ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval

Torsten Suel* Chandan Mathur Jo-Wen Wu Jiangong Zhang
Alex Delis Mehdi Kharrazi Xiaohui Long Kulesh Shanmugasundaram

Department of Computer and Information Science
Polytechnic University
Brooklyn, NY 11201

1. INTRODUCTION

Most major search engines are currently based on cluster architectures, with large numbers of low-cost servers located at one or a few locations and connected by high-speed LANs [2]. Recently, there has been a lot of interest in using peer-to-peer (P2P) architectures to provide large-scale services, and several groups have proposed scalable substrates for P2P applications [9, 11, 13, 14].

Here, we study the problem of building a P2P-based search engine for massive document collections on top of such a substrate. We describe a prototype system called ODISSEA (Open DIStributed Search Engine Architecture) that is currently under development in our group. ODISSEA provides a highly distributed global indexing and query execution service that can be used for content residing inside or outside of a P2P network. ODISSEA is different from most other approaches to P2P search in that it assumes a two-tier search engine architecture and a global index structure that is distributed over the nodes of the system. In this short paper, we give an overview of the proposed system and discuss some basic design choices. We also discuss some preliminary simulation results for distributed query processing on a terabyte-size web page collection that indicate good scalability for our approach.

2. ODISSEA DESIGN OVERVIEW

ODISSEA is a distributed global indexing and query execution service, i.e., a system that maintains a global index structure under document insertions and updates and node joins and failures, and that executes simple but general classes of search queries in an efficient manner. This system provides the lower tier of a proposed two-tier search infrastructure. In the upper tier, there are two classes of clients that interact with this P2P-based lower tier:

1. *Update clients* insert new or updated documents into the system, which stores and indexes them. Update client could be crawlers inserting crawled pages, web servers pushing documents into the index, or nodes in a file sharing system.
2. *Query clients* design optimized query execution plans, based on statistics about term frequencies and correlations, and issue them to the lower tier. Ideally, query clients should be able to use or implement various different ranking methods.

There are two main differences that distinguish ODISSEA from other P2P search systems. First, the assumption of a two-tier architecture that aims to give as much freedom as possible to clients to implement their own policies. The second difference is our assumption of a *global inverted index* structure. Most current approaches (see [10] for an exception) for P2P search assume a local index, where each node maintains an index for its local documents, and queries have to be broadcast to all, or usually at least a significant fraction, of the nodes, in order to get the best results.

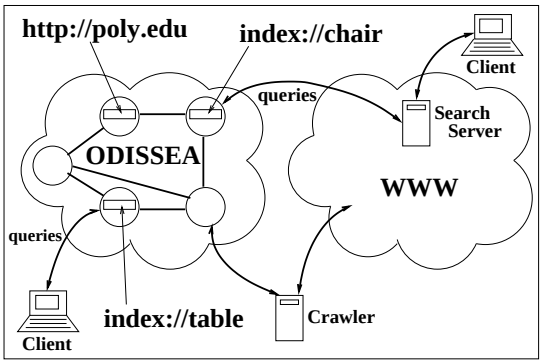

Figure 1: ODISSEA as a web search infrastructure, with a web crawler as update client, and a client-based and a web service-based search client.

Figure 1 shows the basic design. We decided to implement the system on top of an underlying global address space provided by a DHT structure, in particular Pastry [11]. Each object is identified by a hash of its *name*, where the name is the URL of a document or a string such as `index://chair` for the index structure for the term "chair", and is assigned a location determined by the DHT mapping scheme. Thus, the only way to move an object is to rename it, resulting in a mapping to a random other node.

We have four main applications that motivate our research:

(1) to provide full-text search facilities for large document collections within P2P communities.

(2) to search document collections in large intranet environments.

(3) to build a P2P-based search infrastructure for the web as an alternative to the major search engines. (This was our initial and probably most ambitious application.)

(4) for use as "middleware" on top of a system of local indexes that periodically insert postings into the global index for faster processing of certain types of queries.

3. DISCUSSION AND JUSTIFICATION

Two-tier approach: Given the increases in speed and bandwidth of desktop systems, we see the potential for a rich variety of novel search tools and interfaces that exploit client computing resources,

*Contact author. Email: `suel@poly.edu`. Research partly supported by NSF CAREER Award NSF CCR-0093400.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

and that rely on a powerful lower-level web search infrastructure. These tools may perform a large number of web server or search engine accesses during a single user interaction, in order to prefetch, analyze, aggregate, and render content from various sources. Early examples of client-based tools are the Alexa and Google Toolbars, Leticia and PowerScout [7], or tools built with the Google API.

The proposed system would provide such a lower-level search infrastructure, with an powerful open and *agnostic* API that is accessed by client- and proxy-based tools. By *agnostic*, we mean an API that is not limited to a single method for ranking pages (e.g., the Google API, which returns pages according to Google's ranking strategy), but that allows clients to implement their own ranking. There clearly are limits and trade-offs to this goal. The most general solution of performing the ranking at the client requires large amounts of data to be transferred. On the other hand, we believe that limited but powerful classes of ranking functions can be supported by providing appropriate "hooks" in the system.

