DEWS: A Decentralized Engine for Web Search

Reaz Ahmed, Rakibul Haque, Md. Faizul Bari, Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
[r5ahmed|m9haque|mfbari|rboutaba]@uwaterloo.ca

Bertrand Mathieu
Orange Labs, Lannion, France
bertrand2.mathieu@orange-ftgroup.com

(Technical Report: CS-2012-17)

Abstract

The way we explore the Web is largely governed by centrally controlled, clustered search engines, which is not healthy for our freedom in the Internet. A better solution is to enable the Web to index itself in a decentralized manner. In this work we propose a decentralized Web search mechanism, named DEWS, which will enable the 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. DEWS also supports approximate matching of query keywords and incremental retrieval of search results in a decentralized manner. We use the standard LETOR 3.0 dataset to validate the DEWS protocol. Simulation results show that the ranking accuracy of DEWS is very close to the centralized case, while network overhead for collaborative search and indexing is logarithmic on network size. Simulation results also show that DEWS is resilient to changes in the available pool of indexing webservers and works efficiently even in presence of heavy query load.

1 Introduction

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 (*http://news.netcraft.com/*) Web Server Survey, around 18 million websites were added to the Internet in October 2011 making the total to 504.08 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 and the requirement for unbiased search results. Evidently the 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.

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 inherently challenging in any large scale distributed system. Moreover, incremental retrieval of search results in a distributed manner is essential for conserving valuable network bandwidth.

Distributed Hash Table (DHT) based systems [14, 17, 22] offer efficient indexing and lookup of information in a distributed manner, yet they does not natively support approximate matching of query keywords to advertised documents. On the other hand, distributed ranking techniques proposed in existing research works [6, 12, 21] compute approximate ranking of search results based on partial information available locally to each node.

In this paper we propose DEWS, an efficient Web search framework with distributed index and decentralized control. DEWS offers approximate matching of query keywords, distributed ranking, and incremental retrieval of search results, while ensuring efficient network bandwidth usage. We use Plexus [1] for routing and indexing in DEWS. Major contributions of this work (Section 3) 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. This approach significantly reduces routing overhead.
- We also propose a distributed incremental retrieval technique that allows a user to limit his/her search to a small number of nodes while retrieving the most relevant results. A user can progressively query additional nodes for additional results.

The concepts presented in this work have been validated through extensive simulations (Section 4). We used the standard LETOR 3.0 dataset to drive the input of our simulation. Finally, we present the conclusion and future research directions in Section 5.

2 Related Works

Link structure analysis is a very popular technique for ranking search results. Google uses the PageRank [11] algorithm to compute weights of crawled web pages that measure the authority-ship of these pages. Following the success of Google, link structure analysis became very popular among the research communities and a huge number of such algorithms can be found in the literature.

Following centralized systems, most of the distributed ranking approaches used the link structure analysis, specially, PageRank [11]. Sankaralingam *et al.* proposed a distributed PageRank algorithm in [15] for documents ranking in P2P networks. In their proposed approach, every peer initializes a PageRank score to their hosted documents and sends update messages to adjacent peers. Whenever a peer receives an update message, it updates PagaRank scores for its documents based on the hyper-link structure and propagates update messages to adjacent peers. The PageRank scores converge after a several hundred cycles. DynaRank [6] computes and updates PageRank values similar to [15]. Instead of updating the whole web like [15], DynaRank propagates update messages to the connected peers, only when the magnitude of weight change is greater than a threshold value. Shi *et al.* proposed Open System PageRank in [16], where the whole Web is partitioned into k groups (called "page groups") and these groups are assigned to k rankers. Each ranker runs standard PageRank algorithm on the assigned pages. If one page group has links to other "page groups", responsible ranker updates PageRank scores by periodically communicating with the rankers responsible for the linked "page groups". In JXP [12], each peer computes initial weights for their local pages using the standard PageRank algorithm [11] and introduces the notion of "external world", which is a node representing the outgoing and incoming links from a peer. Each time a peer meets with another peer, it updates knowledge about its external world. After a several hundreds of peer meetings, each peer acquires the global knowledge and score for each page. Wang *et al.* used two types of ranks for overall ranking: "local PageRank" computed in each peer based on the standard Pagerank [11] and "ServerRank" computed as the highest "local PageRank" or as sum of all the PageRank of a web server [19]. SiteRank [20] computes the rank at the granulaity level of web sites instead of web page level using PageRank [11]. Distributed PageRank computation based on Iterative aggregation-disaggregation method is proposed in [23]. Wu *et al.* proposed a layered Markov model for distributed ranking where links between web sites are in the higher layers and links between the web pages within a particular web site or domain are in the lower layers [21].

Another research trend is to use Information Retrieval techniques such as VSM (Vector Space Model), which is widely used in centralized ranking systems. However, computing global weight (inverse document frequency or *idf*)

in distributed systems is a challenging issue. A random sampling technique is used in [5] to compute approximate value of idf. In a DHT-based structured network, each keyword is mapped to a particular peer and that peer can compute the approximate value of idf [9]. A Gossip-based algorithm is proposed in [10] to approximate both term frequency (tf) and idf for unstructured P2P networks. In Minerva [2], a hash-sketch [4] based technique is used to approximate global document frequencies. ODISSEA [18] employs only term frequency for ranking search results. YaCy[1] does not have efficient ranking mechanism. Faroo[2] adopts user actions including most frequently viewed pages, and bookmarks without using keyword relevance and PageRank.

