - \{\mathscr{T}_s\}$

  $\mathcal{Y} \leftarrow \mathcal{Y} - \mathcal{T}_s$

  send $\mathcal{O}$ to node X_s

end while

$(m.\mathcal{Y})$ in the incoming messages are combined to a master target list $\mathcal{Y}_m$. Then Plexus routing is applied to select the next hop neighbors and the targets in $\mathcal{Y}_m$ are distributed over the selected neighbors. Since, index advertisement and query messages have small size, many of these messages can be packed in a single message and sent to appropriate neighbors. This approach significantly reduces the number of messages in the network.

3.4 Hyper-link overlay and distributed index

This section presents construction of Hyper-link overlay and mapping to Plexus overlay. Mechanism of creating and storing inverted index in the Plexus overlay is also discussed in this section.

3.4.1 Hyper-link overlay

About 90% hyperlinks in the Web are intra-domain [49]. Topics and ideas in the webpages of a particular website are almost similar or correlated and it is not reasonable to utilize the authority-ship of web documents at the level of single pages; besides a website is usually reorganized and managed periodically without significant changes in semantics and outgoing hyper-links to the rest of the Web [49]. The number of websites in the Web about one hundredth of the number of webpages. Considering these facts we perform link structure analysis at the granularity level of websites. For the rest of this thesis, we use "URL" to refer to the root URL of a website. The links between two websites are the aggregation of the links between the web pages in the websites.

Algorithms for computing URL weights based on the hyperlink structure are iterative and require many iterations to converge. In each iteration URL weights are updated and the new weights are propagated to the adjacent URLs for computation in next iteration. To implement such ranking mechanisms on URLs distributed across an overlay network, we need to preserve the adjacency relationships in hyperlink graph while mapping URLs to nodes. If hyper-linked URLs are mapped to same node or adjacent nodes then network overhead for computing URL weights will be significantly reduced. Unfortunately, there exists no straight forward, hyperlink structure preserving mapping of the Web to an overlay network.

In DEWS, we retain the hyperlink structure as a virtual overlay on top of Plexus overlay. We use a standard shift-add hash function $(\hbar(\cdot))$ to map a URL, say u_i, to a

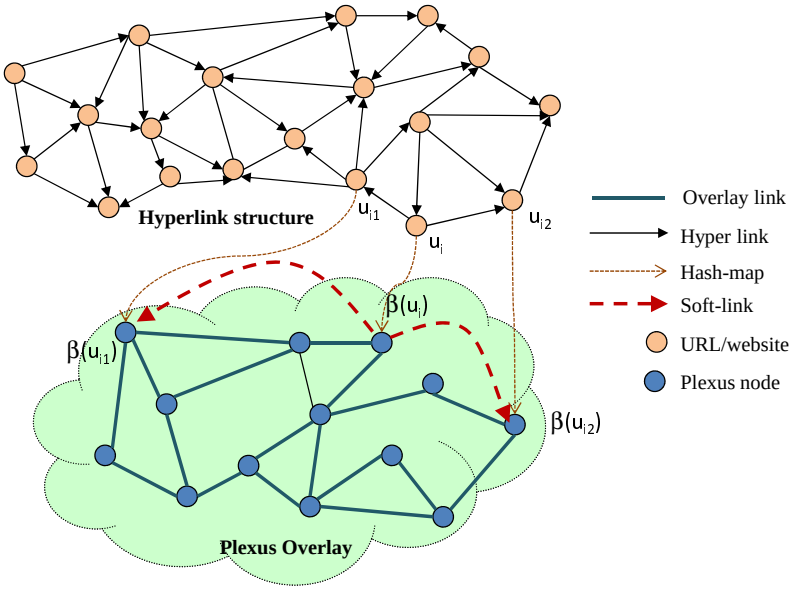


Figure 3.3: Hyperlinks to Plexus overlay mapping

codeword, say $c_k = \hbar(u_i)$. Then we use Plexus routing to lookup $\beta(u_i)$, which is the node responsible for indexing codeword c_k. For each outgoing hyperlink, say u_{it}, of u_i we find the responsible node $\beta(u_{it})$ in a similar manner. During distributed link-structure analysis $\beta(u_i)$ has to frequently send weight update messages to $\beta(u_{it})$.

The index stored in $\beta(u_i)$ for URL u_i has the form $< u_i, w_i, \{< u_{it}, \beta(u_{it})\} >$, where w_i is the link structure weight of u_i and $\beta(u_{it})$ is the soft-link, i.e., cached network address, of node $\beta(u_{it})$ placed in node $\beta(u_i)$. Figure 3.3 illustrates the mechanism of mapping the hyper-link overlay to the Plexus overlay. This figure shows that URLs u_i, u_{i1}, and u_{i2} are mapped to the nodes $\beta(u_i)$, $\beta(u_{i1})$, $\beta(u_{i2})$, respectively in the Plexus overlay and the index in $\beta(u_i)$ for u_i contains links to the $\beta(u_{i1})$ and $\beta(u_{i2})$ to preserve the hyper-link structure form the hyper-link overlay.

3.4.2 Distributed inverted index

We use Plexus to build an inverted index on the important keywords extracted from each website. This allows us to lookup a query keyword and find all the websites containing that keyword by forwarding the query message to a small number of nodes in the network. Figure 3.4 illustrates the mechanism of creating inverted index for a particular website and advertising it to the Plexus overlay.

- Suppose, $\mathcal{K}_i^{rep} = \{k_{ij}^{rep}\}$ is the set of representative keywords for website u_i.
- For each keyword k_{ij}^{rep} in $\mathcal{K}_i^{rep}$, we generate its phonetic code k_{ij}^{dmp} obtained by

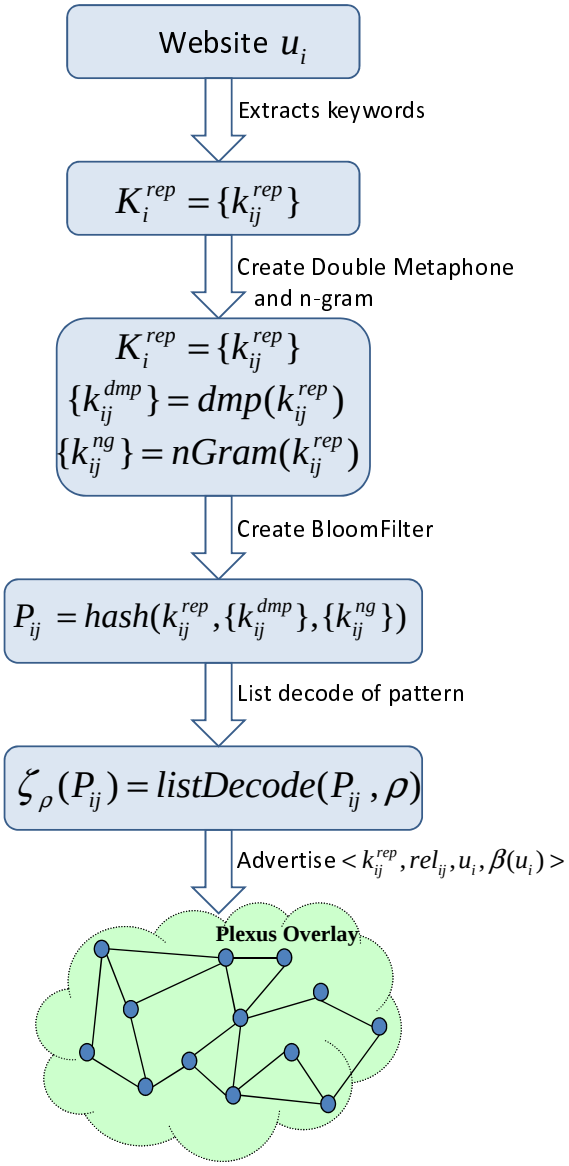


Figure 3.4: Construction of inverted index

applying Double Metaphone encoding (explained in Section 2.6.6) on k_{ij}^{rep}. We use phonetic encoding to reduce the hamming distance between the advertised pattern and search pattern which eventually increases the efficiency of search results. Misspelled search keywords don't have impact on the quality of search results. We compute a list of n-grams (explained in Section 2.6.9) $\{k_{ijz}^{ng}\}$ for each keyword k_{ij}^{rep} in $\mathcal{K}_i^{rep}$. We use n-grams of keywords which allows partial matching and increases the search accuracy in approximate matching.

- We encode the original keyword k_{ij}^{rep} along with the generated keywords k_{ij}^{dmp} and $\{k_{ijz}^{ng}\}$ into an n-bit Bloom filter (explained in Section 2.6.5) P_{ij}.

- We use P_{ij} as a pattern in $\mathbb{F}_2^n$ and list decode it to a set of codewords, $\zeta_\rho(P_{ij}) = \{c_k | c_k \in \mathcal{C} \wedge \delta(P_{ij}, c_k) < \rho\}$, where $\zeta_\rho(\cdot)$ is list decoding function and ρ is list decoding radius.

- Finally, we use Plexus routing to lookup and store the index for k_{ij}^{rep} at the nodes responsible for codewords in $\zeta_\rho(P_{ij})$.

The index for k_{ij}^{rep} is a quadruple $< k_{ij}^{rep}, rel_{ij}, u_i, \beta(u_i) >$, where rel_{ij} is a measure of semantic relevance of k_{ij}^{rep} to u_i. We use $\gamma(k_{ij}^{rep})$ to represent the set of nodes responsible for k_{ij}^{rep}. Evidently, $\gamma(k_{ij}^{rep}) \equiv lookup(\zeta_\rho(BF(k_{ij}^{rep} \cup k_{ij}^{dmp} \cup \{k_{ijz}^{ng}\})))$, $BF(\cdot)$ represents Bloom filter encoding function.

Phonetic codes and n-grams along with list decoding technique provides flexible searching with high accuracy in *DEWS*, which is justified with simulation results in Section b.

3.5 Distributed indexing, searching and ranking

This layer is responsible for indexing websites, searching and computing ranks in a distributed manner. Metrics used for ranking web search results can be broadly classified into two categories: a) keyword to document relevance and b) hyperlink structure of the webpages. Techniques from Information Retrieval (IR) literature are used for measuring relevance ranks. While link structure analysis algorithms like PageRank [38], HITS [28] *etc.*, are used for computing weights or relative significance of each URL. In DEWS, we use both of these measures for ranking search results.

3.5.1 Website Indexing

The pseudo code for indexing a website is presented in Algorithm 3. As discussed in Section 3.4, we maintain two sets of indexes for a website: a) using site URL u_i and b) using representative keywords $\{\mathcal{K}_i^{rep}\}$. In lines 3 to 8 of Algorithm 3, we compute the index on u_i, which involves computing the soft-links ($\beta(u_{it})$) for each outgoing hyper-links from u_i and storing in node $\beta(u_i)$. In lines 9 to 19, we compute the indexes on $\mathcal{K}_i^{rep}$ and advertise the indexes to the responsible nodes. The computation of rel_{ij} in line 14 is discussed in Section 3.6.1.