Global vs. local index: In a local index organization, each node creates its own index for its locally stored documents. Thus, each node has its own postings list for common words such as "chair" or "table", and a query "chair table" is broadcast to all nodes and then the results are combined. In a global index organization, each node holds a global postings list for a subset of the words, as determined, e.g., by hashing. Thus, a query "chair table" is first routed to the node holding the list for "chair" (the shorter list), which then sends its complete list to the node holding "table".

The main problem with local indexes is that all or most nodes are contacted for most queries, and thus they are unlikely to scale beyond a few hundred nodes. There have been attempts to overcome this issue by routing queries only to nodes likely to have good results or those in the vicinity [6, 12, 4]. However, we do not believe that this approach will work well if result quality is a major concern. To see this, consider the current web, where an approach based on local indexes at each site would be either extremely inefficient or give very poor answers.

With a global index, on the other hand, large amounts of data are transmitted, since large document collections result in lists of megabytes or more for many common words. This problem has led some people to reject global indexes as unrealistic for environments with limited bandwidth. However, we believe this problem can be overcome with smart algorithmic techniques. One technique was recently described in [10], where Bloom filters are used to decrease the cost of intersecting lists over the network, though this only improves results by a small factor. We are currently experimenting with distributed query execution algorithms based on ideas by Fagin and others [5] that asymptotically reduce communication. We believe that these techniques, combined with good query optimization, will allow interactive response times on terabyte data sets.

4. QUERY PROCESSING IN ODISSEA

Search engines typically process queries by computing a score for each document containing all the query terms, and then returning the k documents with the highest score. There are many factors that may contribute to this score, including standard term-based scores, global page scores due to Pagerank [3] and similar methods, or distances between search terms in the documents. The simplest approach is to evaluate the score function for all documents in the intersection of the inverted lists for the search terms, resulting in transmissions of multiple megabytes. However, recent work by several groups [5] shows how to evaluate top-k queries without scanning over the entire intersection. The algorithms, originally proposed for multimedia queries (e.g., image retrieval) presort the inverted lists by the term-specific scores that contribute to the final score, and then attempt to determine the top k results by only accessing a small prefix of the lists and performing a limited number of random lookups on the list. We note that slightly similar techniques has also been proposed in the IR community [1, 8], but these usually access a much larger fraction of each list as they seek to avoid any random lookups.

	shortest 20%	middle 20%	longest 20%
Lists	20,277	401,553	4,859,241
A to B	3,229	12,902	3,508
B to A	2,502	7,256	3,223
Total bytes sent	45,847	161,267	53,850
Time in ms	629	1,206	669

Table 1: Communication costs of the protocol.

We have designed a simple distributed protocol based on these techniques, and have measured the performance of this protocol based on a trace from the Excite web search engine and a real data set of 120 million web pages that we crawled from the web. Table 1 shows average results for 120 queries with two keywords, excluding stop words. There are three columns of data, corresponding to queries with a very small, median, and very large length of the shorter of the two lists. The first line shows the average length of the shorter lists, which is the cost of the basic algorithm. The next lines show the number of postings sent from the shorter to the longer list, and vice versa. The final two lines were computed by assuming 8 bytes per posting, a total latency of 400 ms for the two messages, and a bandwidth of 200 KB/s. The score function used was a standard cosine measure, and similar results are obtained when including a normalized Pagerank score.

5. REFERENCES

[1] V. Anh, O. Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. of the 24th Annual SIGIR Conf.*, pages 35–42, 2001.

[2] E. Brewer. Lessons from giant scale services. *IEEE Internet Computing*, pages 46–55, August 2001.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of the Seventh World Wide Web Conf.*, 1998.

[4] F. Cuenca-Acuna and T. Nguyen. Text-based content search and retrieval in ad hoc p2p communities. In *Proc. of The Int. Workshop on Peer-to-Peer Computing*, May 2002.

[5] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2):109–118, June 2002.

[6] A. Kronfol. FASD: a fault-tolerant, adaptive, scalable, distributed search engine. June 2002. Unpub. manuscript.

[7] H. Lieberman, C. Fry, and L. Weitzman. Exploring the web with reconnaissance agents. *Comm. of the ACM*, 44(8), August 2001.

[8] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *J. of the American Society for Information Science*, 47(10):749–764, May 1996.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM Conf.*, 2001.

[10] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. February 2002. Unpublished manuscript.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Int. Conf. on Distributed Systems Platforms*, 2001.

[12] Y. Shen and D. L. Lee. An mdp-based peer-to-peer search server network. In *Proc. of the 3th International Conf. on Web Information Systems Engineering*, pages 269–278, 2002.

[13] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM Conf.*, 2001.

[14] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. TR UCB//CSD-01-1141, UC Berkeley, 2000.