Existing distributed web search techniques do not use structural information and compute ranks based on the incomplete information. In this work, both the link structure weights and keyword relevance are computed utilizing complete information obtained in a distributed manner. As these computations are performed using a complete information, computed results closely match the centrally computed values.

3 System Architecture

We have combined a few techniques in DEWS. We present the network and indexing architectures in Sections 3.1 and 3.2, respectively. The rank computation process is explained in Section 3.3, while the query resolution and incremental retrieval mechanisms are presented in Section 3.4.

3.1 Network Architecture

3.1.1 Plexus Routing

We organize the Web servers into a structured overlay network, since DHT-based solutions have been proven to be efficient in information lookup in very large networks. In addition to efficient lookup we need to perform approximate matching between query keywords and webpage keywords. Like other DHT techniques Plexus supports efficient routing. In addition, support for approximate matching is built into the Plexus routing mechanism, which is not easily achievable by other DHT techniques. Plexus delivers a high level of fault-resilience by using replication and redundant routing paths. For these advantages we have chosen Plexus routing mechanism for DEWS.

We have extended Plexus routing mechanism for enabling distributed ranking and incremental retrieval of search results. Plexus support Hamming distance based matching of bit-vector or patterns. For each representative keywords (as presented in LETOR dataset) from a website, we construct a Bloom filter [3] by hashing the n-grams from the keyword and its Double Metaphone string. Then we use Plexus to index these Bloom filters along with the website's meta-information.

Plexus routing is based on the theory of linear binary codes. Each node in Plexus takes responsibility of one (or more) codeword(s) and indexes all advertised patterns within certain Hamming distance from that codeword(s). Each node maintains $O(\log n)$ links and guarantees message routing between any pair of nodes in $O(\log n)$ hops. We use $RM(2, 6)$ Reed Muller code for implementing Plexus routing in DEWS.

3.1.2 Route Aggregation

Besides offering approximate matching of query keywords and efficient routing, Plexus has the inherent capability of path aggregation for multicast routing. We extend the path aggregation capability of Plexus from single source multicasting to the multi-source case. 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 multicasting, 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.

[1]YacY-http://www.yacy.net/
[2]Faroo-http://www.faroo.com/

Algorithm 1 *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
$\mathcal{Y}_m \leftarrow \bigcup_{m \in msgQ} m.\mathcal{Y}$
{*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

Algorithm 1 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 a very small period of time. Target codeword lists ($m.\mathcal{Y}$) 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, as demonstrated in Section 4.2

3.2 Indexing Architecture

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 [11], HITS [7] *etc.*, are used for computing weights or relative significance of each URL. In DEWS, we use both of these measures for ranking search results.

3.2.1 Hyperlink Overlay

About 90% hyperlinks in the Web are intra-domain [20]. Topics and ideas in the webpages of a particular website are almost similar or correlated and it is not reasonable to utilize the anuthorship 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 hyperlinks to the rest of the Web [20]. The number of websites in the Web is 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 paper, we use "URL" to refer to the root URL of a website.

Algorithms for computing URL weights based on hyperlink structure are iterative and require many iterations to converge. In each iteration URL weights are updated and the new weights are propagated to adjacent URLs for computation in the 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 hyperlinked URLs are mapped to the 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 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 $\beta(u_{it})$. Hence we cache the network address of node $\beta(u_{it})$ at node $\beta(u_i)$, which we call a soft-link. Soft-links mitigate the network overhead generated from repeated lookups. The process of mapping the hyperlink overly over a Plexus overlay is explained in Fig. 1.

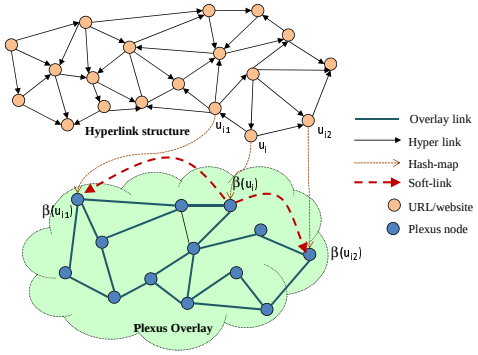


Figure 1: Hyperlinks to Plexus overlay mapping

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 $\beta(u_i)$.

3.2.2 Indexing Keywords

We use Plexus to build an inverted index on the important keywords (as extracted from LETOR 3.0 dataset) for each website. This dataset also contains the BM25 scores for these keywords. 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.

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 k_{ij}^{dmp} by applying Double Metaphone encoding [13] on k_{ij}^{rep}. Double Metaphone encoding attempts to detect phonetic ('sounds-alike') relationship between words. A typical Double Metaphone key is upto four characters long, as this tends to produce the ideal balance between specificity and generality of results. For example, Double Metaphone encoding will produce 'NLSN' for keywords 'Nelson' and 'Nilsen', while 'Adam' will be encoded as 'ATM', which has no resemblance to neither 'Nelson' nor 'Nilson'.

Motivation behind adapting phonetic encoding is twofold: i) any two phonetically equal keywords have no edit distance between them, ii) phonetically inequivalent keywords have less edit distance than edit distance between the original keywords. In both cases, Hamming distance between encoded advertisement and search patterns is lesser than that of the patterns generated from 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.