Algorithm 3 *IndexWebsite*

Inputs:
  u_i: URL of the website to be indexed
Functions:
  $\hbar(u_i)$: hash map u_i to a codeword
  $\gamma_r(P)$: $\{c_k | c_k \in \mathcal{C} \wedge \delta(P, c_k) \leq r\}$
  $lookup(c_k)$: finds the node that stores c_k
$\beta(u_i) \leftarrow lookup(\hbar(u_i))$
for all out-link u_{it} of $\{u_i\}$ **do**
  $\beta(u_{it}) \leftarrow lookup(\hbar(u_{it})))$
end for
$w_i \leftarrow$ initial PageRank of u_i
store $< u_i, w_i, \{u_{it}, \beta(u_{it})\} >$ to node $\beta(u_i)$
$\mathcal{K}_i^{rep} \leftarrow$ set of representative keywords of u_i
for all k_{ij}^{rep} in $\mathcal{K}_i^{rep}$ **do**
  $\{k_{ij}^{dmp}\} \leftarrow DoubleMetaphoneEncode(k_{ij}^{rep})$
  $\{k_{ij}^{ng}\} \leftarrow nGramEncode(k_{ij}^{rep})$
  $P_{ij} \leftarrow BloomFilterEncode(\{k_{ij}^{rep}\} \cup \{k_{ij}^{dmp}\} \cup \{k_{ij}^{ng}\})$
  $rel_{ij} \leftarrow$ relevance of k_{ij}^{rep} to u_i
  for all c_k in $\zeta_\rho(P_{ij})$ **do**
    $v \leftarrow lookup(c_k)$
    store $< k_{ij}^{rep}, rel_{ij}, u_i, \beta(u_i) >$ to node v
  end for
end for

3.5.2 Distributed PageRank

For ranking search results, we have adapted the original PageRank [38] algorithm to the decentralized environment in DEWS. In centralized PageRank algorithm, global weights for each webpage are computed based on the incoming and outgoing links of a particular web page. In DEWS, we compute PageRank for each website u_i and index them using Plexus indexing mechanism at node $\beta(u_i)$ (see Algorithm 3). The PageRank computation equation for each website is as follows:

$$w_i = (1 - \eta) + \eta \sum_{t=1}^{g} \frac{w_{it}}{L(u_{it})} \tag{3.1}$$

Here, w_i is PageRank for website u_i and η is the damping factor for PageRank algorithm. η is usually assigned a value of 0.85. $\{u_{it}\}$ is the set of websites linked to u_i and $L(u_{it})$ is the number of outgoing links from website u_{it}.

Each node periodically executes Algorithm 4 to compute the PageRank weights in a distributed manner. To communicate PageRank information between the nodes, we use a PageRank message containing the triplet $< u_s, u_i, \frac{w_s}{L(u_s)} >$, where node $\beta(u_s)$ sends the message to node $\beta(u_i)$ and $\frac{w_s}{L(u_s)}$ is the contribution of u_s towards PageRank weight of u_i. Each node maintains a separate message queue for each URL it has indexed. In a message queue, incoming PageRank messages are stored for a pre-specified period of time or the queue length exceeds the expected in-degree of that URL. The messages gathered in a message queue are used to compute the PageRank for each URL according to Equation 3.1. If the change in newly computed PageRank value is greater than a pre-defined threshold θ then PageRank update messages are sent to the nodes responsible for each out linked URL.

PageRank algorithm requires many cycles to converge. In each cycle, node $\beta(u_i)$ responsible for URL u_i has to lookup and send PageRank update message to node $\beta(u_{it})$ for each out-linked URL u_{it}. To reduce network overhead due to repeated lookup of node $\beta(u_{it})$, we cache the network address (soft-link) of node $\beta(u_{it})$ at node $\beta(u_i)$. Node $\beta(u_i)$ looks up node $\beta(u_{it})$ for the first time using Plexus. For sending subsequent update messages node $\beta(u_i)$ uses the soft-link to directly send update messages to node $\beta(u_{it})$.

PageRank for URL u_i is computed and maintained in node $\beta(u_i)$, while the computed PageRank value w_i is used in nodes $\gamma(k_{ij}^{rep})$, where a representative keyword k_{ij}^{rep} for website u_i is indexed. The Web is continuously evolving and PageRank for the websites are likely to change over time. As a result, storing PageRank w_i to

Algorithm 4 *Update PageRank*

Internals:
  $\mathcal{Q}_{u_i}$: PageRank message queue for u_i
  $L(u_i)$: Number of outlinks for u_i
  w_i: PageRank weight of u_i
  η: Damping factor for PageRank algorithm
  θ: Update propagation threshold
for all URL u_i indexed in this node $\beta(u_i)$ **do**
  $temp \leftarrow 0$
  for all $< u_{si}, u_i, \frac{w_{si}}{L(u_{si})} > \in \mathcal{Q}_{u_i}$ **do**
    $temp \leftarrow temp + \frac{w_{si}}{L(u_{si})}$
  end for
  $w_i^{new} \leftarrow (1-\eta) + \eta * temp$
  if $|w_i^{new} - w_i| > \theta$ **then**
    $w_i \leftarrow w_i^{new}$
    for all out link u_{it} from u_i **do**
      send PageRank message $< u_i, u_{it}, \frac{w_i}{L(u_i)} >$ to $\beta(u_{it})$
    end for
  end if
end for

node $\gamma(k_{ij}^{rep})$ will not be sufficient; we have to refresh it periodically. To reduce network overhead, softlink to $\beta(u_i)$ are stored in nodes $\gamma(k_{ij}^{rep})$. The softlink structure between nodes $\beta(u_i)$, $\beta(u_{it})$ and $\gamma(k_{ij}^{rep})$ is presented in Figure 3.5.

3.5.3 Search queries and rank results

DEWS breaks down the query into sub-queries each consisting of a single query keyword, say q_l. Similar to the keyword advertisement process explained in Section 3.4.2, we compute the Double Metaphone (*i.e.*, q_l^{dmp}) and n-gram (*i.e.*, q_l^{ng}) of q_l and encode them in a Bloom filter P_l. Then we use Plexus framework to find the nodes responsible for storing the keywords similar to q_l and retrieve a list of triplets like $\{< u_i, w_i, rel_{il} >\}$, which gives us the URLs (u_i) containing query keyword q_l along with the link structure weight (w_i) of u_i, and semantic relevance of q_l to u_i, *i.e.*, rel_{il}. Now, the querying node computes the ranks of the extracted URLs using the following equation:

$$rank(u_i) = \sum_{q_l} \sum_{u_i} \vartheta_{il}(\mu \cdot w_i + (1-\mu) \cdot rel_{il}) \tag{3.2}$$

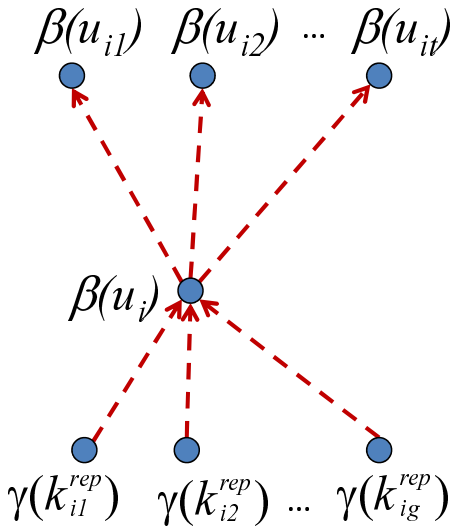


Figure 3.5: Softlink structure in DEWS

In Equation 3.2, μ is a weight adjustment factor governing the relative importance of link structure weight (w_i) and semantic relevance (rel_{il}) in the rank computation process. While ϑ_{il} is a binary variable that assumes a value of one when website u_i contains keyword q_l and zero otherwise.

The query process in DEWS is explained in Algorithm 5. In this algorithm, we have used separate $lookup(c_k)$ for each of the target codeword c_k. In practice separate lookup of each target is very expensive in terms of network usage. Instead, we have used the extended multicast routing mechanism with route aggregation as explained in Section 3.3.2.

We compute the relevance of a query in a website computed in line 10 in Algorithm 5 by the Algorithm 6. r_{ij} in line 6 of Algorithm 6 is the relevance of the keyword k_{ij}^{rep} in the website u_i, which is stored during indexing of u_i (line 14 in Algorithm 3). If we use BM25 as the measurement of relevance, r_{ij} should be simply term frequency (tf) of k_{ij}^{rep} instead of computing relevance considering structure of website and webpage (see Section 3.6.1).

In line 6 of Algorithm 6, inverse document frequency $idf(k_{ij}^{rep})$ is computed as follows:

$$idf(k_{ij}^{rep}) = \log \frac{U}{\psi(k_{ij}^{rep})} \tag{3.3}$$

Here, U is the total number of websites and $\psi(k_{ij}^{rep})$ is the number of websites having keyword k_{ij}^{rep}. $tf(k_{ij}^{rep})$ is a measure of the relevance of k_{ij}^{rep} to u_i, while $idf(k_{ij}^{rep})$ is a measure of relative importance of k_{ij}^{rep} *w.r.t.* other keywords. idf is

Algorithm 5 $Search(Q, \rho, T)$

Input:
  Q: set of query keywords $\{q_l\}$
  T: Most relevant T websites requested
  ρ: list decoding radius
Internals:
  μ: Weight adjustment on link-structure vs relevance
$\xi \leftarrow$ *empty associative array*
for all $q_l \in Q$ **do**
  $\{q_l^{dmp}\} \leftarrow DoubleMetaphoneEncode(q_l)$
  $\{q_l^{ng}\} \leftarrow nGramEncode(q_l)$
  $P_l \leftarrow BloomFilterEncode(q_l \cup \{q_l^{dmp}\} \cup \{q_l^{ng}\})$
  for all $c_k \in listDecode_\rho(P_l)$ **do**
    $n \leftarrow lookup(c_k)$
    for all $\{< u_i, w_i, rel_i >\} \in n.retrive(Q)$ **do**
      $\xi[u_i].value \leftarrow \xi[u_i].value + \mu \cdot w_i + (1 - \mu) \cdot rel_i$
    end for
  end for
end for
sort ξ based on *value*
return top T u_i from ξ

Algorithm 6 $Relevance(Q, k_i^{rep})$

Input:
  Q: set of query keywords $\{q_l\}$
  k_i^{rep}: set of representative keywords
$rel_i \leftarrow$ *empty*
for all $q_l \in Q$ **do**
  for all $k_{ij}^{rep} \in k_i^{rep}$ **do**
    if q_l matches with k_{ij}^{rep} **then**
      $rel_i \leftarrow rel_i + r_{ij} * idf(k_{ij}^{rep})$
    end if
  end for
end for
return rel_i

used to prevent a common term from gaining higher weight and a rare term from having lower weight in a collection.

