Fault-tolerant Distributed Information Retrieval For Supporting Publius Servers and Mobile Peers

Katrina M. Hanna Brian Neil Levine R. Manmatha
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{hanna, brian, manmatha}@cs.umass.edu

***Abstract*— We show how dividing a database and replicating documents and indicies in an overlapping manner provides resilience in the face of node failures, malicious attacks, censorship attempts, and network partitions. This property of fault tolerance is beneficial for a wide range of scenarios, and we examine it in the context of two applications: an IR collection distributed over a set of mobile peers with wireless interfaces; and an IR collection distributed over servers set up to support censorship-resistant peer-to-peer file sharing and web publishing systems, such as Publius. Our use of random replication and split document sources makes it difficult for attackers to remove specific indexed content from the system. Moreover, we show the system is able maintain high IR performance even when 45 out of 50 nodes are unavailable. For mobile nodes, we have show that our design manages the randomness of mode mobility. Nodes are able to contact only direct neighbors who change frequently, not use ad hoc routing protocols, and still maintain good IR performance. This makes our design applicable to mobility situations where routing partitions are common. Our evaluation show nodes require only low additional storage on average.**

I. Introduction

Information Retrieval (IR) systems aim to satisfy a user need by retrieving documents and articles from collections that are the most relevant to a client-supplied query. IR manages unstructured, full-text documents, while traditional database retrieval requires that documents either be highly structured or tagged with meta information (e.g., "name" or "address"). For example, Google offers an IR search service in a centralized form. IR databases can also be divided among a set of *peers*. *We show how dividing a database and replicating documents and indicies in an overlapping manner provides resilience in the face of node failures, malicious attacks, censorship attempts, and network partitions*. This property of fault tolerance is beneficial for a wide range of scenarios, and we examine it in the context of two applications: an IR collection distributed over a set of mobile peers with wireless interfaces; and an IR collection distributed over servers set up to support censorship-resistant peer-to-peer file sharing and web publishing systems, such as Publius [21].

Mobile users who lack connectivity to a centralized, Internet-based IR collection, can benefit significantly from the ability to search documents stored by a network of peers. Commonly, mobile devices are resource-poor and there may not be a single node in a federation of mobile hosts that is capable of indexing voluminous content or responding to numerous queries. A group of mobile peers can share the work of indexing documents, storing those indicies, and responding to queries while providing coverage in a partitioned wireless environment. As an example, a set of doctors managing a refugee camp may desire access to a large collection of medical literature on diseases. In the scheme we propose here, they would each carry a small portion of such a collection (e.g., on a PDA) and resolve queries for other peers. Our evaluations show that our system is fault tolerant and successfully manages unavailability of other peers due to network partitions or attacks.

Publius [21] is a censorship-resistant, distributed file system. An important property of the Publius system is *deniability*, which is the ability of volunteers storing information to plausibly deny knowledge of the content they store. Servers in the Publius system maintain deniability by storing only encrypted content and never seeing plaintext versions; this makes indexing of content infeasible.

This paper was supported in part by National Science Foundation awards ANI-033055, EIA-9983215, and EIA-0080199 and in part by the Center for Intelligent Information Retrieval. Kat Hanna's work is supported in part by a National Science Foundation Graduate Research Fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsors.

We propose that a separate group of volunteer peers acting together to index documents and respond to queries can provide a valuable IR service for the Publius system. By only maintaining an index and not the documents stored in the Publius system, the search engine peers can maintain this deniability just as Publius servers do. Furthermore, our system is robust against denial of service attacks against servers and therefore has the same fault tolerance characteristics as Publius.

While these two scenarios differ, our proposal supports both. We have the following assumptions common to both scenarios. We assume the presence of a group of *search engine peers* (peers, for short) that are willing to be responsible for storing indicies of documents in the system and responding to queries. In the ad-hoc scenario, we expect the peers to be mobile devices; for Publius, we expect that dedicated servers will function as peers. In this work we address the fundamental question: *Can randomized replication of document indices across a group of peers provide the fault tolerance necessary to these and other scenarios, while providing accurate IR results and reasonable resource requirements?* Our results show that:

fix Our methods work well.

In the remainder of this paper we summarize related work (Section II); overview the application of our designs to Publius and Mobile devices (Section III); review our experimental methodology and evaluate our system designs (Section IV); and offer conclusions in Section V. An appendix offers additional related results.

II. Related Work

Our work is related to past work in peer-to-peer systems (including mobility and secure file systems) and distributed information retrieval. Although our work contains elements of both fields, it differs from existing work in important ways. Most previous work in peer-to-peer systems is focused on searching for files using well-known identifiers or a limited set of key words.

The goal of distributed IR is to find a subset of databases appropriate to a specific query, search that subset of databases, and then merge the ranked list obtained from each database into a single ranked list. Thus, previous work in distributed IR has concentrated on the database selection problem as well as the merging of results. In contrast the focus of our system here is to provide a fault tolerant, full-text search and retrieval capability for information spread over a nodes or peers. This fault tolerance enables resistance to censorship, terrorist attacks, disasters or mobile peers moving out of range. The fault tolerance is achieved by doing a certain amount of random replication.

A. Peer-to-peer systems

The first peer-to-peer application to gain widespread popularity was Napster [13], a network for trading music files in the mp3 format. Napster used a centralized approach to enable users to find desired files. A participant logging on to the Napster network would register the songs she wished to share with the Napster server. To find a desired song, she would submit a query to the server based on the artist's name or song title. The server would search its list of songs offered by current users and return a list of matches. The user would then connect to a peer chosen from the list and retrieve the desired file.