To generate advertisement or query pattern P_{ij} from keyword k_{ij}^{rep}, we fragment k_{ij}^{rep} into k-grams ($\{k_{ij}^{rep}\}$) and encode these k-grams along with k_{ij}^{dmp} into an n-bit Bloom filter. We use this Bloom filter as a pattern P_{ij} in $\mathbb{F}_2^n$ and *list decode*[3] 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 on k_{ij}^{rep} at the nodes responsible for codewords in $\zeta_\rho(P_{ij})$. The index for k_{ij}^{rep} is a quadruple $< k_{ij}^{rep}, r_{ij}, u_i, \beta(u_i) >$, where r_{ij} is a measure

[3]List decoding is the process of finding all the codewords within a given Hamming distance (list decoding radius, e) from a (advertisement or query) pattern

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, $\gamma(k_{ij}^{rep}) \equiv lookup(\zeta_\rho(BF(\{k_{ij}^{rep}\} \cup \{k_{ij}^{dmp}\})))$, $BF(\cdot)$ represents Bloom filter encoding function.

3.2.3 Advertising Websites

The pseudocode for adversing a website is presented in Algorithm 2. As discussed in the previous two sections, 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 2, 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 18, we compute the indexes on $\mathcal{K}_i^{rep}$ and advertise the indexes to the responsible nodes.

Algorithm 2 *Publish website*

```
1: Inputs:
       u_i: URL of the website to be advertised
2: Functions:
       ħ(u_i): hash map u_i to a codeword
       γ_r(P): {c_k | c_k ∈ C ∧ δ(P, c_k) ≤ r}
       lookup(c_k): Finds the node that stores c_k
3: β(u_i) ← lookup(ħ(u_i))
4: for all out-link u_it of {u_i} do
5:     β(u_it) ← lookup(ħ(u_it)))
6: end for
7: w_i ← initial PageRank of u_i
8: store < u_i, w_i, {u_it, β(u_it)} > to node β(u_i)
9: K_i^rep ← set of representative keywords of u_i
10: for all k_ij^rep in K_i^rep do
11:     k_ij^dmp ← DoubleMetaphoneEncode(k_ij^rep)
12:     P_ij ← BloomFilterEncode({k_ij^rep} ∪ {k_ij^dmp})
13:     r_ij ← relevance of k_ij^rep to u_i
14:     for all c_k in ζ_ρ(P_ij) do
15:         v ← lookup(c_k)
16:         store < k_ij^rep, r_ij, u_i, β(u_i) > to node v
17:     end for
18: end for
```

3.3 Weight Computation

3.3.1 Keyword Relevance Weight

We use Vector Space Model (VSM) for computing relevance of keyword k_{ij}^{rep} to URL u_i. In VSM, each URL u_i is represented as a vector $\vec{\nu_i} = (r_{i1}, \ldots, r_{ig})$, where r_{ij} represents the relevance of the term or keyword k_{ij}^{rep} in u_i, and g is the number of representative keywords in u_i. The relevance weight r_{ij}, of the j^{th} keyword is computed as $tf(k_{ij}^{rep}) * idf(k_{ij}^{rep})$. Here, term frequency $tf(k_{ij}^{rep})$ is the number of occurrences of k_{ij}^{rep} in website u_i, while inverse document frequency $idf(k_{ij}^{rep})$ is computed as follows:

$$idf(k_{ij}^{rep}) = \log \frac{U}{\psi(k_{ij}^{rep})} \tag{1}$$

Here, U is the total number of websites and $\psi(k_{ij}^{rep})$ is the number of websites containing 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 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 the purely decentralized setup. Instead we use the total number of indexed URLs in a node in place of U as advocated in [9].

3.3.2 Hyper-Link Weight

For ranking search results, we have adapted the original PageRank [11] 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 2). The PageRank computation equation for each website is as follows:

$$w_i = (1-\eta) + \eta \sum_{t=1}^{g} \frac{w_{it}}{L(u_{it})} \tag{2}$$

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 by u_i and $L(u_{it})$ is the number of outgoing links from website u_{it}.

Each node periodically executes Algorithm 3 to maintain the PageRank weights updated in a distributed manner. To communicate PageRank information between the nodes, we use a PageRank update 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 query 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 2. 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 u_{it}.

Algorithm 3 *Update PageRank*

1: 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
2: **for all** URL u_i indexed in this node $\beta(u_i)$ **do**
3: $temp \leftarrow 0$
4: **for all** $< u_{si}, u_i, \frac{w_{si}}{L(u_{si})} > \in \mathcal{Q}_{u_i}$ **do**
5: $temp \leftarrow temp + \frac{w_{si}}{L(u_{si})}$
6: **end for**
7: $w_i^{new} \leftarrow (1-\eta) + \eta * temp$
8: **if** $|w_i^{new} - w_i| > \theta$ **then**
9: $w_i \leftarrow w_i^{new}$
10: **for all** out link u_{it} from u_i **do**
11: send PageRank message $< u_i, u_{it}, \frac{w_i}{L(u_i)} >$ to $\beta(u_{it})$
12: **end for**
13: **end if**
14: **end for**

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 weight, w_i to 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 Fig. 2.