Computing $tf(k_{ij}^{rep})$ for each keyword $k_{ij}^{rep} \in \mathcal{K}_i^{rep}$ from website u_i is straight forward and can be done by analyzing the pages in u_i. For computing $idf(k_{ij}^{rep})$ we need to know two entities, namely U and $\psi(k_{ij}^{rep})$. Now, all documents containing keyword, say k_{ij}^{rep}, are indexed at the same node. Hence, $\psi(k_{ij}^{rep})$ can be computed by searching the local repository of that node. However, it is not trivial to compute U in a purely decentralized setup. We use the total number of indexed URLs in a node in place of U as advocated in [32].

3.6 Web indexing and searching

We assume that machines running one or more webservers provide search functionalities and index their hosted websites.

3.6.1 Website indexing

We assume that an AI (Artificial Intelligence) based keyword extractor is available to extract representative keywords from a website. We compute the relevance of each keyword in the following way and utilize the function *IndexWebsite* (explained in algorithm 3) provided by 'Distributed indexing, search and ranking layer' to index the website.

We utilize structure of a website and a webpage in the computation of keyword relevance to a particular website. We compute PageRank [38] among hyperlink structure of a website to determine weight of each page. We assume each webpage has title, plain texts and anchor texts. We assign different weights to a keyword based on its location, e.g., keyword in a title gets more weight than that in a plain text. We also take into account the frequency of the keyword in that page. We compute keywords relevance r_{ij} of a keyword k_{ij}^{rep} in a website u_i is as follows:

$$r_{ij} = \varepsilon * \sum_{l=1}^{n} \varpi_l * (\gamma_t * tf_{tl}(k_{ij}^{rep}) + \gamma_p * tf_{pl}(k_{ij}^{rep}) + \gamma_a * tf_{al}(k_{ij}^{rep}))$$

In the above equation, ε is a factor which measures how the keyword is common in a collection and determined by the component responsible for extracting k_{ij}^{rep} from u_i. The value of ε can be varied from 0.5 to 1.0. If the keyword is common in the collection, it gets lower value. ϖ_l is the PageRank value of the l^{th} webpage

in u_i, which is computed in the local machine utilizing the hyperlink structure of u_i. $tf_{tl}(k_{ij}^{rep})$, $tf_{pl}(k_{ij}^{rep})$, and $tf_{al}(k_{ij}^{rep})$ are the number of occurrences of k_{ij}^{rep} in title, plain text and anchor text of the l^{th} webpage, respectively. γ_t, γ_p, and γ_a are the positional value of a keyword in title, plain text and anchor text, respectively where $\gamma_t > \gamma_p > \gamma_a$.

3.6.2 Search and incremental retrieval

Users can search the Web through a searching interface provided by *DEWS*. DEWS is able to search with exact keywords, partial keywords and misspelled keywords. It returns the top-k search results corresponding to a query Q utilizing Algorithm 5. *DEWS* allows incremental retrieval. Incremental retrieval refers to gradually retrieving search results in parts from a repository or server, as offered by almost all Web search engines. Though it is a challenging problem to achieve incremental retrieval in a distributed setup, an appropriate solution to the problem can save us valuable network bandwidth.

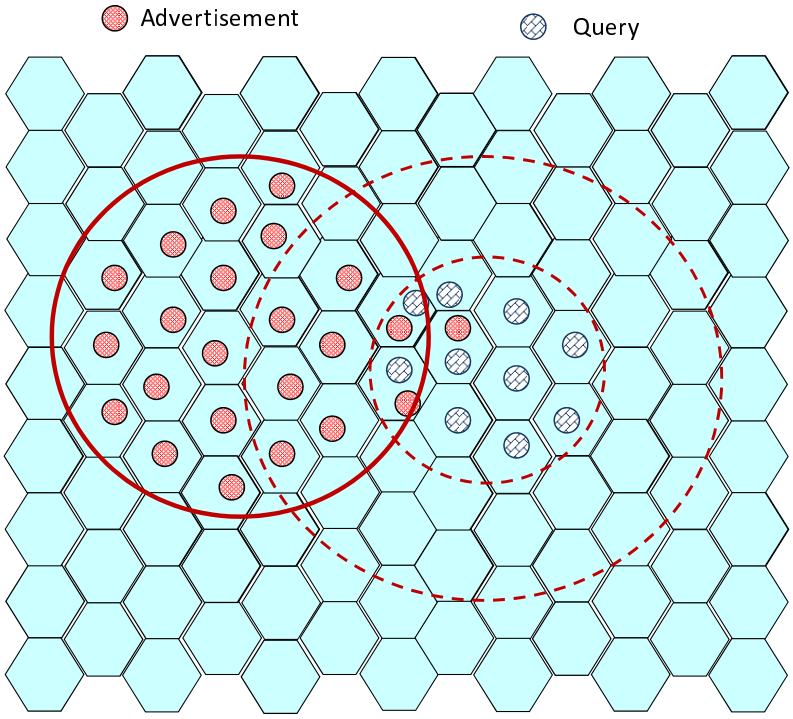


Figure 3.6: Incremental retrieval

We have exploited the Hamming distance based lookup capability of Plexus to enable distributed incremental retrieval in DEWS. In Algorithm 5, list decoding radius ρ can be varied to control the Hamming distance of a query pattern from

the advertised patterns in search result. As explained in Figure 3.6, we start with a small list decoding radius ρ close to half of the minimum distance (d) between any pair of codewords of the Reed-Muller code used for routing. For any query, the closest matching advertised keywords can be found within this radius. By increasing the list decoding radius we can find additional codewords, further away from the query pattern. We repeat the search with these additional codewords if the user requires additional results or not enough result is found in the first round. For most of the cases, desired number of results can be found in the first round, which saves a lot of network bandwidth.

3.7 Summary

In this chapter, we presented a decentralized search engine named *DEWS*, which indexes and searches the Web in a distributed manner. Webservers are organized using the Plexus P2P overlay. We extended the Plexus routing protocol for aggregating route messages in each node towards the destinations and forwarding them in such a way that average number of logical hops per message is reduced. In *DEWS*, each webserver indexes its hosted websites in the Plexus overlay collaborating with other webservers. We preserve the hyper-link structure on the Plexus overlay by caching the addresses of the webservers which index the out-linked URLs. We defined these caches as soft-links which allow *DEWS* to compute PageRank similar to the central computation of PageRank with low network overhead. *DEWS* computes *tf* and *idf* efficiently similar to other DHT-based approaches. We have used phonetic encoding and n-grams during keyword advertisement and query resolution process which enable accurate searching in presence of misspelled and partially specified keywords. *DEWS* also employs incremental retrieval, which allows to search with a smaller decoding radius and network bandwidth. *DEWS* maintains *replica* for better routing and fault resilience.

Chapter 4

Evaluation

4.1 Introduction

We have presented the framework of *DEWS* in Chapter 3. In this chapter, we evaluate its performance using simulations. Performance metrics used in this evaluation include routing efficiency, indexing overhead, convergence time, network overhead, and accuracy of link-structure analysis. In the experiments, we have varied the number of URLs, number of queries, and network size to measure the scalability and robustness of DEWS.

The organization of the rest of this chapter is as follows. In Section 4.2, we describe the performance metrics for evaluating *DEWS*. Overview of simulation setup is presented in Section 4.3. Section 4.4 discusses the performance of *DEWS* based on experimental results. Finally, Section 4.5 further discusses our findings.

4.2 Performance Metrics

We identify the following performance metrics to evaluate *DEWS*:

- **Accuracy of ranking**
  Ranking accuracy of search results has a significant impact on users' satisfaction and gaining popularity as a good search engine. A good search engine should have an efficient ranking mechanism to present the most relevant queried information in the first few search results. We measure accuracy of ranking by Spearman's footrule distance (*SFD*). *SFD* is used to compute positional differences of items (topics, URLs) in two lists. For two ordered lists σ_1 and σ_2 of size k each, *SFD* is defined as $F(\sigma_1, \sigma_2) = \frac{\sum_{i=1}^{k} |\sigma_1(u_i) - \sigma_2(u_i)|}{k*k}$,

where $\sigma_1(u_i)$ and $\sigma_2(u_i)$ are the positions of URL u_i in σ_1 and σ_2, respectively. If a URL is present in one list and absent in the other, its position in the latter list is considered as $k+1$. We use PageRank and BM25 for link weight and keyword relevance, respectively for ranking search results in *DEWS*. We have computed *SFD* for PageRank and BM25 separately to measure the accuracy of ranking in *DEWS*.

- **Flexible search and accuracy of search results**

  Search flexibility is an essential feature of a good search engine as the users may not have the exact knowledge of their requested information. Search keywords in presence of spelling mistakes or partial specification are common in the queries. A good search engine should allow flexible search, i.e., searching with partially specified and misspelled keywords. We have varied edit distances between the queried and advertised keywords to measure the search flexibility and accuracy of search results. We have used precision and recall to measure the accuracy of search results. High precision indicates that search results containing more relevant topics to the queries. Precision is defined as follows:

$$Precision = \frac{|retrieved\ documents \cap relevant\ documents|}{|retrieved\ documents|} \tag{4.1}$$

  Recall indicates the percentage of relevant information retrieved. Recall is defined as follows:

$$Recall = \frac{|retrieved\ documents \cap relevant\ documents|}{|relevant\ documents|} \tag{4.2}$$

- **Incremental retrieval**

  Incremental retrieval allows a search engine to present the most relevant results in the first set of results and more results later on users' requests, if required. Usually users are satisfied with the first set of results (most relevant) of a good search engine. Hence incremental retrieval may reduce the network bandwidth and query resolution time by searching a smaller and most relevant information. We have measured recall and precision of the search results and network bandwidth varying the list decoding radius during query resolution.

- **PageRank convergence**

  PageRank computation is an iterative process. An efficient distributed PageR-

ank algorithm should converge quickly in a large network with low network overhead (number of messages). We have measured time (i.e., number of cycles to reach the situation when there is no more ranking messages in the network) to converge PageRank algorithm in *DEWS*.

- **Routing efficiency**
  Routing is the core feature of a distributed search engine. Efficient routing mechanism allows a search engine indexing and searching the websites with lower network overhead and query resolution time. We have modified the Plexus routing mechanism (Section 3.3.2), which allows *DEWS* routing a set of messages to the destination nodes with a limited number of logical hops. We have measured the efficiency of *DEWS* routing as the percentage of hop reduction from the pair-wise Plexus routing.

- **Network bandwidth**
  A distributed search engine should index, search, and compute weight of websites with low network bandwidth. We have measured the network bandwidth as the average number of logical hops required during these processes.

- **Index overhead**
  A distributed search engine should index the websites with a uniform distribution over all the participating webservers. We have measured the indexing overhead in *DEWS* as the average number of URLs, keywords and softlinks indexed in each node.

- **Fault resilient**
  We have used P2P overlay to organize the webservers. In Peer-to-Peer network, peers go down and up dynamically. We assume that webservers are more stable compared to the peers in any P2P network. A good search engine should provide the search results with high efficiency in presence of a few webservers failures. For this reason, we have measured the impact of webserver failures on searching and routing performance of *DEWS*.

