A Term-Based Inverted Index Partitioning Model for Efficient Distributed Query Processing

B. BARLA CAMBAZOGLU, Yahoo! Research
ENVER KAYAASLAN and SIMON JONASSEN, Yahoo! Research Barcelona
CEVDET AYKANAT, Bilkent University

In a shared-nothing, distributed text retrieval system, queries are processed over an inverted index that is partitioned among a number of index servers. In practice, the index is either document-based or term-based partitioned. This choice is made depending on the properties of the underlying hardware infrastructure, query traffic distribution, and some performance and availability constraints. In query processing on retrieval systems that adopt a term-based index partitioning strategy, the high communication overhead due to the transfer of large amounts of data from the index servers forms a major performance bottleneck, deteriorating the scalability of the entire distributed retrieval system. In this work, to alleviate this problem, we propose a novel inverted index partitioning model that relies on hypergraph partitioning. In the proposed model, concurrently accessed index entries are assigned to the same index servers, based on the inverted index access patterns extracted from the past query logs. The model aims to minimize the communication overhead that will be incurred by future queries while maintaining the computational load balance among the index servers. We evaluate the performance of the proposed model through extensive experiments using a real-life text collection and a search query sample. Our results show that considerable performance gains can be achieved relative to the term-based index partitioning strategies previously proposed in literature. In most cases, however, the performance remains inferior to that attained by document-based partitioning.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Web search engine, term-based index partitioning, distributed query processing, hypergraph partitioning

ACM Reference Format:
Cambazoglu, B. B., Kayaaslan, E., Jonassen, S., and Aykanat, C. 2013. A term-based inverted index partitioning model for efficient distributed query processing. ACM Trans. Web 7, 3, Article 15 (September 2013), 23 pages.
DOI: http://dx.doi.org/10.1145/2516633.2516637

15

1. INTRODUCTION

The massive size of today's document collections when coupled with the ever-growing number of users querying these collections necessitates distributed data storage and

B. Barla Cambazoglu acknowledges support from the Torres Quevedo Program from the Spanish Ministry of Science and Innovation, cofunded by the European Social Fund. S. Jonassen was supported by the iAD Centre (http://iad-centre.no) funded by the Research Council of Norway and the Norwegian University of Science and Technology. E. Kayaaslan is currently affiliated with Bilkent University; S. Jonassen is currently affiliated with the Norwegian University of Science and Technology.
Authors' addresses: B. B. Cambazoglu, Yahoo! Research; email: barla@yahoo-inc.com; E. Kayaaslan and C. Aykanat, Computer Science Department, Bilkent University; S. Jonassen, Department of Computer and Information Science, Norwegian University of Science and Technology.

DOI: http://dx.doi.org/10.1145/2516633.2516637

query processing [Cambazoglu and Baeza-Yates 2011]. Large-scale search services construct and maintain several search clusters composed of many compute nodes in order to increase their query processing throughputs and maintain reasonable query response times [Barroso et al. 2003]. Typically, the query processing throughput of the system is increased by replicating search clusters to exploit interquery parallelism. Queries are processed, in parallel, on the nodes of a selected search cluster. Query processing times are mainly determined by the number of nodes in the search cluster and the size of the document collection.

The most efficient way to process queries is to built an inverted index on the document collection and to process queries over this index [Zobel and Moffat 2006]. An inverted index maintains an inverted list for each term in the collection's vocabulary. Each inverted list keeps the ids of documents in which the lists's respective term appears as well as some auxiliary information (e.g., the within document frequencies of the term and the positions at which the term occurs in the documents).

In distributed text retrieval systems, the inverted index is stored in a distributed manner on the nodes of a search cluster. Each node runs an index server that facilitates access to the index stored in the node. A distributed index can be created by partitioning a full index based on the documents or terms.

In document-based index partitioning, each index server is assigned a nonoverlapping subset of documents, on which a local inverted index is built. A query is evaluated, in parallel, over all local indexes and a small number of best-matching results are returned by the index servers to a central broker, which then merges them to create a final answer set for the query. In general, query processing on document-based-partitioned indexes yields good load balance, low query response times, and high fault tolerance. However, the number of disk accesses incurred by a query grows linearly with the number of nodes in the search cluster. This may form a bottleneck for the query processing throughput if the inverted index is mostly stored on the disk [Badue et al. 2007].

In term-based index partitioning, the terms in the collection vocabulary are partitioned into a number of non-overlapping subsets, and each index server becomes responsible for a different subset of terms. The inverted index hosted by a server consists of the inverted lists corresponding to the terms assigned to the server. A query is processed only on the index servers that host at least one inverted list associated with the query. Since queries are typically very short, only a few index servers are involved in query processing. However, each index server needs to generate and transfer a potentially long list of documents that match the query. This communication overhead forms a performance bottleneck [Lucchese et al. 2007; Zhang and Suel 2007], together with the high computational load imbalance among the index servers [Moffat et al. 2006].

The focus of this work is on efficient query processing on distributed text retrieval systems where the index is partitioned based on terms. We propose a novel inverted index partitioning model that distributes inverted lists such that the lists that are likely to be accessed together are assigned to the same index servers. The model represents the access patterns to inverted lists by a hypergraph. We demonstrate that K-way partitioning of this hypergraph reduces the communication overhead incurred by queries issued to a K-node distributed text retrieval system. The model also involves a load imbalance constraint to maintain the computational load balance of the system during query processing.

To verify the validity of the proposed partitioning model, we conduct experiments over a Web document collection and a search query sample. According to our experimental results, the proposed index partitioning model achieves a significant reduction in the average number of index servers involved in processing of a query, relative to the term-based index partitioning strategies previously proposed in literature. The experiments also indicate considerable reduction in query response times and increase in query

processing throughput. Nevertheless, in most cases, the performance remains inferior to that attained by document-based partitioning.

The rest of the article is organized as follows. Section 2 provides some background on query processing over term-based-partitioned inverted indexes. Section 3 surveys the term-based index partitioning techniques proposed in previous works. Section 4 motivates the term-based index partitioning problem considered in this work and provides a formal problem definition. The proposed index partitioning model is presented in Section 5. The details of our experimental setup and dataset are given in Section 6. The experimental results are presented in Section 7. Section 8 concludes the article.

2. PRELIMINARIES

2.1. Term-Based Index Partitioning

An inverted index $\mathcal{L}$ contains a set of (term, corresponding inverted list) pairs, that is, $\mathcal{L} = \{(t_1, \mathcal{I}_1), (t_2, \mathcal{I}_2), \ldots, (t_T, \mathcal{I}_T)\}$, where $T = |\mathcal{T}|$ is the size of vocabulary $\mathcal{T}$ of the indexed document collection. Inverted lists are composed of postings, where each posting in $\mathcal{I}_i$ keeps some information about a document in which term t_i appears. Typically, this information includes the document's id and term's frequency in the document.