Gnutella [19], another popular peer-to-peer file sharing utility, is fully distributed, employing limited-scope flooding search. The extent of the search is controlled by a time-to-live field in the protocol's *query* message. The Gnutella protocol allows the query search criteria to be of arbitrary length and specifies no content type. In practice, the search criteria is a file name or keyword.

Chord [18] and CAN [17] use consistent hashing techniques to provide a location service. These systems map identifiers to nodes in large-scale distributed systems. Although these and other related systems may be suitable for searches based on keywords, they are not likely to be useful for full-text search, as each word in the document collection would be an identifier.

Papadopouli and Shulzrinne have proposed a related system for mobility. Their system, called 7DS [14], allows mobile peers to share cached data with other peers in an ad hoc system. In contrast with our system, 7DS finds only matches with exact data (e.g., an exact URL). Additionally, the same authors have evaluated power management schemes in broadcast and multicast search schemes for mobile devices [15] that could be applied to our work.

1) Secure Distributed File Systems: Publius is a censorship-resistant web publishing system that provides anonymity to publishers of content, and some deniability of the content of files to storers of data. Files in Publius are replicated across servers and stored in an encrypted form. Shamir's secret sharing is utilized to make decryption keys available to those wishing to retrieve files, while denying the storers of data local access to useful keys. A Publius URL encodes the locations of a file and its associated key shares. Publius currently provides no search mechanism and requires that a user have access to a directory of URLs to be able to retrieve documents. There are three problems with this approach to accessing material. First, with large numbers of documents, it is impractical for a person to navigate a directory. A search mechanism on the other hand would simplify access to documents. Second, if a content storer has access to the URLs it can

decode the locations, retrieve the necessary keys, and examine the content it stores. The easier it is for a content storer to access lists of URLs, the more her deniability is eroded. Third, the ability for consumers to find desired content via text search provides a higher level of deniability than publishing directories of Publius URLs.

Many secure, fault-tolerant, distributed file systems related to Publius have been proposed. These works include Freenet [6], Eternity [2], Mnemosyne [8], and Tangler [20], among others. Intermemory [5] is perhaps the system closest to ours in approach. It randomly replicates data using erase codes and therefore tolerates a number of node failures in the system (up to 17 in the current implementation). Our work is differentiated in that we are not trying to locate a specific document, but rather any data relevant to a user query. Though we provide details on our inter-operation with Publius here, it would be possible to extend our work to other secure file systems where the location of content at peers is not very dynamic.

B. Distributed IR

The traditional model of information retrieval systems assumes a centralized search engine (examples include Google.com or INQUERY [3]). In many situations, information is distributed over multiple databases and a centralized search engine may not be able to access it because it may not have direct access to the content of one or more databases. This may happen because the databases are private and access is limited or the network connectivity is poor. In such situations, the only direct access to such databases may be through a search interface specific to the database. The field of distributed information retrieval [4] focuses on searching a set of distributed database and returning a merged set of results.

Distributed IR requires first characterizing a subset of databases to find which ones are most likely to contain the answer, searching that subset of databases, and then merging the results to create a single ranked list. This model assumes that each search engine indexes a specific database or collection of documents (the databases may or may not overlap). Moreover, this model assumes that the included databases are highly available; if a particularly good collection is not reachable, accuracy results suffer. The main challenges in distributed IR are on determining how a client decides which database to search and how the results from multiple databases can be merged to produce a single rank ordering. Clients can pick different databases based on resource descriptions of each database. The resource descriptions could be provided by the owner of the database. STARTS, for example, is a standard format for describing and communicating the resources of each database [7].

Such protocols assume cooperation between providers which may be reasonable within an organization but may not be reasonable when many different entities are involved. For example, database providers could be uncooperative (because it is not in their interest) or may deliberately provide misleading information (say to drive traffic to their sites). Callan [4] describes a query based sampling technique to find such resource descriptions. After some subset of databases is selected based on these resource descriptions, they are then individually searched. The results from searching each of these databases must then be combined into a single ranked list. This is a difficult problem since the scores produced by search engines on different databases are usually not comparable. A number of approaches using some kind of normalized scores or other heuristic techniques have been developed [4]. More principled methods have been investigated for combining search engines [11] but the extension to multiple databases is non-trivial.

III. System Model

Peers or distributed systems that cooperatively provide an IR service face several challenges in maintaining availability of the service. For example, adversaries may attempt to launch denial-of-service attacks on Publius peers, or disrupt nodes or routes in an ad hoc network of wireless peers. Additionally, network partitions are frequent in a wireless environment, and node mobility makes it difficult to determine or predict exactly what nodes are going to be neighbors or clustered. Network partitions may also occur in the Internet environment in which Publius is assumed to operate.

Often, collections of documents have a natural content overlap, and many documents may be relevant to a clients query. However, when peers are unavailable to clients, a portion of the collection of documents are therefore also unavailable for query retrieval.

We propose that a strategy of intentionally replicating indicies (and documents if desired) at many peers trades storage resources for greater accuracy and robustness in the face of node failure or attack. We also expect replication to manage network partitions and balance work load.