- **Scalability**
  In any distributed system, scalability is an essential feature which requires that new resources can be attached to the system without degrading its performance. For this reason, we measure the efficiency of website indexing (i.e.,

URL and keyword advertisements) and searching while varying the number of websites.

4.3 Simulation setup

4.3.1 Overview of simulation

We have measured various performance metrics under diverse network conditions to evaluate *DEWS*. Measurements have been taken under varied network sizes, number of websites, number of queries, query keywords with varied edit distances, incremental retrieval and node failures. All experiments are done using a queuing model based cycle-driven simulator, where each peer is explicitly modeled using a message queue. In each cycle all nodes get their fair chance to process own message queue in parallel with other nodes. A message queue of a node may contain URL lookup, URL advertisement, keywords advertisement, keyword search, and rank update messages. During the processing of message queue, each message from the queue is retrieved and processed based on the type of the message.

4.3.2 Data Set

We have used LETOR 3.0 dataset [31] for our experiments which is a package of benchmark data sets designed for research on ranking. The dataset is composed of the TREC 2003 and 2004 datasets, which contain a crawl of the *.gov* domain done on January, 2002. There are a total of 1,053,110 html documents and 11,164,829 hyperlinks in the collection. The collection contains three search tasks: topic distillation, homepage finding, and named page finding. TREC 2003 track contains 50, 100, and 150 queries in the above categories respectively and TREC 2004 contains 75 queries per category.

In our experiments, we have refined the dataset available under "$Gov \backslash Feature$" where NULL or missing values are replaced with feature wise minimum values. For computing the pagerank of the HTML documents in the dataset we have used the "Sitemap" of the *.gov* collection available from LETOR 3.0 archive.

4.4 Results and evaluation

In this section, we present the experimental results obtained by simulating the *DEWS* framework. We have presented and evaluated the performance of searching, ranking, routing, and indexing including the ability of fault resilience in *DEWS* using the performance metrics discussed in Section 4.2.

4.4.1 Searching performance

We present the searching performance of *DEWS* according to three points of view: (a) search flexibility and accuracy, (b) impact of phonetic encoding and n-grams on search flexibility and accuracy, and (c) incremental retrieval.

(a) **Search flexibility and accuracy**

DEWS provides flexible searching, i.e., searching with partially specified or misspelled keywords. We indexed 10,000 URLs and associated representative keywords in networks of varied number of nodes. We generated 10,000 query keywords from the randomly selected indexed keywords by varying edit distances from one to three.

Figure 4.1(a) presents average recall of the search results with varied network sizes. This graph shows that recall rate remain constant at 100%, 98%, and 87% for edit distances one, two, and three, respectively. Recall rate is lower for query keywords of higher edit distances because hamming distances between the advertisement pattern and search pattern increase which decrease the number of matched codewords in the advertised and searched codewords.

Figure 4.1(b) presents precision of search results for varied number of queries from 1000 to 10,000. From this graph, it is observed that precision remains constant at 92%, 88%, and 76% for edit distances one, two, and three, respectively. Precision becomes lower when the edit distance increases because irrelevant websites are included in the search results during approximate matching with the query keywords.

(b) **Impact of phonetic encoding and n-grams**

We use Double Metaphone encoding to create advertisement patterns and search patterns during advertisement of inverted index and searching query keywords, respectively. Two facts about phonetic encoding are as follows: i) any two phonetically equivalent keywords have no edit distance between them, ii) phoneti-

cally in-equal keywords have less edit distance than the edit distance between the original keywords. In both cases, hamming distance between advertisement and search patterns is less than the hamming distance between the patterns using original keywords. This low hamming distance increases the percentage of common codewords computed during advertisement and search, which eventually increases the possibility of finding relevant websites.

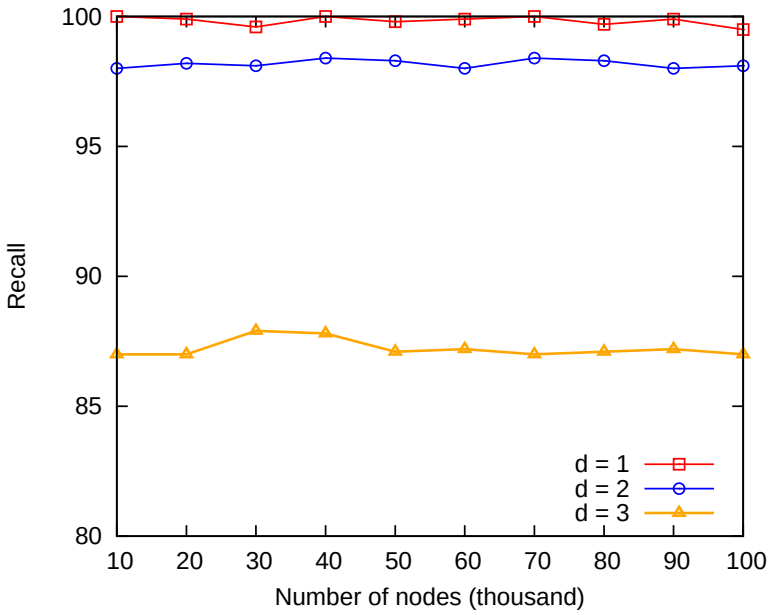
We use n-grams of keywords during pattern creation for both advertisement and search keywords, which enhances flexible search. We randomly selected 10K keywords and generated query keywords from them. We computed the percentage of matched n-grams between a keyword and its corresponding query keywords varying the value of n, which is presented in Figure 4.2(a). It can be observed that percentage of matched 'n-grams' reduces for larger value of n. For this reason, we use 1-gram to increase the possibility of matching in presence of partially specified query keywords.

Figure 4.2(b) shows the average recall of the search results for 10,000 queries in a network of 100K nodes where the query keywords have edit distance two with the advertised keywords. We used three different options to create the advertisement and search patterns. In this figure, we use notations 'dmp', and 'n-grams' to refer to Double Metaphone encoding and n-grams, respectively. From this figure, it can be observed that the use of Double Metaphone encoding, n-grams and list decoding all together during pattern creation provides the best recall rate.

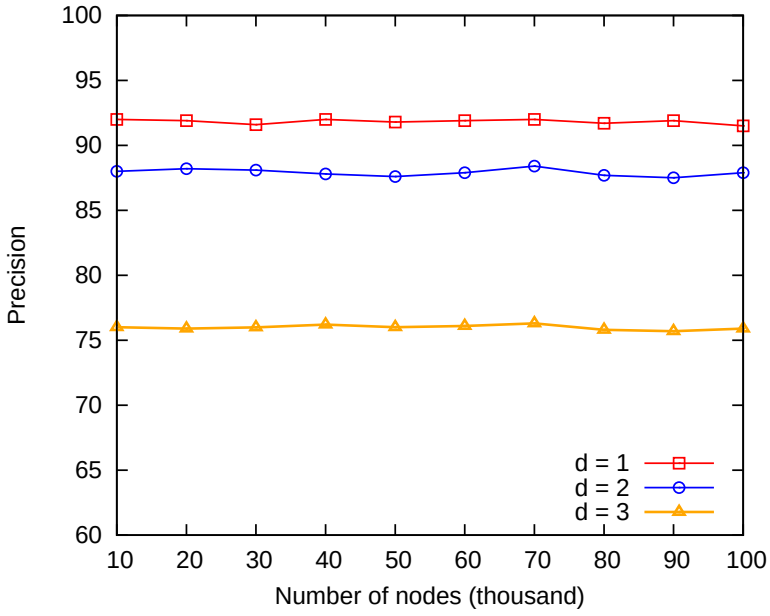
(c) **Incremental retrieval**

We varied the search keywords having edit distances one to three from the advertised keywords and measured the recall and precision of the search results. We measured these two metrics varying the list decoding radius. Figure 4.3(a) presents the recall of the search results for the four steps (with larger list decoding radius in subsequent steps) with search keywords having one to three edit distances from original keywords. This graph shows that recall for keywords having edit distances one and two are around 100% while recall for keywords having edit distance three is about 87% in first step and gradually increases in subsequent steps. Figure 4.3(b) shows precision for the results found in the four subsequent steps. In the first step, the precisions are 92%, 88% and 76% for edit distances one, two and three. This figure represents two facts: i) precision decreases with the increase of edit distance between advertised and searched

keywords, ii) precision decreases slightly in the subsequent steps. The reason is the set of common nodes between the indexing set and search set of nodes decreases when edit distance and decoding radius increases. As we apply approximate matching between the search and indexed keywords, *DEWS* picks some irrelevant websites during approximate search. Figure 4.3(c) shows the network bandwidth consumption (number of logical hops) in different steps for 1000 simultaneous queries in a network of 100K nodes. *DEWS* requires five hops on average in each step to resolve the queries. Thus, the total number of hops increases in the subsequent steps. As *DEWS* provides search results with high accuracy in the first step, subsequent steps can only be performed when users are not satisfied with the presented results to reduce the network bandwidth consumption. The reason is if we search with greater decoding radius in first attempt, it requires to visit greater number of nodes.