In a distributed text retrieval system with K nodes, postings of an index are partitioned among a set $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$ of K index servers. A term-based index partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$ is a partition of $\mathcal{T}$ such that

$$\mathcal{T} = \bigcup_{1 \leq \ell \leq K} \mathcal{T}_\ell \tag{1}$$

under the constraint

$$\mathcal{T}_\ell \cap \mathcal{T}_{\ell'} = \emptyset, \qquad \text{for } 1 \leq \ell, \ell' \leq K \text{ and } \ell \neq \ell'. \tag{2}$$

Based on partition Φ, every term subset $\mathcal{T}_\ell$ is uniquely assigned to an index server S_ℓ, and each index server S_ℓ constructs a subindex $\mathcal{L}_\ell$ as

$$\mathcal{L}_\ell = \{(t_i, \mathcal{I}_i) : t_i \in \mathcal{T}_\ell\}, \tag{3}$$

That is, the index servers are responsible for maintaining only their own sets of terms and hence all postings of an inverted list are assigned together to the same server. Typically, the partitioning is performed such that the computational load distribution on the index servers is balanced.

Given a term-based-partitioned index, queries can be processed in two alternative ways: the traditional central broker scheme or the pipelined query processing scheme. In the following two sections, we provide some background on these two alternative query processing schemes. Then, we describe some inverted list replication heuristics that are typically coupled with these schemes.

2.2. Traditional Query Processing Scheme

The traditional query processing scheme, which is previously considered in many studies [Tomasic and Garcia-Molina 1993; Ribeiro-Neto and Barbosa 1998; MacFarlane et al. 2000; Badue et al. 2001], involves a central broker that is responsible for preprocessing the user query and issuing it to the index servers in a selected search cluster. In this scheme, queries are processed as follows. First, the broker splits the original query $q = \{t_1, t_2, \ldots, t_{|q|}\}$ into a set $\{\widehat{q}_1, \widehat{q}_2, \ldots, \widehat{q}_K\}$ of K subqueries such that each subquery $\widehat{q}_\ell$ contains the query terms whose responsibility is assigned to index server S_ℓ, that is, $\widehat{q}_\ell = \{t_i \in q : (t_i, \mathcal{I}_i) \in \mathcal{L}_\ell\}$. Then, the central broker issues each subquery to its respective index server. Depending on the terms in the query, it is possible to have $\widehat{q}_\ell = \emptyset$, in which case no subquery is issued to S_ℓ.

Upon the receipt of a subquery $\widehat{q}_\ell$, index server S_ℓ accesses its disk and reads the inverted lists associated with the terms in $\widehat{q}_\ell$, that is, for each query term $t_i \in \widehat{q}_\ell$, inverted list $\mathcal{I}_i$ is fetched from the disk. The document ids in the fetched lists are used to determine the set of documents that match $\widehat{q}_\ell$. Typically, the matching can be performed in two different ways, based on the AND logic (the conjunctive mode) or the OR logic (the disjunctive mode). In case of AND logic, a document matches a query only if the document appears in all inverted lists associated with the query terms. In case of OR logic, a document is a match if it appears in at least one of the inverted lists associated with the query terms. Once matching documents are identified, they are assigned scores by some relevance function (e.g., BM25) using the statistical information stored in postings.[1] This typically involves summing up, for each matching document, the score contributions of the query terms. Finally, the matching documents are sorted in decreasing order of their scores to create a local ranking $\mathcal{R}_\ell$, which typically contains document ids and scores, for the subquery issued to index server S_ℓ. This partial ranking is then transferred to the central broker.

Each ranking $\mathcal{R}_\ell$ is only a partial ranking as the local scores do not include the contributions received from the query terms that are hosted by nonlocal index servers. To generate a global ranking of documents, all partial rankings are merged at the central broker. In case of OR logic, the merge operation involves summing up all local scores corresponding to the same document and sorting the final document scores in decreasing order. In case of AND logic, only the documents that appear in all partial rankings are considered for the final ranking. Finally, the central broker returns to the user the highest ranked (typically 10) document ids, potentially together with some additional information (e.g., snippets). This type of query processing has three performance drawbacks.

—*Communication overhead.* If the processing of a query is distributed across more than one index server, the servers involved in the processing have to transfer their entire partial ranking information to the central broker. Especially, if document matching is performed in the disjunctive mode, large volumes of data may need to be communicated over the network, increasing query response times.
—*Bottleneck at the central broker.* The merge operation at the broker may form a bottleneck if the query traffic volume is high or K is large. In such cases, the broker's queue, which temporarily stores the partial ranking information received from the index servers, may start to grow beyond an affordable limit.
—*Computational load imbalance.* If postings are not evenly distributed across the index servers, some index servers may be overloaded during the query processing while others are mostly idle. This, in turn, causes a throughput bottleneck in the overloaded index servers, eventually degrading the performance of the entire system.

2.3. Pipelined Query Processing Scheme

The pipelined query processing scheme is originally proposed by Moffat et al. [2007] and is later enhanced by Jonassen and Bratsberg [2010, 2012a, 2012b]. This scheme solves the bottleneck problem at the central broker and can also improve the computational load balance of the system. In this scheme, for a given query, the set of index servers that are responsible for processing the query is determined as in the traditional scheme. One of the responsible index servers is given the central broker role. This particular server determines a routing sequence for the remaining index servers on which the query will be processed. The same server is also responsible for obtaining the final

[1]The techniques discussed in this work are not affected by the scoring function being used.

result set and returning it to the user or to the part of the retrieval system responsible for presenting the search results.

The processing of the query proceeds by obtaining the local ranking of a server and combining it with the ranking received from the previous server in the routing sequence. The computation of local scores is similar to that in the traditional scheme. The scores are combined with previous scores in different ways, depending on the matching logic. In case of OR logic, each server simply updates received document scores using the information in the postings that the server maintains and/or inserts new entries, forming a new partial ranking with potentially more entries. In case of AND logic, the server intersects the received partial ranking with its local partial ranking, forming a new partial ranking with potentially fewer entries. The generated partial score information is then passed to the next index server in the sequence. The final server in the sequence extracts the top-ranked k documents and returns them to the index server that has the broker role for the query.

We note that the pipelined scheme, in a sense, sacrifices intraquery parallelism in exchange for interquery parallelism. As the role of the central broker is now distributed across all index servers, the result merging bottleneck in the traditional scheme is avoided. Moreover, pipelined query processing allows for more fine-grained load balancing. In case of AND logic, the total communication volume in the traditional scheme forms an upper bound on that of the pipelined scheme, whereas in case of OR logic, it forms a lower bound. Unfortunately, compared to query processing on document-based-partitioned indexes, the following issues still remain as problems.