For this initial study, we assume the following simple replication strategy:

1) An initial peer receives a new document, and indexes it with probability p chosen from a uniform distribution.
2) Regardless of whether the document is indexed, the

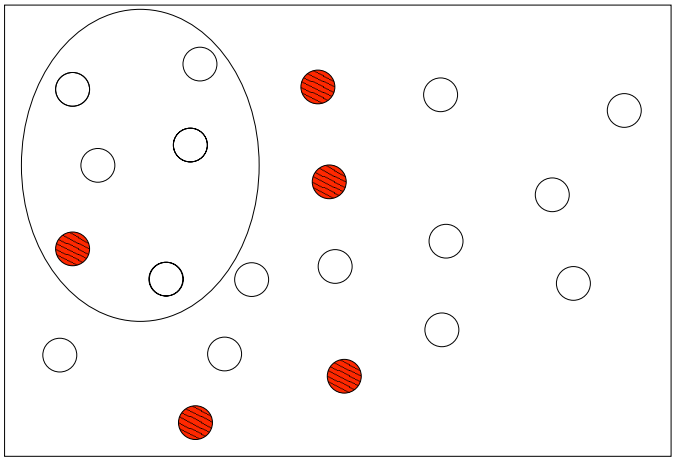

Fig. 1. A Group of mobile peers. Colored nodes have a particular document. The circle represents a collection of a node's neighbors within radio range.

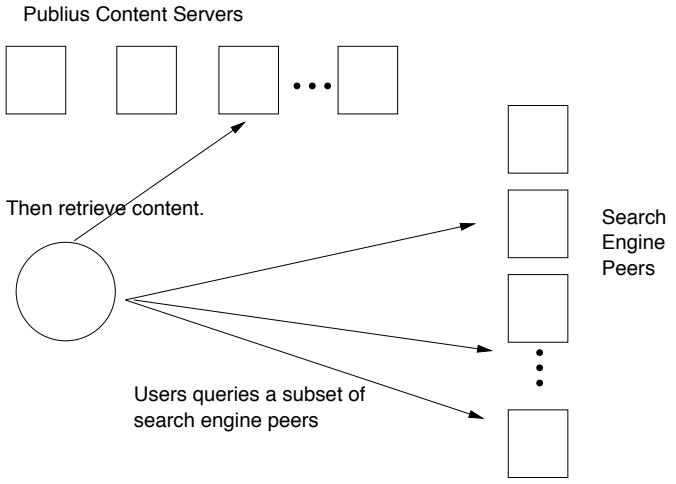

Fig. 2. Search and Retrieval of content in Publius.

peer then passes the document along to all its neighbors, who follow the same algorithm.

(In fact, documents need not be circulated; indicies of the documents, which are smaller may be compressed and distributed to peers instead.)

Accordingly, we assume the distributed set of peers in the Publius scenario are arranged in some logical topology, and ad hoc peers follow the topology provided by the underlying unicast routing services. In place of unicast routing, an efficient flooding or multicast strategy may be used.

Figure 1 shows a diagram of a sample system. The system shown contains 20 nodes; the five shaded nodes represent peers that have probabilistically indexed a given document. The subset in the upper, left corner represents a possible set of nodes to query.

Clients (or peers in the ad hoc system) initiate a query by selecting an initial peer. Given the random replication strategy, it is likely that neighbors have indexed a different, though not disjoint, subset of documents. In general terms, the more neighboring nodes the initial peer contacts to assist in resolving a query, the more accurate the merged results will be; however, contacting more neighbors delays the response and requires more work from the collective system.

Below, we outline the specific operation of our replication and retrieval scheme with Publius and mobile peers.

A. Supporting Publius

In December 2001, there were 47 servers listed on the Publius web site. We don't have an estimate of how much content each server hosts, nor the amount of space required to index that content. Typically, indexing requires less than half the size of the documents; therefore, it is safe to assume that the number of peers required to index the Publius system is no more than 47, assuming they volunteer resources roughly equivalent to those of the Publius servers. Accordingly, in this paper, we explore and evaluate the use of 50 peers. We expect this set of peers to be relatively static, thus the client will be free to choose the peer with whom to initiate a query.

The Publius protocol for publishing documents can be easily modified to include the operation of our peers. Publishers already must contact n servers in the system each of which must store the document. In order to have their documents indexed and search able in our peers, they must either contact some fraction p of all available peers; or they can contact all peers, who would index the document with probability p; or they can contact one peer, who will pass the document to all other peers for possible indexing. Which method is used is a matter of trust. Any methods used in Publius to contact servers anonymously can also be used to contact search engine peers anonymously.

B. Supporting Mobile Nodes

Mobile, ad-hoc networks are formed opportunistically from heterogeneous groups of devices moving within communication range of one another. Routing in these networks is cooperative, with nodes constructing paths through other nearby nodes, possibly to a stationary node, then onto the Internet.

In most cases, accessing desired information from peers in the ad-hoc network rather than from a server on another network is more efficient and does not require access points or base stations. Here we conceive of a group of users in a mobile, ad-hoc network as a group of peers. In contrast to the Publius example, this is a more traditional peer-to-peer application. Here nodes are both pro-

ducers and consumers, indexing available content as well as searching for that content.

We expect that documents can be pre-loaded onto devices when users are federated. With our techniques emergency rescue and medical workers could arrive at a remote disaster location (e.g., tornado, refugee camp, or other long-term, sub-acute disasters) with information such as: GIS information, medical algorithms and literature chemical hazard sheets; field medical manuals; the World Health Organization disaster medicine library (and images); immunization algorithms; and local or community info, such as information on health, fire, and police agencies, as well as Incident Command System job action sheets and command structures. All this information could not fit on a single (inexpensive) mobile device; but would be accessible in a federated set of devices.

Alternatively, as documents can be added interactively (as they are for Publius): they can be broadcast to all peers as the documents are brought into the system. Each peer would index the document with some pre-established probability p.