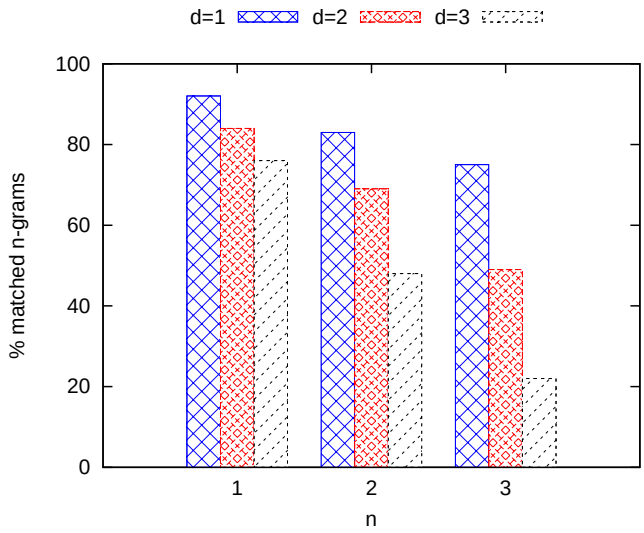


(a) Recall

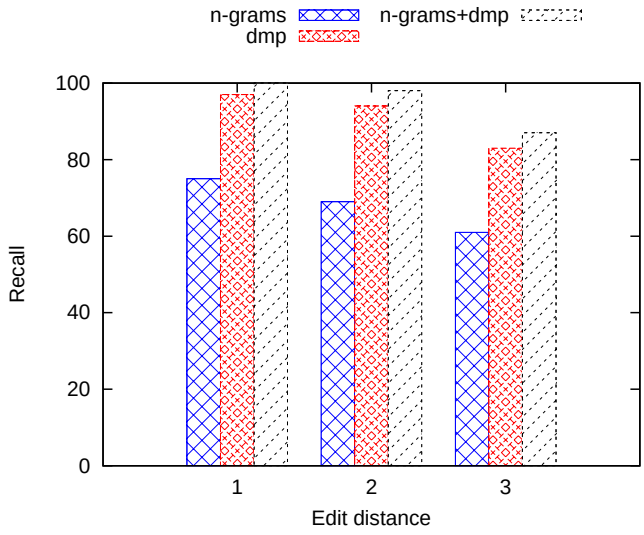


(b) Precision

Figure 4.1: Accuracy of search results

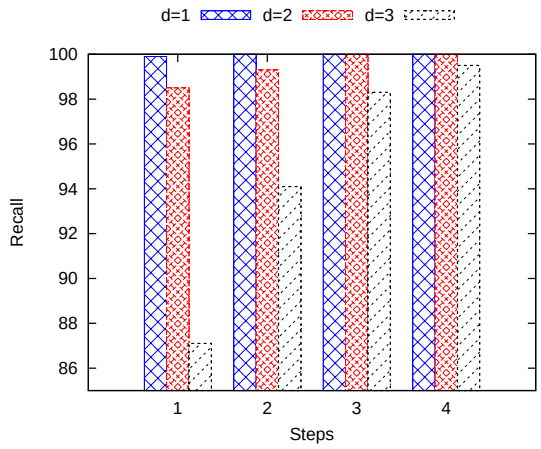


(a) Percentage of mached n-grams

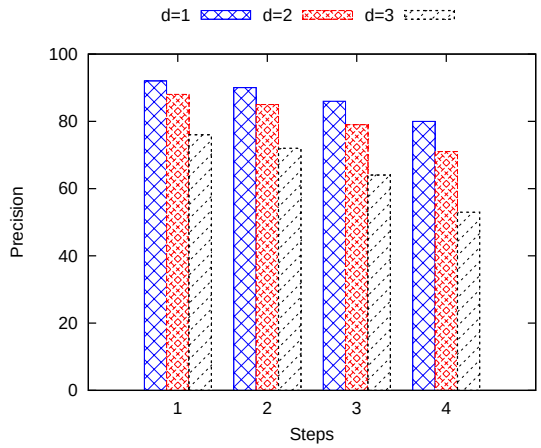


(b) Impact of phonetic encoding and n-grams

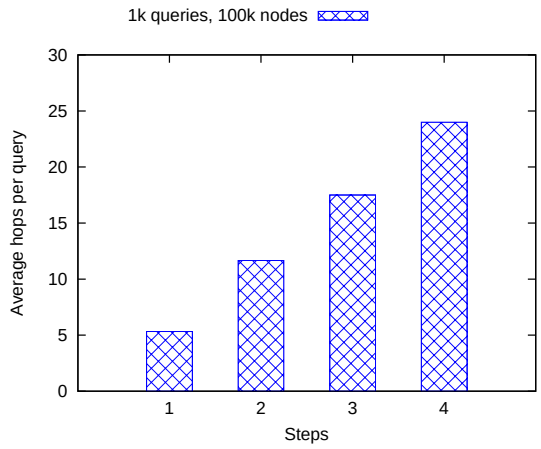
Figure 4.2: Impact of phonetic encoding and n-grams on search performance



(a) Recall



(b) Precision



(c) Network bandwidth

Figure 4.3: Incremental retrieval

4.4.2 Ranking performance

This section presents the ranking performance of *DEWS* in three points of view: (a) time and network overhead for convergence of PageRank algorithm, (b) accuracy of PageRank computation, and (c) accuracy of BM25 computation.

(a) **PageRank convergence**

PageRank computation is an iterative process as discussed in Chapter 3. We assume that it converges when there is no rank update messages in the network. We measured the number of cycles required to converge the algorithm. We indexed varied number of URLs from 1000 to 10,000 in a network of 50,000 nodes. We used different interval times for sending weight update messages. Figure 4.4(a) presents the results of this experiment. It shows that PageRank converges within 60 cycles using one cycle interval. From this figure, it is observed that convergence time increases when the number of URLs increases. The underlying fact can be explained by Figure 4.4(b).

Figure 4.4(b) shows the average number of out-going links from each URL when different numbers of URLs are selected from the LETOR3.0 dataset. This is because when n number of URLs were selected, out-linked URLs within the n URLs were chosen and the rest discarded. It is observed from Figure 4.4(b) that the average number of out-linked URLs increases linearly with the number of URLs.

As the number of out-linked URLs increases linearly, the number of weight update messages increases which eventually increases the number of cycles to converge PageRank in *DEWS*. We claim that if the number of out-links does not increase, then number of URLs has no significant impact on convergence time. Another observation found from Figure 4.4(a) is that convergence time increases when the interval time for sending PageRank update messages increases.

We have measured the average number of messages generated during the convergence of PageRank algorithm by varying the number of URLs from 1000 to 10,000 and interval period one to three cycles, which is shown in Figure 4.4(c). This figure indicates that average number of messages decreases when the number of URLs increases. The reason is with the increase in number of URLs, PageRank value in each URL converge quickly as they receive many PageRank values from their in-linked URLs. Another point is that average number of messages for interval two and three are almost the same and are greater than that

of having an interval of one cycle. The reason is each node gets the opportunity to accumulate the incoming weights and send update messages if required with larger interval.

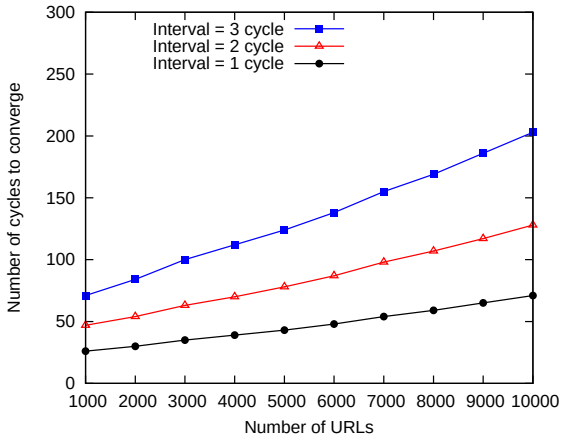
(b) **Accuracy of PageRank computation**

We have measured *SFD* between search results using the PageRank values computed centrally and by *DEWS*. Figure 4.4 shows the *SFD* for top-20, top-100, and top-1000 search results. We indexed 10,000 URLs on a network of 50,000 nodes and initiated PageRank with an update interval of 2 cycles. It is evident from Figure 4.4 that *SFD* drops significantly in the first 60 cycles due to rapid convergence of our distributed PageRank algorithm, while PageRank values become almost constant after 120 cycles. It is observed that *SFD*s become around 0.11, 0.098, and 0.059 after 120 cycles for top-20, top-100, and top-1000 results, respectively. It indicates that the distributed PageRank weights become very close to the centrally computed PageRank weights. Another observation from the figure is that *SFD* for top-1000 is lower than top-20 and top-100. In general, *SFD* is lower for top-k results with higher value of k.

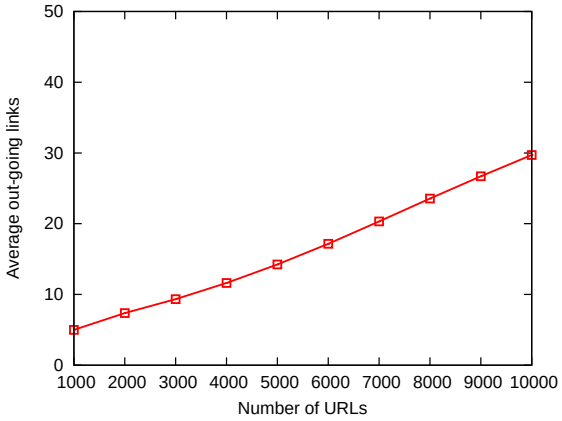
(c) **Accuracy of BM25 computation**

We have used term frequency tf as the relevance of a keyword k_{ij}^{rep} in computation of relevance during website indexing discussed in Chapter 3. idf is computed during the query resolution. Thus, the resultant relevance during query resolution becomes BM25 score of a query keyword to a particular website.

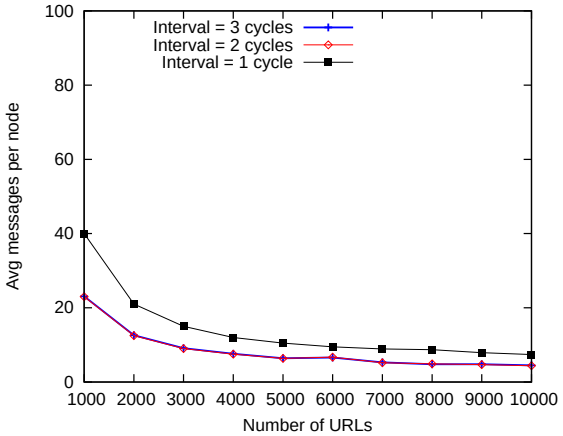
We have compared the search results sorted by BM25 scores within our proposed system with the results sorted using centrally computed BM25 scores. Figure 4.5 presents *SFD* between the two search results for top-20, top-100, and top-1000 results. We indexed 10,000 to 100,000 URLs and their associated keywords on a network of 100,000 nodes. The only difference between the two approaches for computing BM25 scores is that *DEWS* uses an approximate value of N (number of collections) during query resolution instead of exact value. Figure 4.5 shows that *SFD*s remain constant around 0.075, 0.043, and 0.026 for top-20, top-100, and top-1000, respectively. It indicates that *DEWS* computes the BM25 scores efficiently in the proposed distributed manner.