—*Communication overhead.* The volume of data transferred between the index servers can be quite high if the partial rankings are very long or too many index servers are involved in processing of the query.
—*Number of hops.* For long queries, the routing sequence may involve many index servers that contribute to the processing of the query. Having many hops in the routing sequence may increase query response times.
—*Computational load imbalance.* The pipelined query processing scheme considerably improves the computational load balance of the system, relative to the traditional query processing scheme. However, the load imbalance remains as a problem at the micro scale.

2.4. Inverted List Replication

A problem common to both query processing schemes described above is high computational load imbalance, mainly caused by high skewness in inverted list sizes and list access frequencies. In particular, long and frequently accessed inverted lists are likely to cause load imbalance in query processing. One remedy to this problem is to replicate a small fraction of such load-intensive lists across all index servers [Moffat et al. 2006; Lucchese et al. 2007]. In this solution, whenever a replicated list is involved in query processing, a decision needs to be made to select an index server that will be responsible for processing the replicated list. The technique adopted in Moffat et al. [2006] assigns the responsibility of processing a replicated list to the currently least loaded server. This approach achieves good load balance, but it has a high potential to disturb the coherence of inverted list accesses. The technique in Lucchese et al. [2007] restricts the set of responsible servers to those that hold at least one non-replicated term of the query. This approach does not disturb the coherence of list accesses, but may result in relatively higher load imbalance values compared to the above-mentioned technique. In the replication strategy followed in Zhang and Suel [2007], the index servers are given a relatively larger storage budget so that a high fraction of inverted lists are replicated on multiple servers.

3. RELATED WORK

3.1. Term-Based Inverted Index Partitioning

The inverted index partitioning problem is investigated in a number of works. Herein, we restrict our focus only to related studies on term-based index partitioning [Tomasic and Garcia-Molina 1993; Jeong and Omiecinski 1995; Cambazoglu 2006; Cambazoglu and Aykanat 2006; Moffat et al. 2006; Lucchese et al. 2007; Zhang and Suel 2007], omitting previous work on document-based index partitioning [Tomasic and Garcia-Molina 1993; Jeong and Omiecinski 1995; Ma et al. 2002; Badue et al. 2007; Ma et al. 2011]. For a performance comparison between the two partitioning approaches, interested reader may refer to prior studies [Ribeiro-Neto and Barbosa 1998; MacFarlane et al. 2000; Badue et al. 2001; Cambazoglu et al. 2006; Jonassen and Bratsberg 2009], which mainly differ in their assumptions about the underlying retrieval architecture, document matching logic, ranking models, datasets, parameters, and experimental methodologies.

Tomasic and Garcia-Molina [1993] evaluate the performance of term-based index partitioning on a shared-nothing parallel architecture, where each node is assumed to have multiple I/O buses with multiple disks attached to each bus. The terms are evenly partitioned across the disks (it is not explicitly stated how a partition is obtained) and queries are processed under the AND logic assumption. The traditional query processing scheme is extended by a novel two-phase prefetching technique. In the first phase, an initial partial ranking is obtained from the index server that hosts the query term with the shortest inverted list or from the index server that covers the largest number of query terms. In the second phase, subqueries are issued to the remaining index servers together with this initial ranking. Every contacted index server intersects its local ranking with the provided initial ranking. This way, the volume of data communicated to the central broker is reduced.

Jeong and Omiecinski [1995] investigate the performance of different term-based partitioning schemes for a shared-everything multiprocessor system with multiple disks. In their main experiments, they use synthetically created document collections with varying inverted list size skewness and observe the impact of the skewness on the query processing performance. They propose two load balancing heuristics for term-based index partitioning. Their first heuristic distributes inverted lists to servers taking into account the number of postings in the lists so that each server keeps similar amounts of postings. The second heuristic, in addition to inverted list sizes, takes into account the access frequencies of the lists. Their simulations indicate that the query processing throughput of a term-based-partitioned index does not scale well with the query traffic volume if inverted list sizes are highly skewed.

Moffat et al. [2006] conduct a study to investigate the load imbalance problem in term-based-partitioned indexes, assuming the pipelined query processing scheme [Moffat et al. 2007; Webber 2007]. Their work evaluates some alternatives for estimation of index servers' query processing loads and adopts a simple "smallest fit" heuristic for load balancing, coupled with replication of frequently accessed inverted lists on multiple index servers. Although simulations indicate improved load balance, in practice, throughput values remain inferior to those in query processing on document-based-partitioned indexes due to unexpected peaks in load imbalance at the micro scale, that is, the load imbalance remains as an issue.

In an earlier work [Cambazoglu 2006; Cambazoglu and Aykanat 2006], we propose a hypergraph partitioning model that aims to reduce the communication overhead incurred in query processing on term-based-partitioned indexes. Similar to our currently proposed model, the previous model tries to balance the query processing loads of index servers. However, our previous model does not make use of query logs, which

provide valuable information about the access patterns of inverted lists. Moreover, unlike the currently proposed model, the previous model is computationally infeasible to construct, as it requires creating a very large hypergraph that represents the entire document collection and vocabulary, which can be quite large in practice.

Lucchese et al. [2007] propose a term-based index partitioning strategy that relies on a greedy heuristic aiming to assign terms cooccurring in queries to the same index servers. In their heuristic, the terms are iteratively assigned to the index servers, trying to optimize a performance objective that combines query processing throughput and average response time, scaled by a relative importance factor. Also, a small fraction of frequently accessed inverted lists are replicated on all index servers. Compared to random assignment and the bin packing approach proposed in Moffat et al. [2006], some improvement is observed in query locality, that is, on average, fewer index servers are involved in processing of a query.

Zhang and Suel [2007] formulate the term-based index partitioning problem as a graph partitioning problem, where query term pairs whose corresponding inverted lists' intersection size is small are tried to be assigned to the same index server. After obtaining an initial assignment of terms by graph partitioning, different greedy heuristics are used to replicate certain inverted lists on a subset of index servers based on the potential for reduction in the communication overhead. This approach is shown to reduce the volume of the data communicated to the central broker during query processing.

Several studies investigate the performance of query processing on term-based-partitioned inverted indexes in the context of P2P systems. Under the AND logic assumption, Li et al. [2003] investigate the use of Bloom filters for efficient intersection of inverted lists that are stored on different peers. Their approach significantly reduces the communication overhead incurred due to the transfer of inverted lists at the expense of having some false positives in the intersection of the lists. Suel et al. [2003] use a distributed top-k pruning technique in which only a small portion of the shortest inverted list has to be communicated between the peers. Their approach also considerably reduces the communication overhead, but it is limited to certain classes of scoring functions. Zhang and Suel [2005] investigate hybrid algorithms that combine the previously proposed Bloom filter and top-k pruning approaches, obtaining further reduction in the communication overhead.

3.2. Term-Based versus Document-Based Inverted Index Partitioning

There are a number of issues that lead to a trade-off between term-based partitioning and document-based partitioning. Herein, we provide a summary of the claims made until now in regard to these issues.