One assumption that we make for both scenarios, is that nodes are homogeneous in the resources that have available to them. As with any file system and retrieval technique, the hardware determines the real limit on the amount of content that can be stored. However, in a later section we do discuss the storage requirements of each node in the system. This is a challenges that pertains more to peer resources. Hand held wireless devices typically have limited storage capabilities, and are often limited in communication range. The greater the replication of documents and indices, the more storage space we require of an individual peer. On the other hand, if indices are more highly replicated, the number of peers it is necessary to search is smaller; and thus, the work required of each peer is less.

In this application, the number of peers can vary greatly, from tens of users in a small corner of campus to thousands of users attending a large sporting event. However, for both applications we find 50 nodes to be a reasonable number to investigate.

IV. Experimental Methodology

The quality of the information returned in an IR system may be evaluated by computing the *precision* of the system at R retrieved documents. Precision is the proportion of the information retrieved that is relevant to the query. In this section we describe our experiments for evaluating the precision of a variety of systems and present our results.

In this experiment we examined the IR performance achieved by our model as well as others and at what systems cost. The source databases we use in our experiments are from the Text Retrieval Conferences (TREC) run by the National Institute of Standards and Technology (NIST). Specifically we use volumes 1, 2, and 3 from TREC. The approximately 3.2 gigabytes of data is divided by source and publication year. We used a set of short query strings related to a range of topics covered by the databases. The queries are standard TREC queries; the relevance scores of documents in relation to these queries have been pre-established by NIST.

When initiating a set of queries, we randomly chose subsets of nodes from the entire set of peers. Strategies for distributing document indices across nodes greatly affect the IR performance of such a scheme. They also affect the amount of space required at each node and in the system as a whole. These strategies can be roughly categorized as replicated or not, and consisting of homogeneous or heterogeneous content. We refer to the latter two categories as *sources-together* and *sources-split* respectively. In our experiment, the distinction between the two lies in whether we distributed indices with document-level granularity, or as one or more chunks of a database from a single source (e.g., Wall Street Journal articles from 1999).

We compared the four combinations of the above categories:

- **Not Replicated, Sources-Together:** Database sources are divided into 50 roughly equal-sized chunks, with all indices from a chunk placed on one node.
- **Not Replicated, Sources-Split:** We distribute the document indices over the set of nodes in round-robin fashion. Each index is placed on exactly one node.
- **Replicated, Sources-Together:** All of the indices from a given database source are copied to three randomly-chosen nodes.
- **Replicated, Sources-Split:** This is our proposed distribution strategy as outlined in Section III. Each document index is placed on each node with some probability p.

These four distribution strategies require very different amounts of disk space. Table I shows the total space required to store the documents in TREC 1-2-3 as well as the minimum and maximum at a single node. The table also shows that the variation of storage costs at nodes is smaller with our scheme, which is essential for our assumption of heterogeneity. Figure 4 shows more detail on the replicated, source-split strategy that we advocate, including the average indexing and document costs per node

Strategy	Total Space (all nodes) MB	Min node MB	Max node MB
Not replicated, sources-split	3,185	60	66
Not replicated, sources-together	3,185	15	451
Replicated, sources-together	5,170	33	345
Replicated, sources-split ($p = 0.02$)	2,831	54	59
Replicated, sources-split ($p = 0.05$)	7,086	138	144
Replicated, sources-split ($p = 0.1$)	14,203	278	290

TABLE I
DISK SPACE REQUIRED TO STORE DOCUMENTS (MBYTES).

in terms of p.

One disadvantage of the replication strategies is that there is no guarantee that documents will be archived. A minor modification of our strategy could be made to ensure that the probability of archiving any document be one by simply ensuring that the first peer to receive a new document indexes it with probability one. However, a stronger constraint can be made. In the Appendix, we present an analysis using Chernoff bounds to determine bounds on the probability p necessary to ensure that a specific document is available with high probability within the set of nodes. However, such analysis does not directly tell us the probability of finding a relevant document since search engines do not necessarily find every document that is indexed but only some fraction of all the relevant documents indexed. When we factor this in, the probabilities of replication necessary to achieve good replication performance are much lower (if the search engine can only find 50% of the relevant documents then clearly it is not necessary to have all 100% of the relevant documents in the subset). Since search engines are quite difficult to model the appropriate probabilities of replication to achieve good retrieval performance must be determined empirically which we do below.

For our queries we used the INQUERY system [3]. INQUERY implements an inference net model for full-text retrieval. In conjunction with INQUERY, we used CORI [4], which allows retrieval from a distributed set of document databases. In its default mode of operation, CORI characterizes the content of the databases and when presented with a query, chooses the best databases to contact. Ranked results from different databases can usually not be compared directly [4]; CORI uses a heuristic method to normalize document scores based on the maximum and minimum score the document could achieve, allowing rankings to be merged.

In our experiments we specified the peers to use in a particular query, bypassing CORI's selection mechanism. We used CORI's heuristic method for combining the results returned by peers.

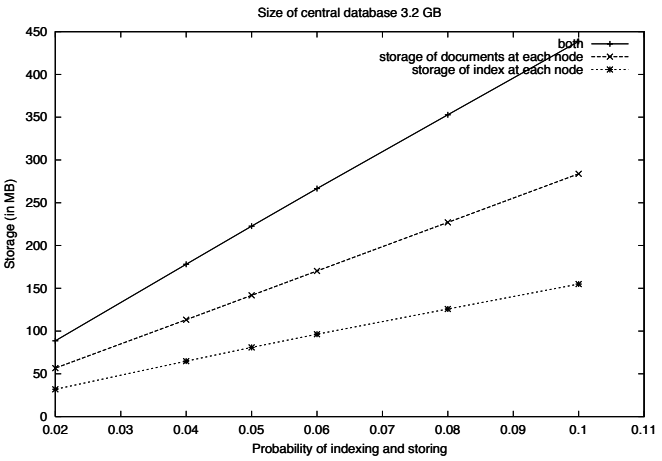

Fig. 4. Example storage requirements of each node for the rep-split scheme with increase probabilities of storage and indexing.