(a) Convergence time



(b) Number of out-links



(c) Network overhead

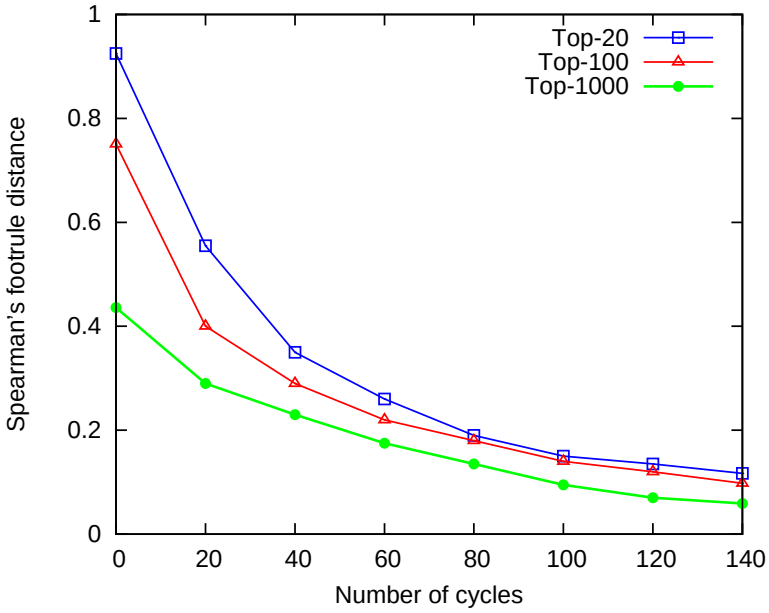


Figure 4.4: Accuracy of PageRank computation

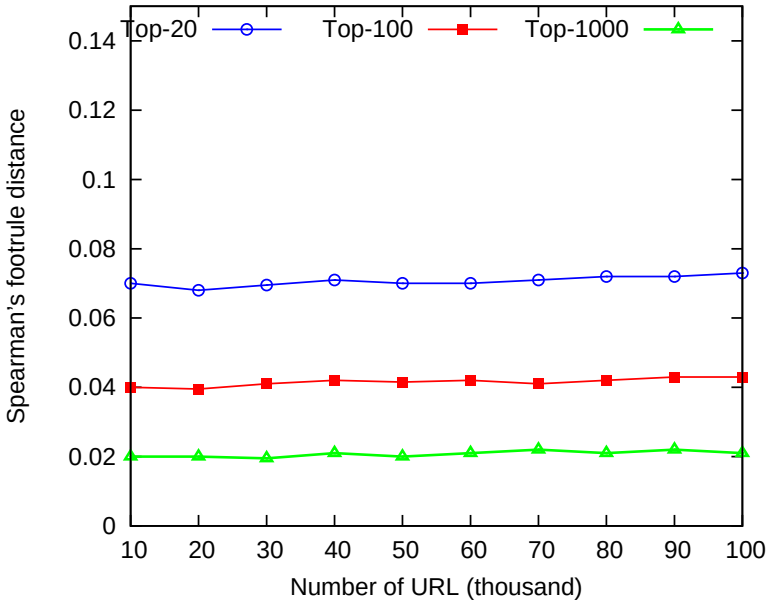


Figure 4.5: Accuracy of BM25 computation

4.4.3 Routing performance

In this section, we discuss the performance of modified Plexus routing, effectiveness of using soft-links and network bandwidth for URL and keyword advertisement.

(a) **Performance of aggregate routing**

We use Plexus routing which is an efficient multicast routing protocol. Plexus routes a message to multiple targets simultaneously which saves a portion of the routing hops that might have occurred if we had used pair-wise routing. We extend the Plexus routing protocol using message aggregation technique in the Section 3.3.2. We have measured the percentage of hop reduction in both Plexus and modified Plexus routing (*DEWS* routing) when the number of destinations increases. Figure 4.6(a) shows the reduction in routing hops (rrh) for Plexus routing and *DEWS* routing calculated as Equation 4.3 and 4.4, respectively.

$$rrh = (1 - \frac{No\ of\ hops\ with\ Plexus\ multicast\ routing}{No\ of\ hops\ for\ pairwise\ routing}) \times 100 \qquad (4.3)$$

$$rrh = (1 - \frac{No\ of\ hops\ with\ \ DEWS\ aggregate\ routing}{No\ of\ hops\ for\ pairwise\ routing}) \times 100 \qquad (4.4)$$

Figure 4.6(a) reveals two facts: i) *rrh* increases in both Plexus and *DEWS* routing when the number of destinations increases, ii) *rrh* is always more in DEWS than in Plexus because the experiment was run with 10,000 simultaneous queries and *DEWS* utilized the opportunity of aggregate routing. We computed *rrh* in *DEWS* for 10K, 20K and 30K simultaneous queries having the same number of destinations. Figure 4.6(b) shows that *rrh* increases when i) number of simultaneous queries in the network increases and ii) number of destinations increases.

(b) **Effectiveness of using soft-links**

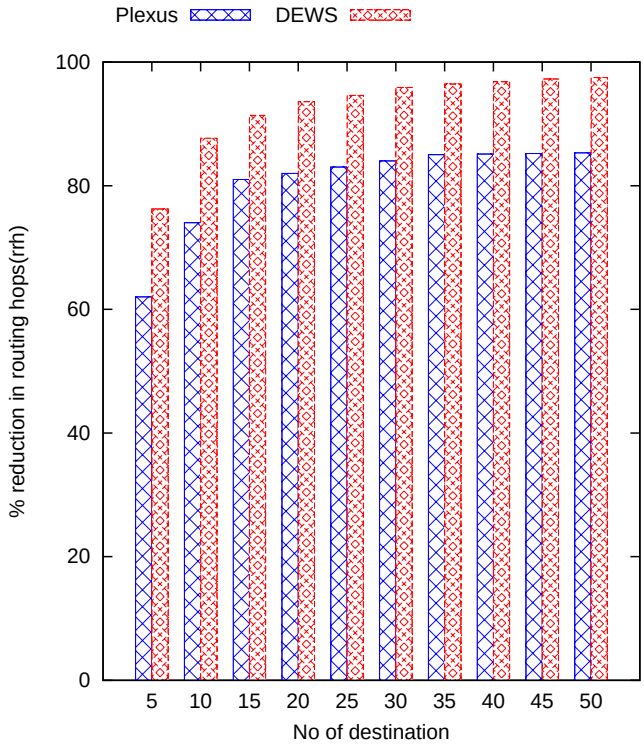
We run an experiment in a network of 50,000 nodes and measure average routing hops with different numbers of simultaneous queries varied from 10,000 to 100,000, which is presented in Figure 4.7. We measured the impact of softlinks on query resolution. The average number of hops required is below 4 without softlinks and below 1 with softlinks. The average number of hops for resolving a query decreases when the number of queries increases because of message aggregation in each hop. It can also be noticed that the number of average hops

is significantly smaller in presence of softlinks. For repeated lookup of target nodes we use softlinks, hence the lower average number of hops per query.

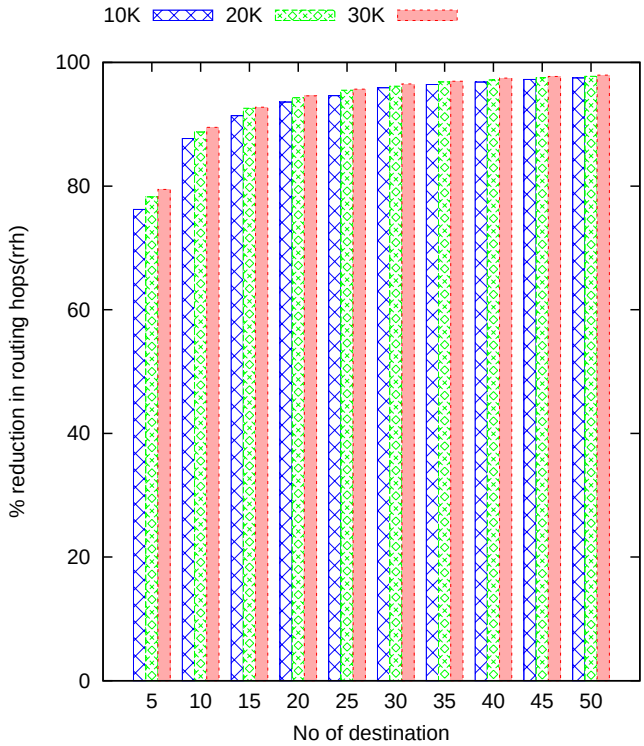
(c) **Network bandwidth**

Figure 4.8(a) presents network bandwidth required for URL advertisement by Plexus routing and *DEWS* 's modified Plexus routing for varied number of URLs from 10,000 to 60,000 in a network of 50,000 nodes. It is observed from the figure that Plexus routing requires around 30 hops where *DEWS* routing requires below 10 hops. Another important observation is the number of hops using *DEWS* routing decreases slightly when the number of URLs increases. The reason is when the number of URLs increases, the average number of outgoing links increases which gives the opportunity of aggregate routing of *URL lookup* messages and reduces the average number of hops.

We have measured the network bandwidth required during keyword indexing by varying the number of keywords from 10,000 to 60,000 in a network of 50,000 nodes, which is presented in Figure 4.8(b). From the figure, it can be observed that number of average hops in Plexus routing remains constant at 15 while number of keywords increases from 10,000 to 60,000. On the other hand, number of average hops in *DEWS* routing decreases gradually from 12 to 5 when the number of simultaneous keyword advertisements increases. The reason is when the number of simultaneous keyword advertisements increases, *DEWS* routing gets the opportunity of aggregation which reduces the average number of hops. We have also measured the scalability of URL and keyword advertisements. We can infer a couple of things from Figure 4.8(c). Firstly, average hops for advertisement do not increase significantly with increased network size. Second, with message aggregation, average hops for both keyword and URL advertisement becomes almost half. And third, URL advertisement requires more hops than keyword advertisement regardless of message aggregation. The reason behind this behavior can be well-explained from Figure 3.5: for advertising a URL, say u_i, we have to lookup $\beta(u_{it})$ for each out link of u_i, while advertising the keywords in $\mathcal{K}_i^{rep}$ we lookup $\beta(u_i)$ once and use it for every keyword $k_{ij}^{rep} \in \mathcal{K}_i^{rep}$. The average number of routing hops for query resolution does not increase significantly when the network size increases as depicted in Figure 4.8(c).



(a) *rrh* in Plexus and *DEWS* routing



(b) *rrh* in *DEWS* routing for simultaneous queries

Figure 4.6: Performance of modified Plexus routing

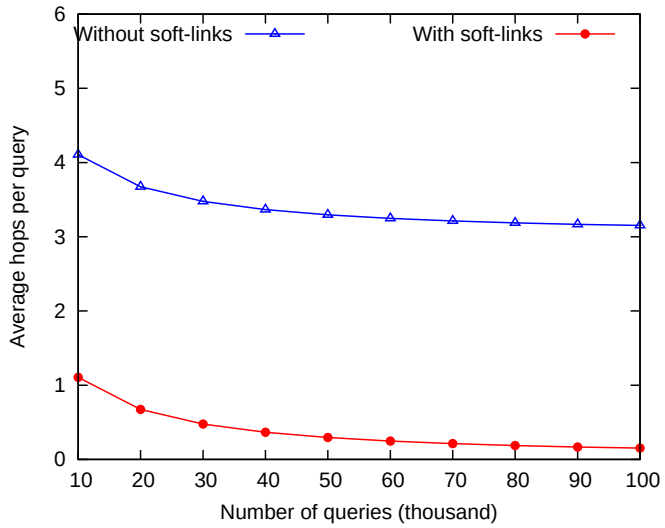
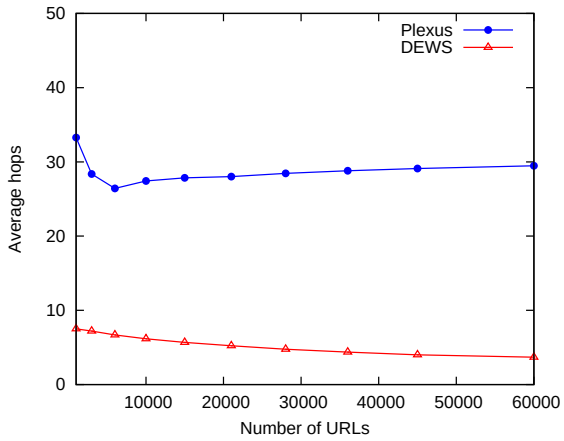
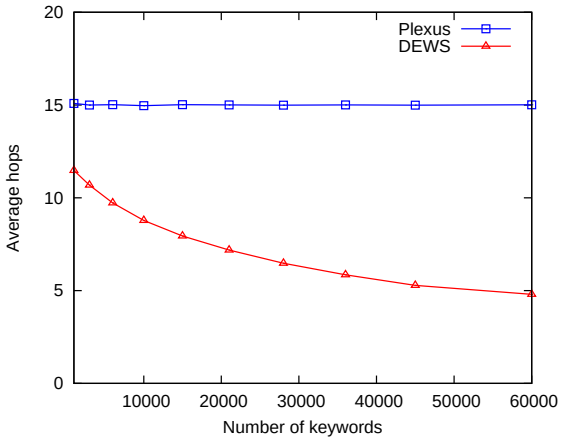