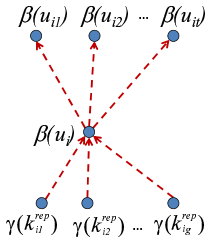


Figure 2: Softlink structure in DEWS

3.4 Resolving Web Query

3.4.1 Search and Ranking

Most frequent queries in the Web consists of 3 to 5 keywords. To resolve Web queries in DEWS, we breakdown the query into subqueries each consisting a single query keyword, say q_l. Similar to the keyword advertisement process explained in Section 3.2.2, we compute the Double Metaphone (*i.e.*, q_l^{dmp}), k-gram ($\{q_l\}$), 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 $\{< u_i, w_i, r_{il} >\}$, 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.*, r_{il}. Now, the querying node computes the ranks of the extracted URLs using the following equation:

$$rank(u_i) = \sum_{q_l} \sum_{u_i} \vartheta_{il}(\mu \cdot w_i + (1-\mu) \cdot r_{il}) \tag{3}$$

In Equation 3, μ is a weight adjustment factor governing the relative importance of link structure weight (w_i) and semantic relevance (r_{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.

Algorithm 4 *Query*

```
Input:
    Q: set of query keywords {q_l}
    T: Most relevant T websites requested
Internals:
    μ: Weight adjustment on link-structure vs relevance
    ρ: list decoding radius
ξ ← empty associative array
for all q_l ∈ Q do
  q_l^dmp ← DoubleMetaphoneEncode(q_l)
  P_l ← BloomFilterEncode({q_l} ∪ {q_l^dmp})
  for all c_k ∈ listDecode_ρ(P_l) do
    n ← lookup(c_k)
    for all {< u_i, w_i, r_il >} ∈ n.retrive(q_l) do
      ξ[u_i].value ← ξ[u_i].value + μ · w_i + (1 − μ) · r_il
    end for
  end for
end for
sort ξ based on value
return top T u_i from ξ
```

The query process in DEWS is explained in Algorithm 4. In this algorithm, we need 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 3.1.2.

3.4.2 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.

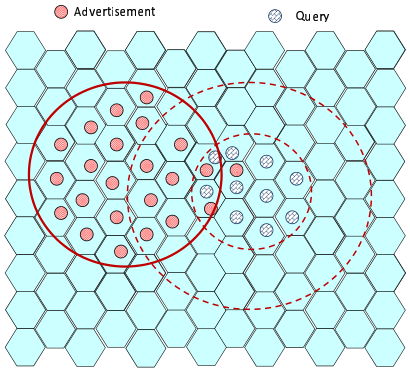


Figure 3: Incremental search

We have exploited the Hamming distance based lookup capability of Plexus to enable distributed incremental retrieval in DEWS. In Plexus search mechanism, list decoding radius ρ can be varied to control the Hamming distance of the discovered advertisement patterns from a query pattern. Since the edit distance between query and advertisement keywords is proportional to the Hamming distance of the corresponding query and advertisement patterns, we can discover the documents having lesser similarity to the query keywords by gradually increasing ρ.

As explained in Fig. 3, we gradually explore the nodes near a query pattern in steps. For the initial step, we use a small 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 in subsequent steps, 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 step. For most of the cases, desired number of results can be found in the first step, which can save a lot of network bandwidth as demonstrated in Section 4.4.2.

Unlike expanding ring search or iterative deepening search in unstructured P2P networks, we can find the target nodes for the next step from the neighbour list of the queried nodes in the current step. Hence, we do not need to query the same set of nodes again and again for consecutive steps.

4 Experiment and evaluation

In this section, we evaluate the performance of DEWS based on the results obtained by implementing the DEWS framework in a simulated environment. Performance metrics used for evaluation are routing hops, indexing overhead, convergence time, network message overhead, and accuracy of link-structure analysis. For our simulations we have varied the number of URLs, number of queries, number of nodes, number of keywords, edit distance between advertised and query keywords, number of simulation cycles *etc.* to measure the scalability and robustness of DEWS.

4.1 Simulation Setup

We have developed a cycle-driven simulator to evaluate various aspects of DEWS. In each cycle, every node in the simulated network processes their incoming messages and routes to the destinations specified in the messages using the modified Plexus routing algorithm presented in Section 3.1.2.

We have used the "Web Track" dataset from LETOR 3.0 [8] for our experiments. 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.

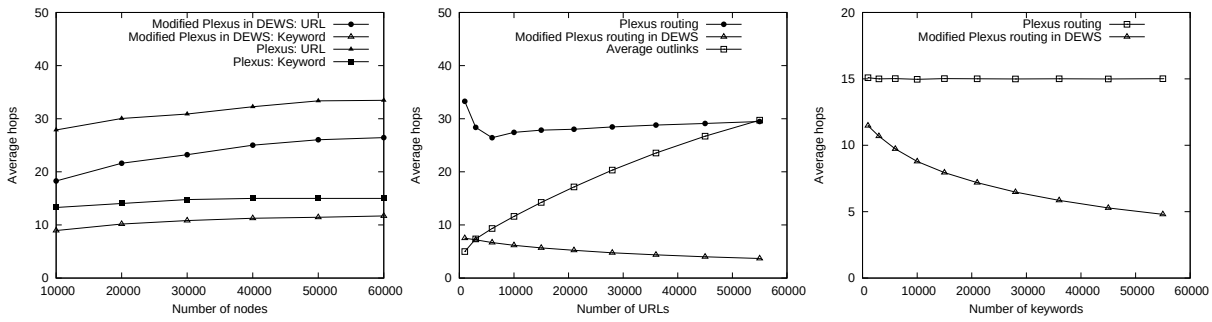
In our experiments, we are using all three categories of search queries and the refined dataset available under "$Gov\backslash Feature\_min$" where NULL or missing values are replaced with feature wise minimum values. For computing the PageRank of the HTML documents in the dataset we are using the "Sitemap" of the .gov collection. Relevant keywords are extracted by parsing the meta-data files associated with each query file available under LETOR 3.0.