In this experiment, for each strategy, we examined the performance of querying subsets of $s = 5, 10, 15, 20$ and 25 nodes. For each subset size we randomly chose s nodes, then ran 50 queries to the chosen nodes. For each subset size, we repeated this 20 times, averaging the results, shown in Figure 3. Error bars on all graphs represent standard deviations.

A. Results

The four plots of Figure 3 show the results for retrieval of different numbers of documents ranging from 10 to 200. As is well known in IR, as more documents are retrieved by users, precision tends to drop. This is because a larger set of results makes it more challenging to locate only relevant results. A number of observations can be made from the graphs.

First, the replicated-source-split (rep-split, for short) strategy achieve higher accuracy than the other three scenarios for values of p from 0.04 to 0.10. For example, for retrieval of 20 documents, with subsets of five, rep-split with $p = 0.1$ has a 30% increase in precision

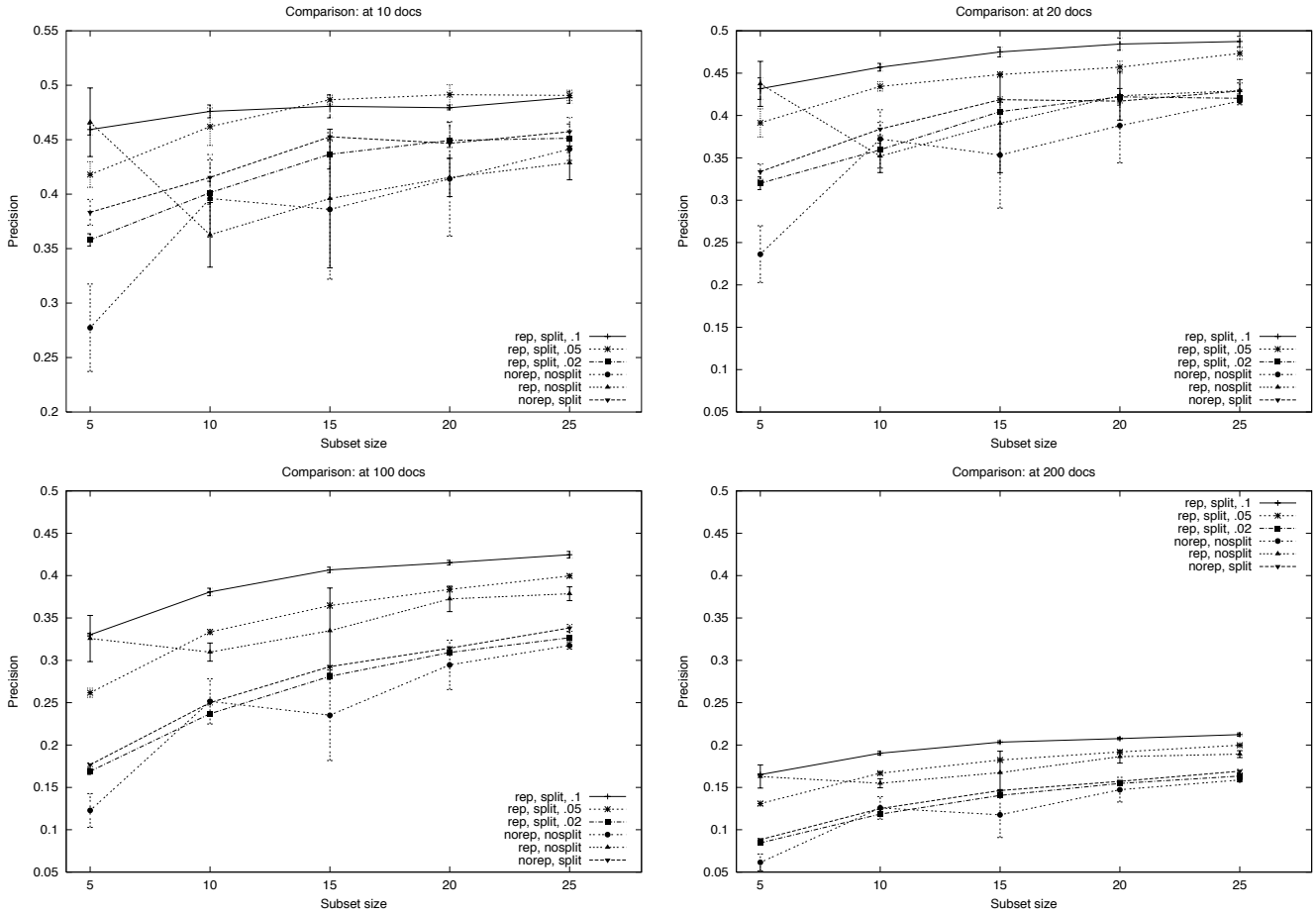

Fig. 3. Precisions comparison of all techniques and varying probabilities of indexing for different numbers of retrieved documents: (Top Left) 10 documents; (Top Right) 20 documents; (Bottom Left) 100 documents; (Bottom Right) 200 documents.

from .34 to .44 as compared to the non-replicated/split strategy. Moreover, the increase is greater over the no-replication/no-split and replicated/no-split strategies. When $p = .02$ there is not an advantage to our proposed strategy, and this is also where the (total) storage costs are no greater (see Figure I). Hence, we see a direct relationship between the cost of p in terms of storage and increased precision.