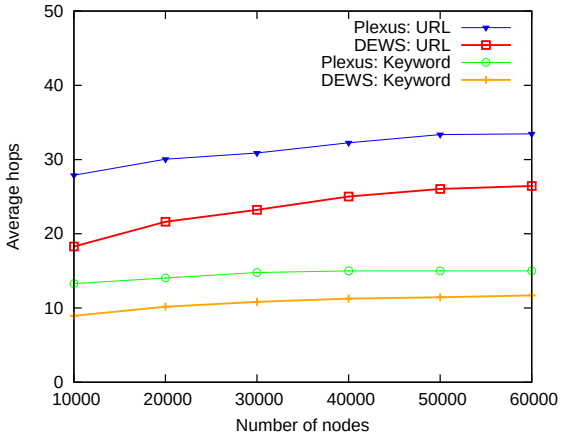
Figure 4.7: Network bandwidth for query resolution



(a) URL indexing



(b) Keyword indexing

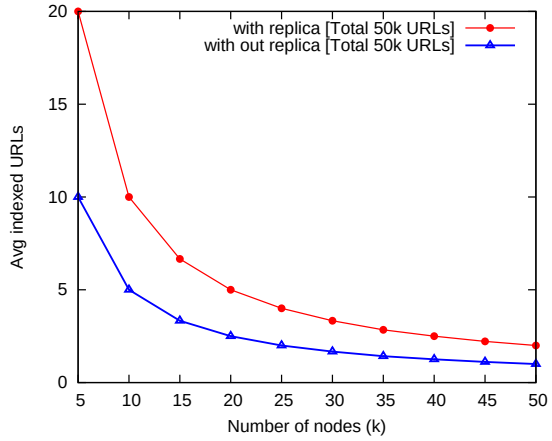


(c) Scalability

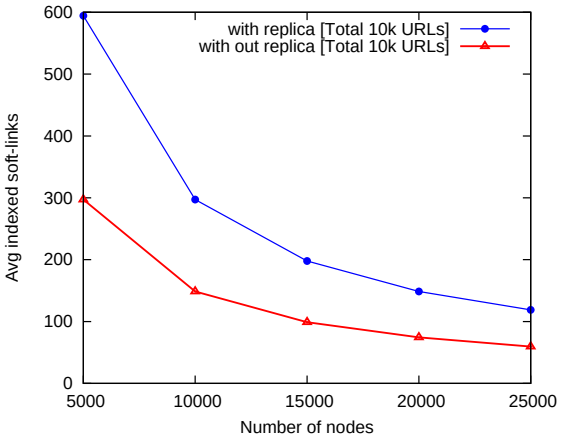
Figure 4.8: Network bandwidth for website indexing

4.4.4 Indexing performance

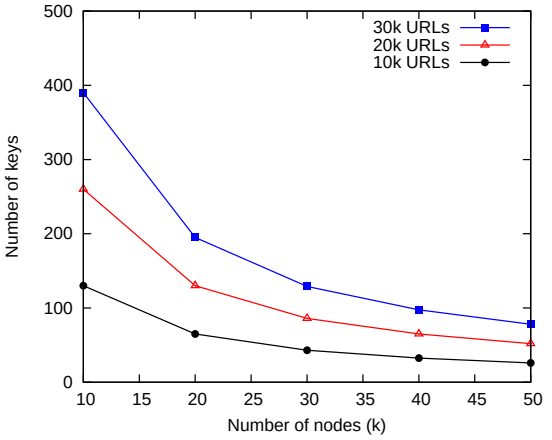
We present average number of indexed URLs, softlinks and keywords per node in Figures 4.9(a), 4.9(b), and 4.9(c), respectively in varied network sizes from 10,000 to 50,000 nodes. It is evident from these figures that the average number of indexed URLs, indexed softlinks and keywords decreases linearly when the network size increases. It should be noted that the number of indexed URLs, softlinks and keywords becomes almost double in presence of replication. The reason behind this behavior can be explained from the replication policy in Plexus, where the node responsible for codeword c_k maintains a replica of its indexes to the node responsible for codeword $\overline{c_k}$, where $\overline{c_k}$ is the bit-wise complement codeword of c_k.



(a) URL indexing



(b) Soft-link indexing

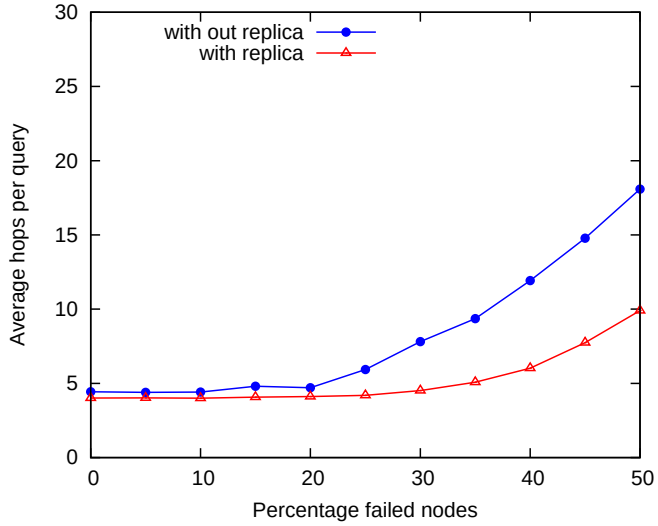


(c) Keyword indexing

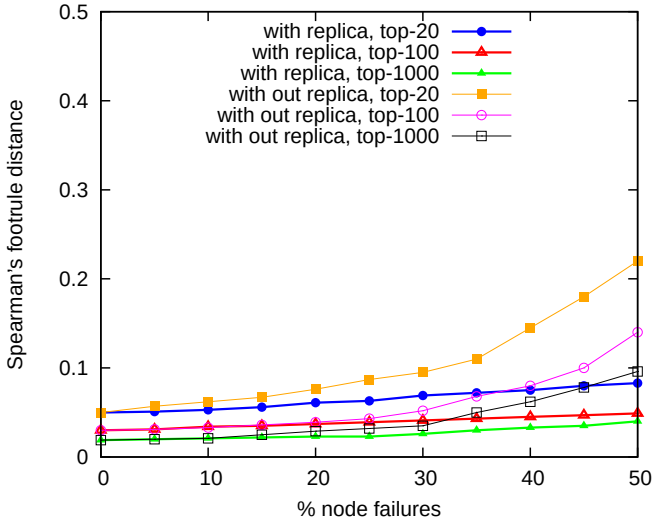
Figure 4.9: Indexing overhead

4.4.5 Fault resilience

DEWS has to work on a continuously changing overlay topology as new webservers can join DEWS network and existing webservers may fail. We used the built-in abilities of Plexus routing for alternate route selection and replication in DEWS to achieve increased failure resilience. We investigated the impact of failure on query routing performance (Figure 4.10(a)) and PageRank accuracy (Figure 4.10(b)) in a DEWS network of 50,000 nodes.



(a) Query resolution



(b) Accuracy of PageRank

Figure 4.10: Fault resilient

We can get a number of insights from these two significant graphs. Firstly, rout-

ing performance and ranking accuracy in DEWS do not degrade when the failure rate is below 30%. Second, average number of hops for query resolution increases as the percentage of failed nodes increases (see Figure 4.10(a)). In presence of node failures, Plexus routes queries through alternate routing paths and possibility of message aggregation decreases, hence the increase in number of routing hops. Third, the average number of hops per query is greater in the no-replication case than the replication case. When we use Plexus replication scheme, we can resolve a query either at a target node or its replica. This feature allows us to route a query to a target node or its replica, whichever is closer in terms of network hops, hence the reduction in query resolution hops. Finally, the Spearman's footrule distance is lower in presence of replication. Without replication, the URLs indexed at the failed nodes remain missing from search result and Spearman's footrule distance increases accordingly.

4.5 Summary

This chapter presented simulation results and evaluation of *DEWS*. We defined few performance metrics including search flexibility and accuracy, accuracy of ranking, routing efficiency, network bandwidth, scalability and fault tolerance. We used a cycle driven simulator written in Java. LETOR 3.0 dataset was used to drive our simulation. We found that *DEWS* provides flexible searching with high accuracy. For example, 87% recall and 76% precision rates were attained in presence of query keywords having edit distance three from the advertised keywords. This high search performance is attained by combining phonetic encoding and n-gram segregation with the list decoding mechanism. We also varied the list decoding radius and found that search accuracy increases slightly and network bandwidth increases linearly when the radius increases. For this reason, *DEWS* searches with a smaller radius first and larger radius later if users are not satisfied with the first set of results. This incremental retrieval technique saves network bandwidth. *DEWS* computes PageRank and BM25 in a distributed manner. We found that the accuracy of PageRank and BM25 are very close to the centrally computed values. We also found that PageRank algorithm converged within 60 cycles. It required less than 20 average messages per node. We measured the performance of the modified Plexus routing protocol, which required less network bandwidth for simultaneous advertisements and query resolution than the unmodified Plexus routing mecha-

nism. From the experimental results, we found that indexing overhead for URL, keywords and soft-links were small and indexes were uniformly distributed over all the webservers. We varied the percentage of failed nodes and found that routing performance and PageRank accuracy did not degrade considerably upto 30% failure level.

Chapter 5

Conclusion and Future Research

Exponential growth rate of number of websites stresses the demand for a decentralized Web search engine. Existing distributed approaches do not provide all the desirable properties of such an engine. In this thesis, we have presented *DEWS* which meets all the desirable properties of a decentralized Web search engine.

We summarize the contributions of this thesis in Section 5.1 and give concluding remarks in Section 5.2. Finally, Section 5.3 presents some future research directions.

5.1 Summary of Contributions

The contributions of this thesis can be summarized as follows:

- We presented a novel technique for enabling the Web to index itself where webservers collaboratively create distributed index of webpages and respond to user queries.
- Our presented framework supports approximate keyword matching and complete ranking of webpages in a distributed manner without incurring significant network or storage overheads.
- We presented a new route aggregation protocol that extends the original Plexus routing protocol, which significantly reduces the number of routing messages in the network.
- We also presented a distributed incremental retrieval technique that allows a user to limit his/her search to a small number of nodes.

