Information Systems 34 (2009) 290–303

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

PHIRST: A distributed architecture for P2P information retrieval[☆]

Avi Rosenfeld[a,*,1], Claudia V. Goldman[c], Gal A Kaminka[b], Sarit Kraus[b]

[a] *Department of Industrial Engineering, Jerusalem College of Technology, Jerusalem, Israel*
[b] *Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel*
[c] *Samsung Telecom Research Israel, Yakum, Israel*

A R T I C L E I N F O

Article history:
Received 22 August 2007
Received in revised form
5 June 2008
Accepted 24 August 2008
Recommended by: P. Loucopoulos

Keywords:
Distributed databases
Information retrieval
Peer to peer systems

A B S T R A C T

Recent progress in peer to peer (P2P) search algorithms has presented viable structured and unstructured approaches for full-text search. We posit that these existing approaches are each best suited for different types of queries. We present PHIRST, the first system to facilitate effective full-text search within P2P databases. PHIRST works by effectively leveraging between the relative strengths of these approaches. Similar to structured approaches, agents first publish terms within their stored documents. However, frequent terms are quickly identified and not exhaustively stored, resulting in a significant reduction in the system's storage requirements. During query lookup, agents use unstructured search to compensate for the lack of fully published terms. Additionally, they explicitly weigh between the costs involved in structured and unstructured approaches, allowing for a significant reduction in query costs. Finally, we address how node failures can be effectively addressed through storing multiple copies of selected data. We evaluated the effectiveness of our approach using both real-world and artificial queries. We found that in most situations our approach yields near perfect recall. We discuss the limitations of our system, as well as possible compensatory strategies.

1. Introduction

Full-text search, or the ability to locate documents based on terms found within documents, is arguably one of the most essential tasks in any distributed database [1]. Search engines such as Google [2] have demonstrated the effectiveness of centralized search. However, classic solutions also demonstrate the challenge of large-scale search. For example, a search on Google for the word, "a", currently returns over 15 billion pages [2]. Though Google's servers are capable of storing this magnitude of storage, this approach is infeasible for distributed solutions involving more limited devices.

In this paper, we address the challenge of implementing full-text search within peer-to-peer (P2P) network databases. Our motivation is to demonstrate the feasibility of implementing a P2P database comprised of resource limited machines, such as handheld devices. Thus, any solution must be keenly aware of the following constraints: *cost*—many networks, such as cellular networks, have costs associated with each message. One key goal of the system is to keep communication costs low. *Hardware limitations*—we assume each device is limited in its amount of storage. Any proposed solution must take this limitation into consideration. *Distributed*—any proposed solution must be distributed equitably. As we assume a network of agents with similar hardware composition, no one agent can be required to have storage or communication requirements grossly beyond that of other machines.

[☆] This material is based upon work supported in part by ISF Grant # 1685/07.
[*] Corresponding author. Tel.: +972 8 926 8028.
E-mail addresses: rosenfa@jct.ac.il (A. Rosenfeld), c.goldman@samsung.com (C.V. Goldman), galk@cs.biu.ac.il (G.A. Kaminka), sarit@cs.biu.ac.il (S. Kraus).
[1] Portions of the work by the primary author was completed while at Bar-Ilan University.

0306-4379/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2008.08.002

Resilient—our assumption is that peers are able to connect and disconnect at will from the network. As a result, our system must be able to deal with peer failures, a concept typically referred to as churn [3,4].

To date, three basic approaches have been proposed for full-text search within P2P databases [5]. Structured approaches are based on the classic information retrieval (IR) theory [6], and use inverted lists to quickly find query terms. However, they rely on expensive publishing and query lookup stages. A second approach creates super-peers, or nodes that are able to locally interact with a large subset of agents. While this approach does significantly reduce publishing costs, it violates the distributed requirement in our system. Finally, unstructured approaches involve no publishing, but are unsuccessful in locating hard to find items [5].