—*Hardware properties.* Term-based partitioning works well in distributed search systems connected with a fast network while the speed of disk accesses is more critical in case of document-based partitioning [Tomasic and Garcia-Molina 1993; Ribeiro-Neto and Barbosa 1998].
—*Parallelism.* Document-based partitioning provides better intraquery parallelism as the same query can be processed by all index servers, at the same time. Term-based partitioning, on the other hand, provides better interquery parallelism since multiple queries can be concurrently processed in the search system, mostly in the absence of intraquery parallelism [Ribeiro-Neto and Barbosa 1998].
—*Distributed index construction.* Assuming the documents are already uniformly distributed on the nodes, constructing a document-based-partitioned index is a relatively trivial task since each node can simply build a local index on its collection. Constructing a term-based-partitioned index via message passing, on the other hand,

requires coordination and communication among the nodes [Ribeiro-Neto et al. 1998; Kucukyilmaz et al. 2012].

—*Load imbalance.* Document-based partitioning leads to better computational load balancing compared to term-based partitioning [MacFarlane et al. 2000]. Although the pipelined scheme helps reducing the load imbalance at the batch-level, the load imbalance remains as a problem at the micro-scale due to the random bursts in the workloads of some index servers [Moffat et al. 2006].

—*Bottleneck at the broker.* The central broker is more likely to be a bottleneck in case of term-based partitioning than document-based partitioning [MacFarlane et al. 2000].

—*Collection statistics.* In document-based partitioning, the inverse document frequency information needs to be computed and communicated to the index servers. This is not needed in case of term-based partitioning [Badue et al. 2001].

—*Fault tolerance.* Document-based partitioning provides better fault tolerance than term-based index partitioning in case of node failures since the result quality is less likely to be affected from a node failure in case of a document-based-partitioned index [Barroso et al. 2003].

—*Memory footprint.* In case of term-based partitioning, less memory is needed to store the vocabulary of the collection because the vocabulary is distributed across the index servers [Moffat et al. 2006]. Similarly, in case of document-based partitioning, less memory is needed to store the document features (e.g., document length, PageRank) since this data structure is distributed across the index servers and the features associated with a particular document are stored in only one index server, instead of all index servers.

—*Scalability.* Document-based partitioning provides better scalability in terms of increasing number of index servers and collection size, compared to term-based partitioning [Moffat et al. 2007; Webber 2007].

4. REDUCING THE COMMUNICATION OVERHEAD

4.1. Motivation

The objective of this work is to devise a term-based index partitioning model that tries to reduce the communication overhead incurred in query processing while maintaining the computational load balance among the index servers. Following previous observations [Cambazoglu 2006; Cambazoglu and Aykanat 2006; Lucchese et al. 2007; Zhang and Suel 2007], the idea is to assign to the same index servers the inverted lists that are often accessed together while processing queries. In a sense, the goal is to group inverted lists into K clusters such that the lists that are frequently accessed together fall into the same clusters. This technique is expected to achieve reduction in the communication overheads of index servers due to the increased likelihood of early aggregation of document scores in index servers. Moreover, as the processing is expected to span fewer index servers, this technique can also reduce the number of hops in the pipelined query processing scheme. The largest gains in the communication overhead are obtained if all inverted lists associated with the query are available in one index server, in which case it suffices to communicate only the local top-k result set of the server.

4.2. Formal Problem Definition

We are given a set $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$ of queries,[2] a set $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ of terms in the collection vocabulary, an inverted index $\mathcal{L} = \{(t_1, \mathcal{I}_1), (t_2, \mathcal{I}_2), \ldots, (t_{|\mathcal{L}|}, \mathcal{I}_{|\mathcal{L}|})\}$, and a set $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$ of K index servers. Each query $q_j \in \mathcal{Q}$ is composed of a set of

[2]We use the term query to refer to individual occurrences of queries, not the query string.

terms in the collection vocabulary, that is, $q_j \subseteq \mathcal{T}$. Each term $t_i \in \mathcal{T}$ is associated with an access frequency f_i, that is, the number of queries that contain the term. Given these, we now provide a number of definitions and then formally state our problem.

Definition 1 (*Hitting set of a query*). For a term partition Φ, the hitting set $h(q_j, \Phi)$ of a query $q_j \in \mathcal{Q}$ is defined as the set of index servers that hold at least one term of q_j [Lucchese et al. 2007], that is,

$$h(q_j, \Phi) = \{S_\ell \in \mathcal{S} : q_j \cap \mathcal{T}_\ell \neq \emptyset\}. \tag{4}$$

Let $c(q_j, \Phi)$ denote the communication cost incurred when processing q_j under term partition Φ. Herein, we model the communication cost in two alternative ways. First, we assume that q_j incurs a communication cost only if the processing involves more than one index server. This is because if all inverted lists associated with q_j are available in the same index server, no partial score information needs to be transferred from the index servers as it is sufficient to communicate only the final top k scores. In this case, we can approximate the communication cost by

$$c(q_j, \Phi) = \begin{cases} 0, & |h(q_j, \Phi)| = 1; \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

Second, we model the communication cost by the number of network messages that contain some partial score information. For the traditional query processing scheme, this number is $|h(q_j, \Phi)|$ since every index server $S_\ell \in h(q_j, \Phi)$ communicates its partial scores to the central broker. Similarly, for the pipelined query processing scheme, exactly $|h(q_j, \Phi)|$ network messages are exchanged between the index servers (including the data transfer for the final result set). Hence, we can estimate the communication overhead as

$$c(q_j, \Phi) = |h(q_j, \Phi)|. \tag{6}$$

In processing every subquery, we can assume that each term $t_i \in \widehat{q}_j$ incurs a computational load proportional to its inverted list size [Gan and Suel 2009]. This implies that each term $t_i \in \mathcal{T}$ introduces a load $f_i \times |\mathcal{I}_i|$ in processing of all queries in $\mathcal{Q}$. Hence, the overall load $L_\ell(\Phi)$ of a server S_ℓ with respect to a given term partition Φ becomes

$$L_\ell(\Phi) = \sum_{t_i \in \mathcal{T}_\ell} f_i \times |\mathcal{I}_i|. \tag{7}$$

Definition 2 (ϵ-*Balanced partition*). Given an inverted index $\mathcal{L}$, a query set $\mathcal{Q}$, and a set $\mathcal{S}$ of servers, a term partition Φ is said to be ϵ-balanced if the computational load $L_\ell(\Phi)$ of each server S_ℓ satisfies the constraint

$$L_\ell(\Phi) \leq L_{\text{avg}}(\Phi)(1 + \epsilon), \tag{8}$$

where $L_{\text{avg}}(\Phi)$ denotes the average computational load of index servers.