5.2 Thesis Summary and Concluding Remarks

In this thesis, we have presented *DEWS*- a self-indexing architecture for the Web. *DEWS* enables the webservers to collaboratively index the Web and respond to Web queries in a completely decentralized manner. It uses a structured overlay based on the second order Reed-Muller code. We adopted the concept of Double Metaphone encoding for minimizing Hamming distance between the advertisement and query patterns, which eventually decreases the effect of edit distance between the advertised and queried keywords. *DEWS* utilizes Double Metaphone and n-grams with list decoding of Reed-Muller code to allow flexible search with partially specified misspelled keywords. We also proposed an incremental retrieval technique varying the list decoding radius to reduce network bandwidth consumption. We preserved a hyperlink overlay on top of a structured overlay to compute PageRank in a distributed manner having accuracy closer to the centralized computation. We computed keyword relevance considering the structure of website, webpage, position of keywords in the webpages in a distributed manner with high accuracy unlike existing approaches. The route aggregation technique proposed in this work outperforms the original Plexus routing protocol in terms of network usage efficiency. Using the extended routing protocol, *DEWS* indexes websites and search queries with low network overhead. *DEWS* maintains replica of indexes which makes it robust to webserver failures.

We evaluated the concepts presented in this thesis through simulations results. We used LETOR dataset to validate our ranking algorithms. As demonstrated by the simulation results, for a network of about 50,000 nodes and 10,000 URLs, PageRank in *DEWS* converged within 120 cycles where Spearman's footrule distance became 0.11 to the centrally computed PageRank for top-20 results. *DEWS* required low network bandwidth for URL and keyword indexing as well as query lookup which was around five logical hops on average. Experimental results showed that in *DEWS*, each node indexed 5 URLs on average in a network of 10,000 nodes when 50,000 URLs were indexed. We also found that around 400 keywords on average were indexed by each node in a network of 10,000 nodes when 30,000 URLs were advertised. In experimental results, Spearman's footrule distance was around 0.075 for top-20 results between BM25 scores computed in *DEWS* and centrally from the dataset. Beyond computation of efficient ranking of websites, another major contribution of *DEWS* is to provide flexible searching with high accuracy. Experimental results showed that *DEWS* achieved around 100%, 98%, and 87% recalls

for query keywords with one, two and three edit distances, respectively. Precisions for query keywords with one, two and three edit distances were 92%, 88%, and 76%, respectively. Simulations results also showed that DEWS is highly resilient to node failures due to the existence of alternate routing paths and smart replication policy. Neither routing efficiency nor ranking accuracy degrades significantly even in presence of 30% failures.

Based on the simulation results, we can conclude that *DEWS* is a complete decentralized Web search engine which provides flexible search with high accuracy of search results and ranking.

5.3 Future Directions

In the following, we outline few directions for future research:

- We used hyper-link overlay on top of a structured P2P overlay named *Plexus* to rank web search results using PageRank and keyword relevance (e.g., BM25) computed in a distributed manner. It would be interesting to use multi-level *Plexus* overlay by combining 1^{st} and 2^{nd} order Reed-Muller codes representing super peers and regular peers, respectively. A set of search topics can be predefined and assigned to the super peers which will facilitate the searching and ranking mechanism.

- Introducing a semantic overlay on top of *Plexus* overlay would be interesting to allow semantic search. In this case, the challenge is to identify ontological mapping of keywords. Recent developments of ontological mappings in various areas such as health informatics show promise toward efficient semantic search.

- We used a hyper-link overlay on top of *Plexus* overlay to compute PageRank in a distributed manner. We hashed an URL and found a node to index it. We used softlink to preserve link between two nodes (say, n_1 and n_2) indexing linked URLs (say, u_i and u_j). It would be interesting to index two linked URLs u_i and u_j in two nodes n_x and n_y which are neighbors to each other. If it is possible to preserve the proximity of URLs in the hyperlink overlay on *Plexus* overlay, it would require less network bandwidth and convergence time during PageRank computation.

- We used centrally computed PageRank to compare PageRank computed in *DEWS* and LETOR3.0 dataset to compare BM25 as keyword relevance. We

did not compare DEWS combining both PageRank and keyword relevance (incorporating many factors including structure of website and webpages) with any established search engine. It would be interesting to establish a dataset by crawling the Web to validate web search results in *DEWS* and compare *DEWS* with other established search engines.

Bibliography

[1] The Gnutella website, http://www.gnutella.com. 17

[2] http://en.wikipedia.org/wiki/levenshtein_distance. 25

[3] http://en.wikipedia.org/wiki/okapi_bm25. 25

[4] http://en.wikipedia.org/wiki/yacy. 18

[5] http://www.faroo.com/. 18

[6] http://www.vaughns-1-pagers.com/internet/google-ranking-factors.htm/. 7

[7] Yacy website: http://yacy.net/en/. 17

[8] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: a self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3):29–33, 2003. 17

[9] K. Aberer and J. Wu. A framework for decentralized ranking in web information retrieval. *Web Technologies and Applications*, pages 594–594, 2003. 14

[10] R. Ahmed and R. Boutaba. Plexus: A scalable peer-to-peer protocol enabling efficient subset search. *Networking, IEEE/ACM Transactions on*, 17(1):130–143, 2009. 20, 22, 30, 33

[11] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative p2p search. In *Proceedings of the 31st international conference on Very large data bases*, pages 1263–1266. VLDB Endowment, 2005. 15

[12] K. Bharat and G. Mihaila. Hilltop: A search engine based on expert documents. In *Proc. of the 9th International WWW Conference (Poster)*, 2000. 10

[13] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, 1970. 23

[14] M. Bouklit and F. Mathieu. Backrank: an alternative for pagerank? In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1122–1123. ACM, 2005. 10

[15] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998. 11

[16] A. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient pagerank approximation via graph aggregation. *Information Retrieval*, 9(2):123–138, 2006. 10

[17] C. L. K. P. C. R. D.G. Hoffman, D.A. Leonard and J. Wall. Coding Theory;The Essentials. 18

[18] I. Dumer, G. Kabatiansky, and C. Tavernier. List decoding of reed-muller codes up to the johnson bound with almost linear complexity. In *Information Theory, 2006 IEEE International Symposium on*, pages 138–142. IEEE, 2006. 20

[19] N. Eiron, K. McCurley, and J. Tomlin. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web*, pages 309–318. ACM, 2004. 10

[20] P. Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985. 15

[21] R. Fourquet and C. Tavernier. An improved list decoding algorithm for the second order reed–muller codes and its applications. *Designs, Codes and Cryptography*, 49(1):323–340, 2008. 20

[22] D. Gleich, L. Zhukov, and P. Berkhin. Fast parallel pagerank: A linear system approach. *Yahoo! Research Technical Report YRL-2004*, 38, 2004. 10

[23] P. Gopalan, A. Klivans, and D. Zuckerman. List-decoding reed-muller codes over small fields. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 265–274. ACM, 2008. 20

[24] H. Gylfason, O. Khan, and G. Schoenebeck. Chora: Expert-based p2p web search. *Agents and Peer-to-Peer Computing*, pages 74–85, 2008. 16

[25] T. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, pages 784–796, 2003. 9

[26] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003. 9

[27] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank. Technical report, Citeseer, 2003. 10

[28] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999. 10, 11, 38

[29] C. Kohlschutter, P. Chirita, and W. Nejdl. Efficient parallel computation of pagerank. *Advances in information retrieval*, pages 241–252, 2006. 10

[30] R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Transactions on Information Systems (TOIS)*, 19(2):131–160, 2001. 11

[31] T. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pages 3–10, 2007. 1, 50

[32] Z. Lu, B. Ling, W. Qian, W. S. Ng, and A. Zhou. A distributed ranking strategy in peer-to-peer based information retrieval systems. In *APWeb'04*, pages 279–284, 2004. 44

[33] T. Luu, F. Klemm, I. Podnar, M. Rajman, and K. Aberer. Alvis peers: a scalable full-text peer-to-peer retrieval engine. In *Proceedings of the international workshop on Information retrieval in peer-to-peer networks*, pages 41–48. ACM, 2006. 17

[34] B. Manaskasemsak and A. Rungsawang. Parallel pagerank computation on a gigabit pc cluster. In *Advanced Information Networking and Applications,*

2004. AINA 2004. 18th International Conference on, volume 1, pages 273–277. IEEE, 2004. 10

[35] F. Mathieu and M. Bouklit. The effect of the back button in a random walk: application for pagerank. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 370–371. ACM, 2004. 10

[36] S. Michel, P. Triantafillou, and G. Weikum. Klee: a framework for distributed top-k query algorithms. In *Proceedings of the 31st international conference on Very large data bases*, pages 637–648. VLDB Endowment, 2005. 15

[37] D. Nemirovsky. Weighted pagerank: cluster-related weights. Technical report, DTIC Document, 2008. 10

[38] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Technical Report, Stanford Digital Library Technologies Project, 1999. 9, 38, 40, 44

[39] J. Parreira, D. Donato, S. Michel, and G. Weikum. Efficient and decentralized pagerank approximation in a peer-to-peer web search network. In *Proceedings of the 32nd international conference on Very large data bases*, pages 415–426. VLDB Endowment, 2006. 13

[40] R. Pellikaan and X. Wu. List decoding of q-ary reed-muller codes. *Information Theory, IEEE Transactions on*, 50(4):679–682, 2004. 20

[41] O. D. Project. http://dmoz.org. 9

[42] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: a public dht service and its uses. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 73–84. ACM, 2005. 16

[43] K. Sankaralingam, S. Sethumadhavan, and J. Browne. Distributed pagerank for p2p systems. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 58–68. IEEE, 2003. 11

[44] S. Shi, J. Yu, G. Yang, and D. Wang. Distributed page ranking in structured p2p networks. In *ICPP*, pages 179–186. IEEE Computer Society, 2003. 12, 13

[45] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable Peer-to-Peer lookup protocol for Internet applications. *IEEE/ACM Transaction on Networking (TON)*, 11(1):17–32, 2003. 15

[46] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In *Proceedings of the International Workshop on the Web and Databases*, volume 49, 2003. 15

[47] Y. Wang and D. DeWitt. Computing pagerank in a distributed internet search system. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 420–431. VLDB Endowment, 2004. 14

[48] J. Wu. Towards a decentralized search architecture for the web and p2p systems. In *Proceedings of the Adaptive Hypermedia and Adaptive Web-Based Systems Workshop held at the HyperText03 Conference*, 2003. 14

[49] J. Wu and K. Aberer. Using siterank for p2p web retrieval. In *Technical report, Swiss Fed. Inst. of Tech.* Citeseer, 2004. 14, 35

[50] J. Zhou, K. Li, and L. Tang. Towards a fully distributed p2p web search engine. In *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pages 332–338. IEEE, 2004. 16