4.2 Routing Performance

In this section we evaluate advertisement and indexing behavior in DEWS from four view points: a) scalability of advertisement routing with network size (Section 4.2.1), b) effect of Plexus routing on message aggregation (Section 4.2.2), c) indexing overhead (Section 4.2.3) and d) effectiveness of softlink on query routing performance (Section 4.2.4).

4.2.1 Scalability

Fig. 4(a) presents the average number of routing hops per advertisement for different network sizes. We captured the impact of message aggregation on average routing hops in presence of 1000 simultaneous advertisements. We can infer a couple of things from this curve: 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 behaviour can be well-explained from Fig. 2. 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 $k_{ij}^{rep} \in \mathcal{K}_i^{rep}$.



(a) Advertisement scalability (b) Aggregate routing - URL (c) Aggregate routing - keyword

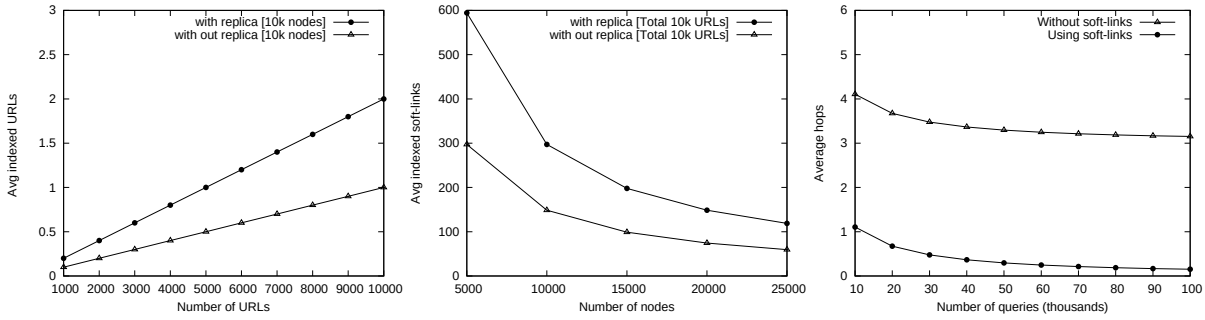
Figure 4: Routing efficiency for advertisement

4.2.2 Message Aggregation

Figures 4(b) and 4(c) present average hops for varying number of simultaneous URL advertisements and keyword advertisements, respectively. We ran this experiment on a network of 50,000 nodes. As expected for the original Plexus routing, average hops per URL or keyword advertisement remain almost constant for a fixed network size. In line with the findings from the previous experiment, average hops reduces significantly in presence of the proposed route aggregation scheme. In addition, for route aggregation scheme average hops decrease as the number of simultaneous advertisements increases, because each node gets more alternatives to combine in the outgoing messages.

4.2.3 Index Overhead

We present average number of indexed URLs and average number of softlinks per nodes in Figures 5(a) and 5(b), respectively. It is evident from Fig. 5(a) that the average number of URL index varies linearly with the number of URL advertisements for a network of fixed size. While Fig. 5(b) presents that the average number of softlinks per node decreases with increased network size when the number of advertised URLs is fixed. These curves prove the uniform distribution of URL indexes and softlinks over the nodes. It should also be noted that the number of indexed URLs and softlinks become 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 $\overline{c_k}$. Here $\overline{c_k}$ is the bit-wise complement of codeword c_k.



(a) URLs per node (b) Softlinks per node (c) Query routing efficiency

Figure 5: Index overhead and query routing efficiency

4.2.4 Effectiveness of Soft-link

In this experiment, (Fig. 5(c)), we measure the impact of softlinks on query resolution. We run this experiment in a network of 50,000 nodes and measure average routing hops with different numbers of simultaneous queries. The average number of hops for resolving a query decreases when the number of queries increases because of the increased 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 low average hops per query.

4.3 Ranking Performance

In this section we measure the accuracy (Section 4.3.1) and convergence time (Section 4.3.2) of the distributed ranking mechanism as presented in Section 3.3.2.