Problem 1 (*Term-Based index partitioning problem*). Given a set $\mathcal{Q}$ of queries, an inverted index $\mathcal{L}$, a set $\mathcal{S}$ of index servers, and a load imbalance threshold $\epsilon \geq 0$, find an ϵ-balanced term partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$ that induces an index partition $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_K\}$ such that the total communication cost $\Psi(\Phi)$ is minimized, where

$$\Psi(\Phi) = \sum_{q_j \in \mathcal{Q}} c(q_j, \Phi). \tag{9}$$

5. TERM-BASED INDEX PARTITIONING MODEL

Our solution to the term-based index partitioning problem is based on hypergraph partitioning. Hence, we first provide some background on hypergraph partitioning before presenting the proposed model.

5.1. Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set $\mathcal{V}$ of vertices and a set $\mathcal{N}$ of nets [Berge 1985]. Each net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$. The set $\text{Pins}(n_j)$ of vertices connected by a net n_j are called the pins of net n_j. Each vertex $v_i \in \mathcal{V}$ has a weight w_i.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition if each part $\mathcal{V}_\ell$ is nonempty, the parts are pairwise disjoint, and the union of the parts gives $\mathcal{V}$. In Π, a net is said to connect a part if it has at least one pin in that part. The connectivity set $\Lambda(n_j, \Pi)$ of a net n_j is the set of parts connected by n_j. The connectivity $\lambda(n_j, \Pi) = |\Lambda(n_j, \Pi)|$ of a net n_j is equal to the number of parts connected by n_j. A net n_j with connectivity $\lambda(n_j, \Pi) = 1$ is referred to as an internal net. If the connectivity of a net is larger than one, it is referred to as a cut net. The set of cut nets of a partition Π is denoted by $\mathcal{N}_{\text{cut}}(\Pi)$.

A partition Π is said to be ϵ-balanced if each part $\mathcal{V}_\ell$ satisfies the balance criterion

$$W_\ell \leq W_{\text{avg}}(1 + \epsilon), \qquad \text{for } \ell = 1, 2, \ldots, K, \tag{10}$$

where the weight $W_\ell = \sum_{v_i \in \mathcal{V}_\ell} w_i$ of a part $\mathcal{V}_\ell$ is defined as the sum of the weights of the vertices in that part and W_{avg} is the average part weight.

The K-way hypergraph partitioning problem is defined as finding an ϵ-balanced partition Π that optimizes a cutsize function defined over the nets of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. The hypergraph partitioning problem is known to be NP-hard [Alpert and Kahng 1995]. There are several cutsize functions developed and used in the literature [Alpert and Kahng 1995]. The metrics used by our term-based partitioning model are the cutnet ($\chi_{\text{cut}}(\Pi)$) and connectivity ($\chi_{\text{con}}(\Pi)$) metrics, which are defined as

$$\chi_{\text{cut}}(\Pi) = |\mathcal{N}_{\text{cut}}(\Pi)| \tag{11}$$

and

$$\chi_{\text{con}}(\Pi) = \sum_{n_j \in \mathcal{N}} \lambda(n_j, \Pi). \tag{12}$$

The objective in the cutnet metric is to minimize the number of nets that span more than one part. The objective in the connectivity metric, on the other hand, is to minimize the total number of parts that are spanned by the nets in the hypergraph.

5.2. Model

In our model, we represent the interaction between the queries in a query set $\mathcal{Q}$ and the inverted lists in an inverted index $\mathcal{L}$ by means of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. In $\mathcal{H}$, each term $t_i \in \mathcal{T}$ is represented by a vertex $v_i \in \mathcal{V}$, and each query $q_j \in \mathcal{Q}$ is represented by a net $n_j \in \mathcal{N}$. Each net n_j connects the set of vertices representing the terms that constitute query q_j. That is,

$$\text{Pins}(n_j) = \{v_i : t_i \in q_j\}. \tag{13}$$

Each vertex v_i is associated with a weight $w_i = f_i \times |\mathcal{I}_i|$, which represents the total estimated computational load for processing t_i.

A K-way vertex partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of hypergraph $\mathcal{H}$ is decoded as a K-way term partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$ of $\mathcal{T}$. That is, each vertex part $\mathcal{V}_\ell$ in Π corresponds to the subset $\mathcal{T}_\ell$ of terms assigned to index server S_ℓ. Since the total weight W_ℓ of each part V_ℓ is equal to the total load $L_\ell(\Phi)$ of the corresponding index server S_ℓ, balancing the

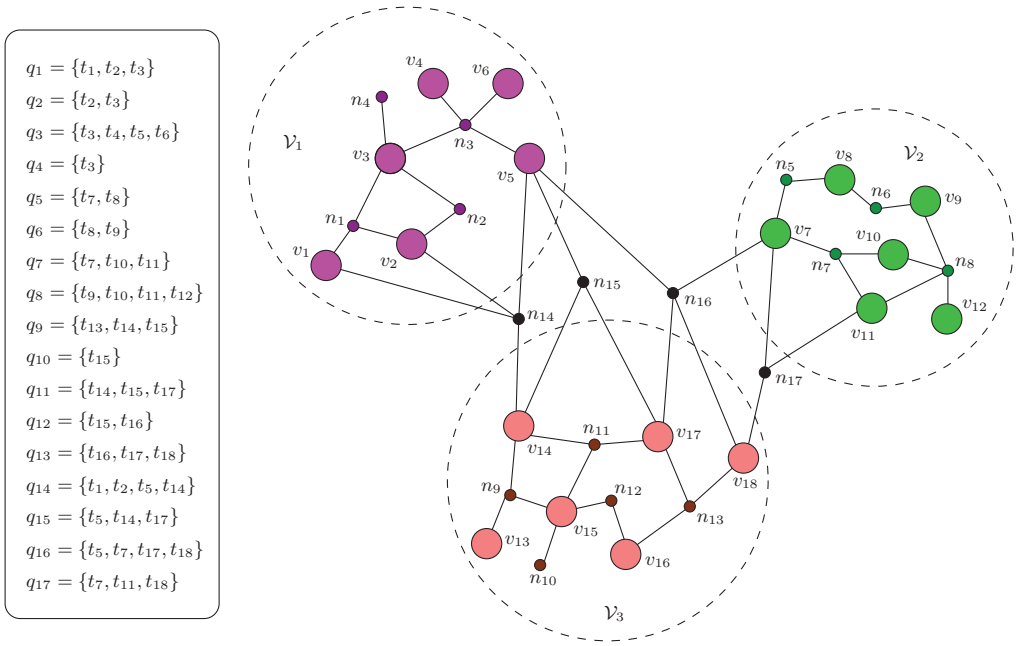


Fig. 1. A three-way partition of the hypergraph representing the relationship between a sample query log and an inverted index.

part weights according to the balance criterion in Equation (8) effectively balances the computational load among the index servers, satisfying the constraint in Equation (10).