B. Evaluations of Mobile Hosts

Second, the other scenarios suffer from a high variance in their performance and disk space required at nodes. Here we see an advantage of splitting up collections at the document level.

Third, increased subset size does not make a significant difference in performance. From here we conclude that most of the advantage of contacting multiple servers is gained for lower subset sizes. We find this to be a particular advantage in application to Publius and mobility for a number of reasons that we describe presently.

In the Publius scenario, firstly this means that the IR performance of the system remains adequate even if only a very small number of servers remain (e.g., 5 out of 50) after a denial of service attack. Secondly, the replication at nodes removes the need for complicated methods of selecting servers based on relevance (e.g., CORI [10]) — it doesn't matter which subset of nodes is chosen by clients; instead, servers can be selection based on their network performance, a characteristic commonly ignored by IR server selection techniques. Finally, the amount of work required of each server is low since only a small subset of servers need be contacted by each user.

In the mobile scenario, the implication is that ad hoc routing is not required if enough neighbors are in range. In fact, in our evaluations of mobile nodes, presented in the next section, we do not employ any ad hoc routing between nodes. This simplifies the operational complexity of devices, reduces traffic on the network caused by flooded route requests (e.g., as done by AODV [16] or DSR [9]), and reduces work required of peers.

For Publius peers, we assume that servers are unavailable to nodes mainly because of attacks and outages. However, for mobile nodes, other peers may be unreachable simply due to network partitions that result from node

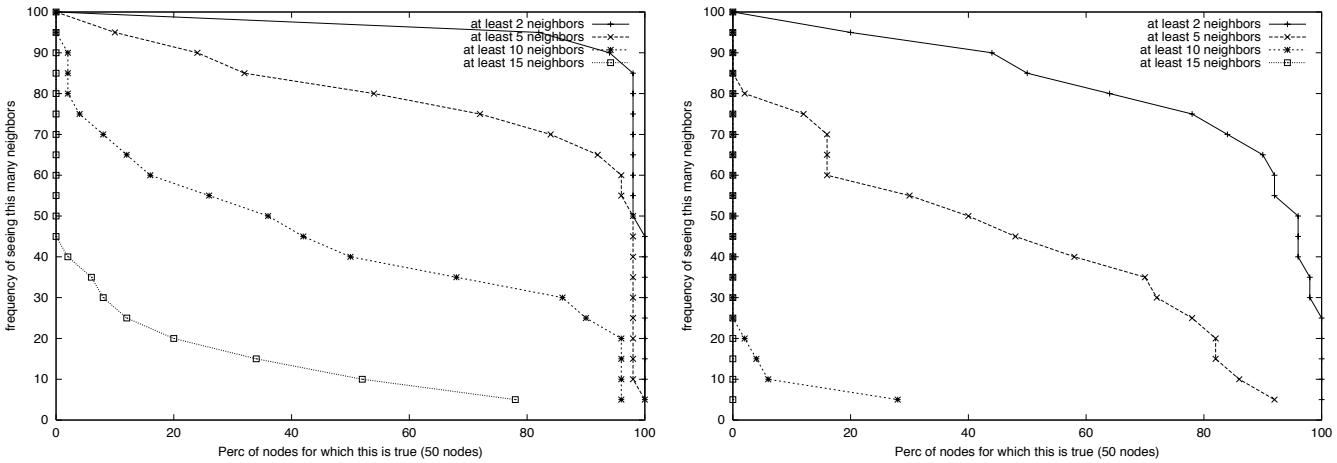

Fig. 5. Density of neighbors in the two mobile environments. (Left) 500m-by-500m. (Right) 1000m-by-1000m.

mobility and physical interference. If ad hoc routing protocols are in place (e.g., AODV or DSR), nodes can query any peer to which it has a route. We evaluated our method under the stricter assumption that nodes can only contact neighbors to which they are within broadcast range. Therefore, it is the density of nodes that most affects the performance of our scheme.

Accordingly, we set up simulations of two environments using the Rice University Monarch mobility extensions to NS2 [1]. For both simulations, we assumed 50 nodes moving in the *random waypoint model*. (This single mobility model was sufficient as a preliminary examination of the effects of node density.) Nodes moved at a speed of 2 meters per second and paused at their randomly-chosen destinations for 20 seconds. In the first scenario, the nodes moved in a field without obstructions 500m-by-500m. In the second scenario, we kept all parameters the same and increased the area of the field to 1000m-by-1000m. Figure 5 shows the density of each scenario; the plots show, for example, that in the smaller field, almost all the nodes had at least two neighbors 85% of the time. In the larger field, only about 45% of the nodes had at least 2 neighbors 85% of the time. Each node in the simulation is configured to simulate the range of an 802.11 interface. On average, nodes in the dense scenario had 16 neighbors with a standard deviation of 5.1; in the sparse scenario, nodes had 5.2 neighbors on average with a standard deviation of 2.7.

In our simulations, starting at 50 seconds into the simulation and then every 18 seconds, a node queries its direct neighbors (i.e., no ad hoc routing is used). We evaluated the precision of queries initiated by five randomly chosen nodes; this number was due only to the extremely long processing time of our simulation. Data for number of neighbors of each specific node we evaluated is shown in Figure 6.