4.3.1 Accuracy

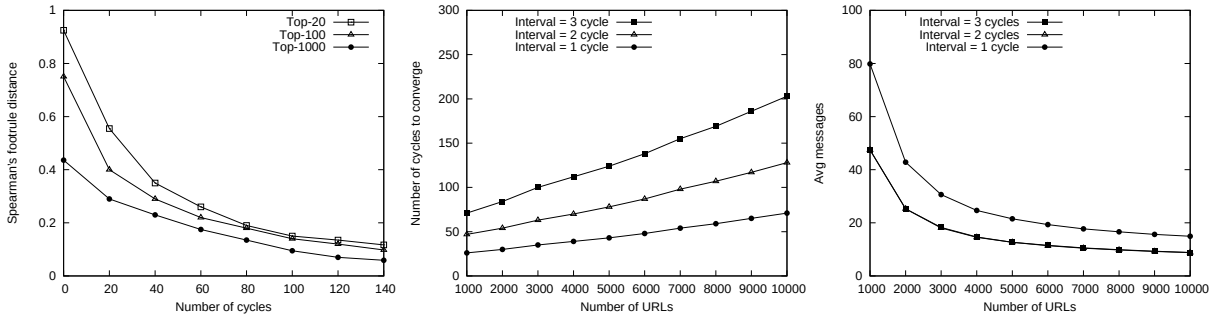
We use Spearman's footrule distance[4] to measure the accuracy of distributed PageRank algorithm in DEWS. Fig. 6(a) presents Spearman's footrule distances between distributed PageRank in DEWS and centrally computed PageRank for top-20, top-100, and top-1000 results. We advertised 10,000 URLs on a network of 50,000 nodes and PageRank update interval was set to 2 cycles. It is evident from Fig. 6(a) that Spearman's footrule distance drops significantly for the first 60 cycles due to rapid convergence in our distributed PageRank algorithm, while PageRank values become almost constant after 120 cycles. It should also be noted that the Spearman's footrule distance becomes around 0.1 after 100 cycles. It indicates that the distributed PageRank weights become very close to the centrally computed PageRank weights.

[4]For two ordered lists σ_1 and σ_2 of size k each, Spearman's footrule distance is defined as $F(\sigma_1, \sigma_2) = \frac{\sum_{i=1}^{k} |\sigma_1(i) - \sigma_2(i)|}{k*k}$, where $\sigma_1(i)$ and $\sigma_2(i)$ are the positions of URL 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. Evidently the lower the Spearman's footrule distance the better the ranking accuracy

4.3.2 Convergence

In this experiment we measure convergence time (in number of cycles) and network overhead (in number of messages) for the distributed ranking algorithm in DEWS. We performed this experiment on a network of 50,000 nodes and observed convergence behavior *w.r.t.* the number of advertised URLs. We used a value of 0.0001 for θ - the update propagation threshold (see Algorithm 3) and varied update propagation interval from 1 to 3 cycles. We assume that the distributed ranking algorithm has converged when the PageRank weights converge for every URL.

It can be seen from Fig. 6(b) that the number of cycles to converge does not increase significantly as the number of advertised URL increases. On the other hand, convergence time reduces as update propagation interval is reduced from 3 cycles to 1 cycle. Obviously this reduction in convergence time is achieved at the expense of increased network overhead as can be seen in Fig. 6(c). As the update propagation interval increases, each node gets enough time to accumulate all incoming PageRank weight and the computed PageRank weight converges faster.

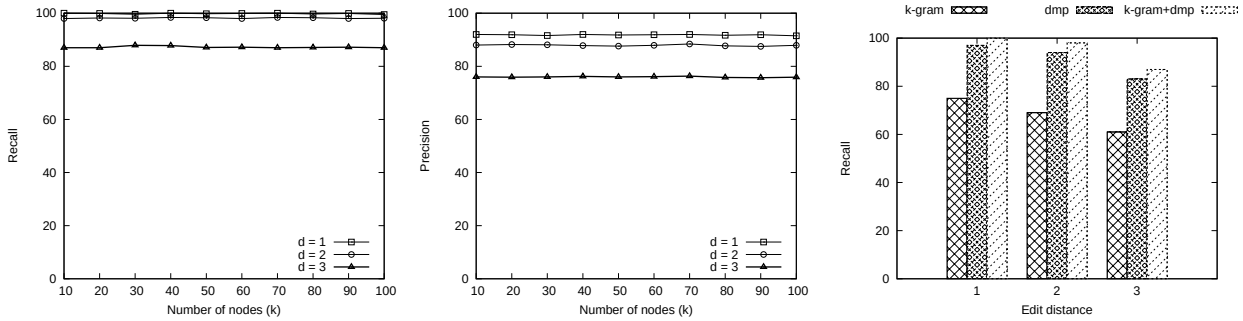


(a) Spearman's footrule distance (b) Convergence time (c) Network overhead

Figure 6: Ranking performance

4.4 Search Performance

4.4.1 Flexibility and accuracy



(a) Recall (b) Precision (c) Impact of phonetic encoding and k-grams

Figure 7: Search performance in DEWS

DEWS provides flexible searching with partially specified or misspelled keywords. We indexed 10K URLs and associated representative keywords in networks of different sizes. We generated 10K queries from randomly selected indexed keywords by varying edit distances from one to three. Figure 7(a) presents average recall rate of the search results in varied network sizes. This graph shows that recall rate remains constant at 100%, 98%, and 87% for edit distances one, two and three, respectively. Recall rate is lower for higher edit distances because the Hamming

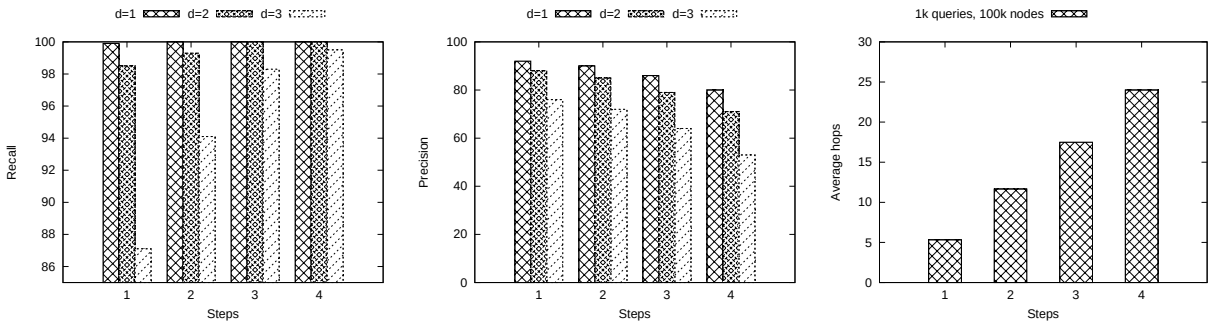
distance between advertisement and query patterns is proportional to the edit distance between advertisement and query keywords.