In a K-way vertex partition Π of $\mathcal{H}$, consider a net n_j with connectivity set $\Lambda(n_j, \Pi)$. By definition, for each part $V_\ell \in \Lambda(n_j, \Pi)$, we have $\text{Pins}(n_j) \cap \mathcal{V}_\ell \neq \emptyset$, that is, $q_j \cap \mathcal{T}_\ell \neq \emptyset$. Thus, $\mathcal{T}_\ell \in h(q_j, \Phi)$ if and only if $V_\ell \in \Lambda(n_j, \Pi)$. This implies

$$h(q_j, \Phi) = \{\mathcal{S}_\ell \in \mathcal{S} : \mathcal{V}_\ell \in \Lambda(n_j, \Pi)\}, \tag{14}$$

which shows the one-to-one correspondence between the connectivity set $\Lambda(n_j, \Pi)$ of a net n_j in Π and the hitting set $h(q_j, \Phi)$ of query q_j in Φ, induced by Π. Hence, the minimization of the cutsize according to the cutnet (Equation (11)) and connectivity (Equation (12)) metrics accurately captures the minimization of the total communication cost $\Psi(\Phi)$ when $c(q_j, \Phi)$ is modeled by Equation (5) and Equation (6), respectively.

We illustrate the model with an example, involving a toy inverted index with vocabulary $\mathcal{T} = \{t_1, t_2, \ldots, t_{18}\}$, a query set $\mathcal{Q} = \{q_1, q_2, \ldots, q_{17}\}$, and three index servers ($\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$). Figure 1 shows the sample queries and a three-way partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\}$ of the toy index. We can interpret the figure as follows. Terms t_1 to t_6 are assigned to server $\mathcal{S}_1$; terms t_7 to t_{12} are assigned to server $\mathcal{S}_2$; and the remaining terms t_{13} to t_{18} are assigned to server $\mathcal{S}_3$. According to this assignment, the queries in sets $\{q_1, q_2, q_3, q_4\}$, $\{q_5, q_6, q_7, q_8\}$, and $\{q_9, q_{10}, q_{11}, q_{12}\}$ can be fully processed by servers $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$, respectively. For these queries, the servers communicate only their final top-k result sets, that is, the communication of the partial score information is not necessary. However, the remaining queries q_{14}, q_{15}, q_{16}, and q_{17} necessitate the communication of partial scores. In particular, during the processing of q_{16}, all three servers communicate their partial scores. For queries q_{14} and q_{15}, only servers $\mathcal{S}_1$ and $\mathcal{S}_3$, and for query q_{17}, only servers $\mathcal{S}_2$ and $\mathcal{S}_3$ need to communicate data. Consequently, if the cost of a query is modeled by Equation (5), the total communication cost is estimated as $\Psi(\Phi) = 13 \times 0 + 4 \times 1 = 4$. If it is modeled by Equation (6), we have $\Psi(\Phi) = 13 \times 1 + 3 \times 2 + 1 \times 3 = 22$.

We note that, unlike most of the work mentioned in Section 3, our work aims to capture the computational load balance and the minimization of the communication overhead at the same time. In this respect, among the existing models, the closest to ours are those proposed in Cambazoglu [2006], Cambazoglu and Aykanat [2006], and Zhang and Suel [2007]. The model proposed in [Cambazoglu 2006; Cambazoglu and Aykanat 2006] constructs a hypergraph using the inverted lists in the index. This approach does not scale well with increasing index sizes as it requires constructing and partitioning a very large hypergraph, where each term in the vocabulary of the collection is represented by a vertex and each document is represented by a net. In this hypergraph, a vertex is among the pins of a net if the document corresponding to the net contains the term corresponding to the vertex, that is, the number of pins is proportional to the number of postings in the index. In case of a Web-scale inverted index, where the number of documents is in the order of tens of billions and the average number of unique terms in a document is in the order of hundreds, the resulting hypergraph may contain trillions of pins. The existing hypergraph partitioning tools are not capable of partitioning such large hypergraphs. The approach of Zhang and Suel [2007] constructs and partitions a graph where each edge represents a pair of inverted lists. Each edge is weighted by the size of the intersection between corresponding inverted lists. For a Web-scale inverted index, this approach may be computationally infeasible due to the massive amount of list intersection operations that must be carried out.

The proposed model does not require accessing the inverted lists (except for their sizes). Hence, its scalability is independent of the size of the inverted index. Constructing our model, however, requires having information about the access patterns to the lists. In practice, this information can be obtained from the past query logs collected over a period of time. To account for the changes in the query stream, the model can be periodically reconstructed using a recent query log and the previously deployed index can be replaced by the newly generated index. Herein, we assume a similar mechanism, i.e., we build our model using a training query log and evaluate its performance over a more recent test query log.

We note that, in processing queries using a term-based-partitioned inverted index, there are many cost components that determine the overall performance, mainly the cost of disk seeks, disk data transfer, query processing, and network communication. We feel that it is difficult to come up with a theoretically sound index partitioning model that can capture all of these cost components at the same time. The proposed model simply aims to minimize the average hitting set size of queries, trying to gather coaccessed query terms on the same index servers. This approach indirectly reduces the communication volume while increasing the computational benefits due to query processing optimizations (e.g., skipping and MaxScore [Jonassen and Bratsberg 2012a]). The reason our model cannot directly capture the communication volume is because this information becomes available during the actual query evaluation, that is, it is not available during the model construction phase. Moreover, in the pipelined query processing scheme, the total communication volume incurred by a query depends on the query routing policy, which is again determined on-the-fly. We note that our model cannot capture the computational load of a query either. This is because the state of the result cache before a query is processed cannot be exactly known during the model construction phase. Our model can capture the computational load balance under the assumption that future access frequencies of inverted lists can be accurately estimated. Here, once again, the state of the posting list cache at the time of fetching a list from disk cannot be known by the model. But, the vertex weighting scheme used in the model can be modified to capture the fact that certain posting lists almost always reside in the posting list cache and hence do not incur any disk access overhead.

We limit the discussion provided in the rest of the article to the pipelined query processing scheme. Our model, however, is applicable to the traditional query processing scheme as well. Moreover, our model is not restricted by the logic used in document matching.

6. EXPERIMENTAL SETUP

In our experiments, we use about 25 million documents from the publicly available TREC GOV2 collection and 100,000 queries from the publicly available TREC Terabyte Track 2006 Efficiency Topics.[3] Both documents and queries are preprocessed using the same procedure, which includes case conversion, stemming, and stop-word removal. The number of queries that remain after preprocessing is 96,764. The size of the full inverted index built on the document collection is about 7GB. The inverted lists are compressed by the NewPFD algorithm [Yan et al. 2009]. The vocabulary contains 15,417,016 unique terms, each stored as a 20-character token with additional data (e.g., term and collection frequency, maximum score, term-to-node mapping, inverted file end offset). The main (global) lexicon stored on the broker node is 500MB and the short (local) lexicon maintained on each node is about 50MB. Both types of lexicons are kept in memory as sorted arrays and are accessed by binary search. The access cost is negligible compared to the cost of posting list processing. The broker node stores a memory-based array of maximum scores which contributes with another 200MB of data, but according to our observations in an earlier work [Jonassen and Bratsberg 2012a], this significantly improves the performance.