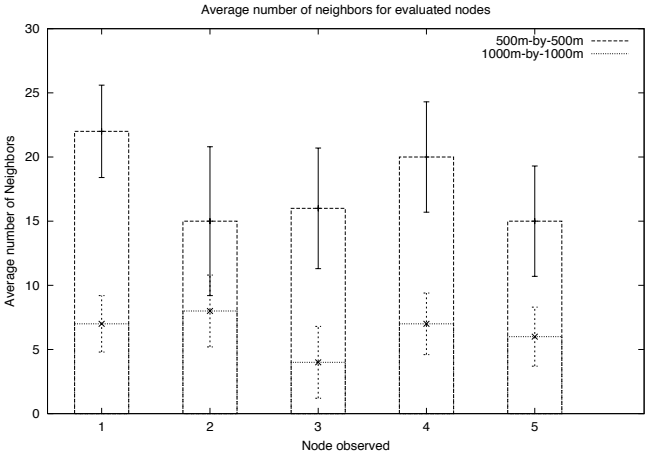

Fig. 6. Density of select nodes evaluated. Error bars indicate standard deviations.

Figure 7 shows the precision of the nodes for both dense and sparse simulations for a range different numbers of retrieved documents. The results show that the rep-split strategy is able to manage node mobility quite well. Because documents are randomly replicated, which neighbors are near to a node is now of no consequence. Furthermore, no ad hoc routing was employed in the simulation, which reduces the amount of work and complexity of the system.

Evaluating the amount of work performed by nodes in the simulation is simple. If we assign a unit cost to a query and each response, then the amount of work performed by any node is not more than the number of neighbors it has during the simulation — on average, the amount of work performed per query is proportional to the average number of neighbors it had, shown in Figure 6.

The TREC 1-2-3 database we examined requires 3.2 Gb of storage, however, even with $p = 0.05$, nodes in

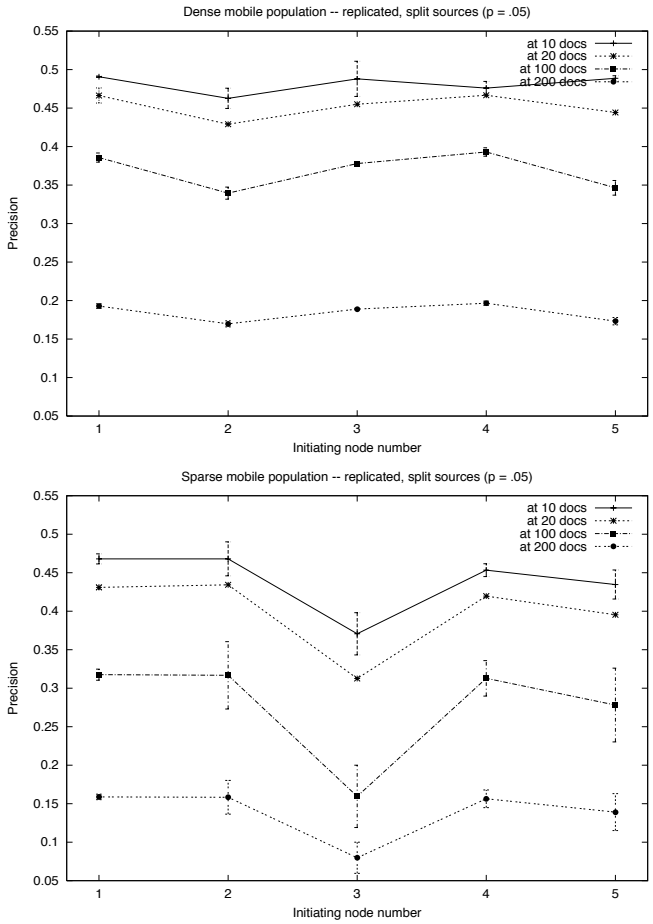

Fig. 7. Averaged information retrieval performance of five peers: (top) 500m-by-500m environment; (bottom) 1000m-by-1000m environment.

our simulation require on average only 141Mb of storage space for documents and 81 Mb for indicies; this is within the hardware resources of even a currently available Compaq iPAQ 3580. Even with only five neighbors available on average, the nodes were able to retrieve documents with high accuracy. We expect our technique will scale to much larger databases as each individual mobile device is capable of storing more data. For example, Compaq IPaq 3850s accepts SD memory cards; currently, 512 Mb SD cards are available and cards up to 4 Gb are planned. Extrapolating our results, with 4Gb on each mobile device, a database of over 60 Gb could be distributed over 50 peers when $p = 0.05$.

V. Conclusion

We have designed and evaluated a protocol for fault tolerant distributed information retrieval. We have shown the broad applicability of our design by considering two scenarios. First, IR search engines for Publius that are robust against denial-of-service attacks. Random replication of split sources makes it difficult for attackers to remove specific indexed content from the system. Moreover, by setting our indexing probability p to low values below 0.1, the system is able to return relevant results even when 45 out of 50 nodes are unavailable. Second, for mobile nodes, we have shown that our design manages the randomness of mode mobility. Nodes are able to contact only direct neighbors who change frequently, not use ad hoc routing protocols, and still maintain good IR performance. This makes our design applicable to mobility situations where routing partitions are common. Our evaluation of storage requirements show that nodes require about the same about of storage, making our system ideal for collections of homogeneous hardware. Additionally, the value of p is directly proportional to the amount of work required of peers. As we able to good results for low values of p, we expect our system to be applicable to low resource devices such as PDAs.

References

[1] Network simulator version 2 (ns2). http://http://www.isi.edu/nsnam/ns.

[2] R. Anderson. The eternity service. In *Proc. Pragocrypt*, 1996.