Figure 7(b) presents the average precision of search results for 10K queries in networks of different sizes. From this graph, it is observed that precisions remain constant at 92%, 88%, and 76% for edit distances one, two, and three, respectively. Precision becomes lower when edit distance increases because many irrelevant websites are included in the search results.

Notably, the results in Figure 7 affirm that precision and recall are independent of network size and only depend on edit distance between advertisement and query keywords. DEWS achieves a recall rate of 98% and a precision of 88% for queries with edit distance two from the original keywords. However, recall and precision of search results drop to 87% and 76%, respectively for edit distance three. Hence, we can expect to achieve very good precision and recall for partially specified or misspelled keywords within two edit distance away from the correct one.

Figure 7(c) shows the average recall rate of the search results for 10K queries in a network of 100K nodes where the query keywords have varying edit distances from the advertised keywords. In this experiment we evaluate the impact of Double Metaphone encoding and k-grams decomposition of query or advertisement keywords on recall. In this graph, we use 'dmp' and 'k-gram' for Double Metaphone encoding and k-grams, respectively. From this figure, it can be observed that the best recall is achieved when Double Metaphone encoding and k-grams decomposition is combined for generating the advertisement and query patterns. In this experiemnt we used $k = 1$, *i.e.*, each character is inserted independently in the Bloom filter.

4.4.2 Incremental retrieval



(a) Recall (b) Precision (c) Network bandwidth

Figure 8: Incremental retrieval

We generated 10,000 search keywords with edit distances one, two, and three from the advertised keywords, and measured the recall and precision of the search results. We measured these two metrics in four steps of incremental retrieval. Figure 8(a) presents the recall rate. 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. Recall rate is lower for higher edit distances because Hamming distance between advertisement and query patterns is proportional to the edit distance between advertisement and query keywords.

Figure 8(b) shows the precision for the results found in the four subsequent steps. In the first step, the precision values are 92%, 88% and 76% for edit distances one, two, and three, respectively. This figure represents two facts: i) precision decreases with increasing 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 match approximately between the search keywords and indexed keywords, DEWS picks some irrelevant websites, hence the decrease in precision.

Figure 8(c) shows the number of routing hops in different steps (with increasing decoding radius) for 1000 simultaneous queries in a network of 100K nodes. The total number of routing hops increases with increasing decoding radius,

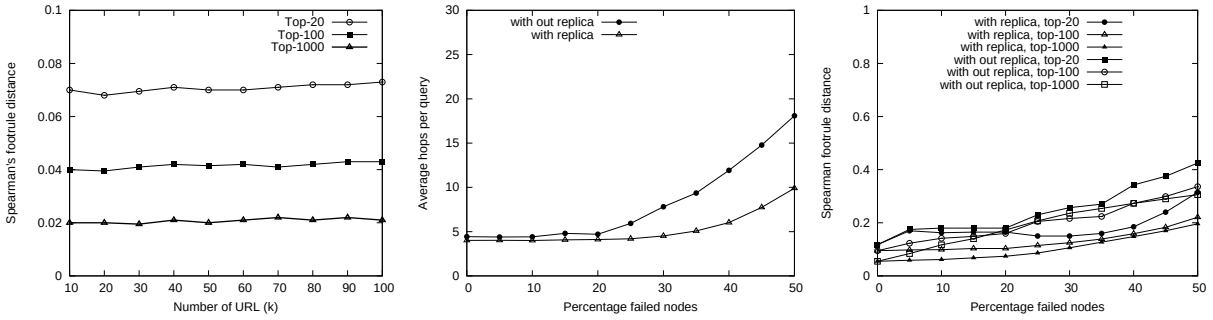
which is expected. It should be noted that the number of codewords, and hence the number of responsible nodes, increases non-linearly with increase in decoding radius. However, due to the overlay proximity of target nodes in consecutive steps network overhead has not increased significantly. DEWS provides most relevant results in the first step with high accuracy, which avoids the requirement to perform subsequent steps for most users. These steps will be performed only when users are not satisfied with the initial results. This mechanism reduces overall bandwidth consumption and provides improved user experience.

4.4.3 Accuracy of BM25

We have used Term Frequency (tf) as the relevance of a keyword k_{ij}^{rep} and also computed Inverse Document Frequency (idf) during query resolution. This mechanism implicitly computes the well known BM25 score. We have computed the Spearman's footrule distance between the search results sorted by our BM25 score (implicitly computed) and centrally computed BM25 score (available from LETOR 3.0). Figure 9(a) presents the Spearman's footrule distance between the two search results for top-20, top-100, and top-1000 results. We indexed 10K to 100K URLs and their associated keywords on a network of 100K nodes. The only difference for computing BM25 scores between the computation in DEWS and centralized manner is that DEWS uses an approximate value of N (number of collections) during query resolution instead of exact value. Figure 9(a) shows that Spearman's footrule distances remain constant at 0.07, 0.043, and 0.02 for top-20, top-100, and top-1000, respectively. It indicates that DEWS computes the BM25 scores efficiently in distributed manner.