To create a more realistic setup, we assume the presence of a query result cache with infinite capacity [Cambazoglu et al. 2010]. We divide the cleansed query sample into three parts, each serving a different purpose. The first 60,000 queries are used to construct the proposed partitioning model. The next 10,000 queries are used to warm up the retrieval system. Finally, the following 20,000 queries are used in the actual performance evaluation. Due to the infinite result cache assumption, only the compulsory misses need to be evaluated using the inverted index. Hence, we consider only the unique queries in the model construction and evaluation steps. Moreover, we ignore a query if none of its terms appears in the vocabulary of the document collection.

The experiments are conducted on a nine-node PC cluster interconnected with a Gigabit network. One of the nodes is used as a broker, and the other nodes are used as workers. Each node has two 2.0 GHz quad-core CPUs, 8 GB of memory, and a SATA hard-disk. The query processing framework is implemented in Java.[4] In the implementation, we use the Okapi BM25 scoring model with skipping and MaxScore optimizations [Jonassen and Bratsberg 2012a].

To partition the constructed hypergraphs, we use the PaToH hypergraph partitioning tool [Aykanat et al. 2008; Catalyurek and Aykanat 1999] with its default parameters (except for the load imbalance constraint, which is set to 5%). PaToH uses the recursive bipartitoning (RB) framework to produce K-way partitions. In each RB step, it adopts the multilevel framework to obtain two-way partitions. Each RB step involves three phases: coarsening, initial partitioning, and uncoarsening. In the coarsening phase, various matching heuristic are used to cluster highly interacting vertices. During the uncoarsening phase, some FM-based iterative improvement heuristics are used to refine the bipartition found in the initial partitioning phase.

The running time of the FM-based heuristics is linearly proportional with the number of pins in the hypergraph. The matching heuristics used in the coarsening phase are

[3]TREC Terabyte Track 2006 Efficiency Topics, http://plg.uwaterloo.ca/ claclark/TB06.
[4]The code is available on GitHub: https://github.com/s-j/laika.

Table I. The Naming Convention Used in the Article

Type	Tag	Description
Inverted index partitioning strategy	CutHP	Proposed model optimizing the cutnet metric (Equation (11))
	ConHP	Proposed model optimizing the connectivity metric (Equation (12))
	Moffat	Bin packing approach in Moffat et al. [2006]
	Zhang	Graph partitioning approach in Zhang and Suel [2007]
	Lucchese	Partitioning heuristic in Lucchese et al. [2007]
	DP	Document-based partitioning with round-robin distribution
Server selection policy	NR	Inverted lists are not replicated
	LL	Currently least loaded server is selected
	HS	Index server is selected from those in the hitting set
Processing mode	AND	Queries are processed in conjunctive mode
	OR	Queries are processed in disjunctive mode

relatively expensive and they determine the overall running time. Their running times can be as high as the sum of the squares of the net sizes. In our particular problem, the size a net n_i is equal to the number of terms in the corresponding query q_i. Hence, the overall running time complexity of the hypergraph partitioning tool is $O(\log K \times \sum_{q_i \in \mathcal{Q}} |q_i|^2)$. The hypergraphs used in our experiments could be partitioned in only a few minutes.

We evaluate the proposed index partitioning model for the cutnet metric (Equation (11)) as well as the connectivity metric (Equation (12)). We refer to the corresponding strategies as CutHP and ConHP, respectively. As the baseline, we consider three previously proposed term-based index partitioning strategies. First, we consider the bin packing approach, where inverted lists are assigned to index servers in decreasing order of their expected loads (the "past L_t" strategy proposed by Moffat et al. [2006]). Second, we consider the graph partitioning strategy proposed by Zhang and Suel [2007]. To partition the constructed graphs, we use the MeTiS graph partitioning tool (version 5.0) [Karypis and Kumar 1998]. Finally, we evaluate the heuristic proposed by Lucchese et al. [2007]. We refer to these three strategies as Moffat, Zhang, and Lucchese, respectively. In addition to these, we consider document-based partitioning as another baseline. In our implementation of this partitioning strategy, herein referred to as DP, the documents are distributed to index servers in a round-robin fashion, that is, the documents are assigned to each index server in a circular order.

The aforementioned term-based partitioning strategies are coupled with index replication. In particular, we replicate the longest 100 inverted lists on all index servers, following the practice in Moffat et al. [2006]. The postings of the replicated inverted lists form 8.46% of all postings in the inverted index. We evaluate two different policies to select the index server responsible for processing a replicated list (see Section 2.4 for more details). In the first strategy [Moffat et al. 2006], the currently least loaded server is selected. In the second strategy [Lucchese et al. 2007], the responsible server is selected from those in the hitting set of the query. We denote these two server selection policies by the LL (least loaded) and HS (hitting set) tags, respectively. We also evaluate a policy where the inverted lists are not replicated. This policy is denoted by the NR (no replication) tag.

We adopt the pipelined query processing scheme in all strategies involving term-based partitioning. In query processing, we consider the conjunctive mode (AND logic) as well as the disjunctive mode (OR logic). The naming convention we follow in the rest of the article is summarized in Table I.

In our experiments, we reset the disk cache of the operating system and flush the memory before each experiment as an effort towards reading some portion of the

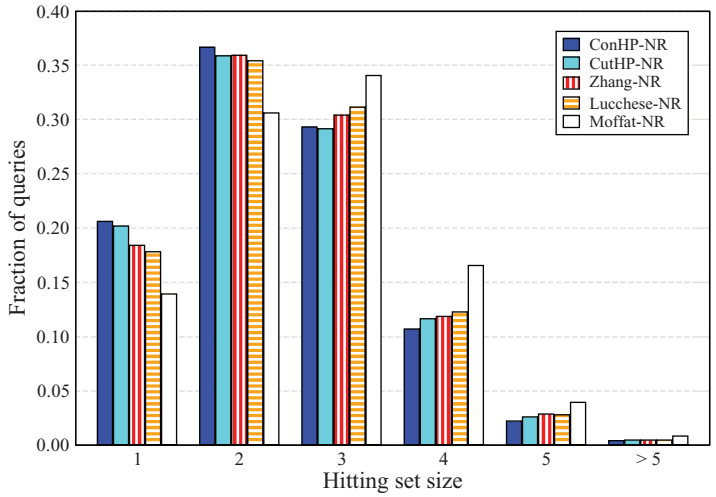


Fig. 2. Fraction of queries that have a certain hitting set size.

inverted list data from the disk. We repeat each experiment twice in alternating order of considered multiprogramming levels[5] (e.g., first $m = 1, 8, \ldots, 64$ and then $m = 64, 8, \ldots, 1$) and take the average of the values observed in the two runs. Unless otherwise stated, we set the number of index servers to eight in all experiments (i.e., $K = 8$).