In this paper we present PHIRST, a system for *P*eer-to-peer *H*ybr*i*d *R*estricted *S*earch for *T*ext. This approach has three key contributions. First, PHIRST is the first system capable of performing distributed full-text search—something previously thought to be infeasible [1]. The key to PHIRST's success is its ability to restrict the amount of data needed to be published to execute full-text search. Not only does this ensure that the hardware limitations of agents' nodes are not exceeded, it also better distributes the system's storage. Furthermore, a peer's average data load actually decreases as peers with documents are added. Thus, the system becomes progressively more scalable as its size increases. Nonetheless, PHIRST is still able to effectively process full-text search through a hybrid approach that leverages the advantages of structured search (SS) and unstructured search (US) algorithms. PHIRST's limited published data are used to locate hard-to-find items. US is used to find common terms that were not published. Second, not only does PHIRST present a feasible approach for full-text search, but it also processes these searches with lower cost as well. We also present full-text query algorithms where nodes explicitly reason based on estimated search costs about which search approach to use, reducing query costs. Finally, we present how storing redundant copies of these entries can effectively deal with temporary node failures without the need of any centralized mechanism.

To validate the effectiveness of PHIRST, we used a real web corpus [7]. We found that the hybrid approach we present used significantly less storage to store all inverted lists than previous approaches where all terms were published [1,5]. Next, we used artificial and real queries to evaluate the system. The artificial queries demonstrated the strengths and limitations of our system. The unstructured component of PHIRST was extremely successful in finding frequent terms, and the structured component was equally successful in finding any pairs of terms where at least one term was not frequent. In both of these cases, the recall of our system was always 100%. The system's performance did have less than 100% recall when terms of 2 or more words of medium frequency were constructed. We present several compensatory strategies for addressing this limitation in the system. Finally, to evaluate the practical impact of this potential drawback, we studied real queries taken from IMDB's movie database [8] and found PHIRST was in fact effective in answering these queries.

2. Related work

Classical IR systems use a centralized server to store inverted lists of every term in every document within the system [6]. These lists are "inverted" in that the server stores lists of the location for each term, and not the term itself. Inverted lists can store other information, such as the term's location in the document, the number of occurrences for that term, etc. Search results are then returned by intersecting the inverted lists for all terms in the query. These results are then typically ranked using heuristics such as TF/IDF [9]. For example, if searching for the terms, "family movie", one would first lookup the inverted list of "family", intersect that file with that of "movie", and then order the results before sending them back to the user.

The goal of a P2P system is to provide results of equal quality without the need of a centralized server with the inverted lists. Potentially, the distributed solution may have advantages such as no single point of failure, lower maintenance costs, and more up-to-date data. Toward this goal a variety of distributed mechanisms have been proposed.

Structures such as distributed hash tables (DHTs) are one way to distribute the process of storing inverted lists. Many DHT frameworks have been presented, such as Bamboo [4], Chord [10], and Tapestry [11]. A DHT could then be used for IR in two stages: publishing and query lookups. As agents join the network, they need to update the system's inverted lists with their terms. This is done by every agent sending a "publish" message to the DHT with the unique terms it contains. In DHT systems, these messages are routed to the peer with the inverted list in $\log(N)$ hops, with N being the total number of agents in the network [4,10]. During query lookups, an agent must first identify which peer(s) store the inverted lists for the desired term(s). Again, this lookup can be done in $\log(N)$ hops [4,10]. Then, the agent must retrieve these lists and intersect them to find which peer(s) contain all of the terms.

Li et al. [1] present formidable challenges in implementing both the publishing and lookup phases of this approach in large distributed networks. Assuming a word exists in all documents, its inverted list will be of this length. Thus, the storage requirements for these inverted lists are likely to exceed the hardware abilities of agents in these systems as the number of documents grows. Furthermore, sending large lists will incur a large communication cost, even potentially exceeding the bandwidth limitation of the network. Because of these difficulties, they concluded that naive implementations of P2P full-text search are simply infeasible.

Several recent developments have been suggested to make a full-text distributed system viable. One suggestion is to process the SS starting with the node storing the term with the fewest peer entries in its inverted list. That node then forwards its list to the node with the next longest list,