[3] J. Broglio, J.P. Callan, and W.B. Croft. Inquery system overview. In *Proceedings of the TIPSTER Text Program (Phase I)*. Morgan Kaufmann, 1994.

[4] Jamie Callan. *Advances in Information Retrieval*, chapter Distributed Information Retrieval, pages 127–150. Kluwer Academic Publishers, 2000.

[5] Y. Chen et al. A prototype implementation of archival intermemory. In *Proceedings of the Fourth ACM International Conference on Digital Libraries*, 1999.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, number LNCS 2009. Springer, 2001.

[7] L. Gravano, K. Chang, H. Garcia-Molina, and A. Paepcke. STARTS Stanford protocol proposal for Internet retrieval and search. In *Technical Report SIDL-WP-1996-0043, Computer Science Department, Stanford University*, 1996.

[8] S. Hand and T. Roscoe. Mnemosyne: Peer-to-peer steganographic storage. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.

[9] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. G. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks. IEEE Internet Draft, February 2002. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-07.txt.

[10] J.P.Callan, Z.Lu, and W.B.Croft. Searching Distributed Collections with Inference Network. In *the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval(SIGIR 95)*, pages 21–29, 1995.

[11] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *the Proc. of the 24th ACM SIGIR conf. on Research and Developement in Information Retrieval*, pages 267–275, Sept 2001.

[12] R. Motawani and P. Raghavan. *Randomized Algorithms*, chapter 4. Cambridge University Press, 1995.

[13] Napster. http://www.napster.com.

[14] M. Papadopouli and H. Schulzrinne. "seven degrees of separation in mobile ad hoc networks". In *IEEE GLOBECOM*, November 2000.

[15] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *ACM SIGMOBILE Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, October 2001.

[16] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, 2001.

[18] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Computer Communication Review*, 2001.

[19] unknown. The gnutell protocol specification v0.4. http://clip2.com/GnutellaProtocol04.pdf.

[20] M. Waldman and D. Mazi. Tangler: a censorship-resistant publishing system based on document entanglements. In *ACM Conference on Computer and Communications Security*, pages 126–135, 2001.

[21] M. Waldman, A.D. Rubin, and L.F. Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

APPENDIX

The probabilistic document indexing strategy results in a random distribution of indices across peers. In order to search for desired content, some subset of the peers must be queried. In this section, we concern ourselves with how the size of such a subset, denoted by s, affects p and the guarantees that a document will be indexed in the subset.

Consider an example in which we have $n = 100$ peers in the entire system and we wish to have $s = 10$. We want to be able to find a high percentage of the content, e.g. 95%, by searching no more than 10 nodes. Let r be the desired number of peers to index a particular document. We want an arbitrary document, d, to be indexed by at least 1 of the 10; so when $s = 10$, on average, $r = (1/10) \times 100 = 10$. Or, in another example, if we want $s = 5$, then d must be indexed by, on average, $(1/5) \times 100 = 20$ nodes and $r = 20$. Generally, $r = n/s$. The relationships are clear: as s decreases, r increases and each node is required to provide more storage and do more work answering queries.

Chernoff bounds [12] allow us to consider deviation of a random variable from its expectation as follows.

$$Pr[X \leq (1-\delta)sp] \leq e^{\frac{-\delta^2 sp}{2}}, \ 0 < \delta \leq 1 \qquad (1)$$

The right-hand side of Eq. 1 provides an approximation of the probability that X deviates from the expected value by a factor of $1 - \delta$. We can use this equation to calculate a bound on the probability that at least one node in the subset has indexed the document by calculating the complement. We let $(1-\delta)sp = 0$, and therefore $\delta = 1$. The calculation is straightforward:

$$Pr[X \leq (1-1)sp] \leq e^{\frac{-(1^2)sp}{2}}$$
$$Pr[X \leq 0] \leq e^{\frac{-sp}{2}} \qquad (2)$$

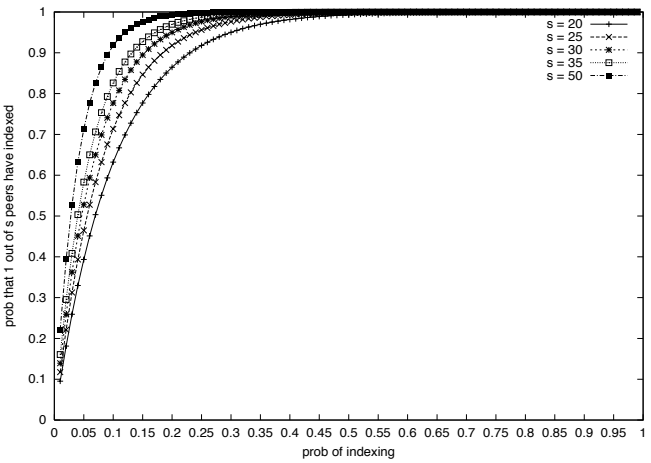

Fig. 8. The probability that 1 member of a subset has indexed a document.

Figure 8 shows the relationship between values of p, the probability of indexing, and the probability that one peer out of a subset of size s has indexed a document (the complement of Eq. 2). The x-axis shows the values of p in increments of 0.01; the y-axis is the probability that at least 1 out of s nodes has indexed a document. Each plot represents a different value of s.

As expected, the probability that 1 out of s nodes has indexed a document is greater for higher values of s. Consider the following examples. If we have $n = 100$ and want 5 nodes to store a document, we have, on average, $s = 20$. The value of p given by the Chernoff Bound is about 0.3. The corresponding probability that 1 node out of a subset of 20 has indexed the document is about 95%.