4.5 Failure Resilience

DEWS has to work on a continuously changing overlay topology as new webservers can join DEWS network and existing webservers may fail or leave. We used the built-in abilities of Plexus routing for alternate route selection and replication to achieve failure resilience. In this section we investigate the impact of failure on query routing (Fig. 9(b)) and ranking accuracy (Fig. 9(c)) in a network of 50K nodes. We can get couple of insights from these two graphs. First, routing performance and ranking accuracy in DEWS do not degrade when the failure rate is below 20%. Second, average hops for query resolution increases as the percentage of failed nodes increase (see Fig. 9(b)). In presence of node failures, Plexus routes queries in alternate routing paths and possibility of message aggregation decreases, hence the increase in routing hops. Third, the average hops per query is greater for 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 are missing from search result and hence Spearman's footrule distance increases.



(a) Accuracy of BM25 computation (b) Avg hops for query resolution (c) Spearman's footrule distance

Figure 9: Accuracy of BM25 computation and Robustness of DEW in dynamic environment

5 Conclusion and future work

In this work, 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. In DEWS we have the provision for approximate matching on query keywords, and distributed ranking on semantic relevance and link-structure characteristics. As demonstrated by the experimental results, network and storage overheads for achieving this decentralization is not significant and the proposed framework scales well with network size and number of indexed websites. In addition, the ranking accuracy of DEWS is comparable to the ranking accuracy of a centralized ranking solution. The route aggregation technique proposed in this work outperforms the original Plexus routing protocol in terms of network usage efficiency. Experimental results also show that DEWS is highly resilient to node failures due to the existence or alternate routing paths and smart replication policy. Neither routing efficiency nor ranking accuracy degrades significantly even in presence of 20% failures. Compared to a centralized solution, DEWS will incur some network overhead. In exchange DEWS will give us the freedom of searching and exploring the Web without any control or restrictions, as can be imposed by the contemporary search engines.

The concepts presented in this work have been validated with rigorous simulations and critical analysis of the simulation results. As a next step, we are developing a DEWS prototype that can be deployed on a real network. We are also developing an improved ranking system where by PageRank weights are influenced by the number of common keywords between two pages. We expect this scheme to yield semantically more accurate results.

References

[1] R. Ahmed and R. Boutaba. Plexus: A scalable peer-to-peer protocol enabling efficient subset search. *IEEE/ACM TON*, 17(1):130–143, 2009.

[2] 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.

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

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

[5] V. Gopalakrishnan, R. Morselli, B. Bhattacharjee, P. Keleher, and A. Srinivasan. Distributed ranked search. In *Proceedings of the 14th international conference on High performance computing*, HiPC'07, pages 7–20, Berlin, Heidelberg, 2007. Springer-Verlag.

[6] M. Kale and P. S. Thilagam. Dyna-rank: Efficient calculation and updation of pagerank. In *Proceedings of the 2008 International Conference on Computer Science and Information Technology*, ICCSIT '08, pages 808–812, Washington, DC, USA, 2008. IEEE Computer Society.

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

[8] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark Dataset for Research on Learning to Rank for IR. In *LR4IR*, 2007.

[9] Z. Lu, B. Ling, W. Qian, W. S. Ng, and A. Zhou. A distributed ranking strategy in P2P based IR systems. In *APWeb*, pages 279–284, 2004.

[10] R. Neumayer, C. Doulkeridis, and K. Nørvåg. A hybrid approach for estimating document frequencies in unstructured p2p networks. *Inf. Syst.*, 36(3):579–595, May 2011.

[11] 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.

[12] J. X. Parreira and G. Weikum. JXP: Global Authority Scores in a P2P Network. In *8th Int. Workshop on Web and Databases (WebDB)*, 2005.

[13] L. Philips. The double metaphone search algorithm. *C/C++ Users J.*, 18:38–43, June 2000.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31:161–172, August 2001.

[15] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed pagerank for p2p systems. In *Proc. of IPDC*, pages 58–68. IEEE, 2003.

[16] S. Shi, J. Yu, G. Yang, and D. Wang. Distributed page ranking in structured p2p networks. In *Proc. ICPP*, pages 179–186, 2003.

[17] 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 TON*, 11(1):17–32, 2003.

[18] 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.

[19] Y. Wang and D. DeWitt. Computing pagerank in a distributed internet search system. In *Proc. VLDB*, volume 30, pages 420–431, 2004.

[20] J. Wu and K. Aberer. Using siterank for decentralized computation of web document ranking. In P. D. Bra and W. Nejdl, editors, *AH*, volume 3137 of *Lecture Notes in Computer Science*, pages 265–274. Springer, 2004.

[21] J. Wu and K. Aberer. Using a Layered Markov Model for Distributed Web Ranking Computation. In *Proc. ICDCS*, pages 533–542, Jun. 2005.

[22] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE JSAC*, 22(1):41–53, Jan. 2004.

[23] Y. Zhu, S. Ye, and X. Li. Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *Proc. ACM ICIKM*, pages 578–585, 2005.