7. EXPERIMENTAL RESULTS

7.1. Partitioning Quality

We first evaluate the partitioning quality of the competing term-based index partitioning strategies. Ideally, better strategies should lead to smaller hitting set sizes, that is, the query terms should span fewer index servers. In Figure 2, we display the fraction of queries which have a certain hitting set size. In case of `ConHP` and `CutHP` strategies, about one-fifth of the queries can be answered by only one index server, without requiring any communication for the intermediate score information. In general, the `ConHP` strategy, whose optimization objective is to minimize the hitting set size, performs slightly better than `CutHP`. Compared to the proposed strategies, the baseline strategies demonstrate relatively inferior performance in reducing the hitting set size. The worst performance is obtained by `Moffat`, which aims to improve the load balance instead of reducing the hitting set size. The results of this experiment indicate that the proposed model can effectively increase the likelihood that coaccessed inverted lists are stored on the same index server.

Figure 3 shows the hitting set sizes with respect to queries of different lengths, that is, the average number of index servers that are involved in processing queries of a certain length. Obviously, the hitting set size is one for all queries that contain a single term, independent of the employed partitioning strategy. In general, the proposed model seems to result in smaller hitting set sizes relative to the baseline strategies, independent of the query length. For queries containing more than seven terms, however, we observe a slightly different behavior in that the performance gap between different strategies becomes smaller. This is due to the presence of very long queries

[5]Multiprogramming level refers to the number of queries that are concurrently being processed in the system at any point in time.

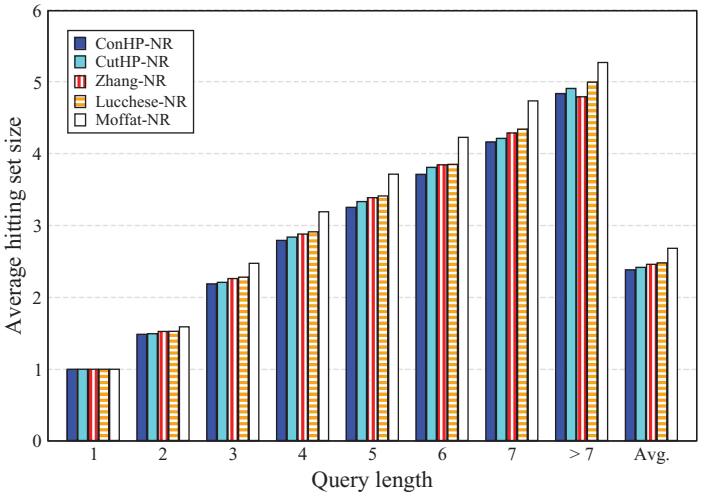


Fig. 3. Variation of the average hitting set size with query length.

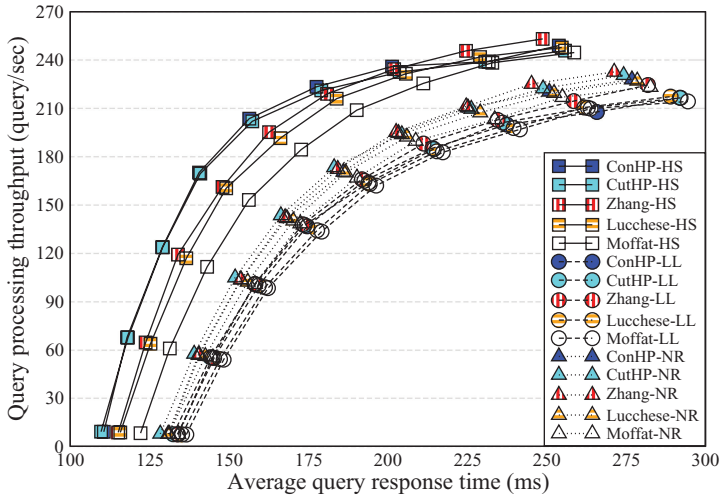
(i.e., $|q| \gg K$) whose processing spans all index servers, no matter how the index is partitioned. For such queries, the proposed model is less effective in reducing the hitting set size. On average (the last five columns in the figure), the hitting set sizes are 2.39, 2.42, 2.46, 2.48, and 2.69 for ConHP, CutHP, Zhang, Lucchese, and Moffat, respectively.

7.2. Performance Comparison of Term-Based Partitioning Strategies

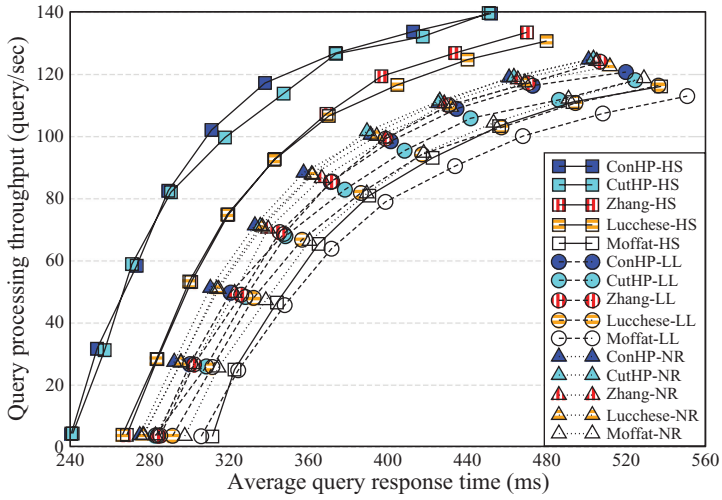
The goal of the experiments whose results are reported in this section is to evaluate the impact of the competing index partitioning strategies on the actual query processing performance. We evaluate the performance at different multiprogramming levels. In particular, we set the multiprogramming level m to 1, 8, 16, 24, 32, 40, 48, 56, and 64. The first data point in the curves corresponds to $m = 1$ while the last data point corresponds to $m = 64$. In the experiments, we evaluate all possible combinations of index partitioning strategies (ConHP, CutHP, Zhang, Lucchese, Moffat) with the policies for selecting an index server responsible for processing a replicated list (NR, LL, HS).

Figure 4(a) shows the average query response time and query processing throughput values observed at different multiprogramming levels, assuming the AND logic in query processing. According to the figure, independent of the selected index partitioning strategy, the HS replicated list selection policy always yields better performance than the LL policy. Interestingly, also the NR policy always gives better results than the LL policy. Under the LL or NR policies, the performance gap between the competing index partitioning strategies is relatively small. If the HS policy is employed, however, the performance gaps are more visible. Especially at lower multiprogramming levels, the performance of the proposed ConHP and CutHP strategies is better than the performance of the baseline strategies Zhang, Lucchese, and Moffat. At higher multiprogramming levels (56 and 64), Zhang performs slightly better than the rest of the strategies. The throughput values start to saturate around 250 query/sec.

In Figure 4(b), we report similar performance results for the OR mode. As expected, at the same multiprogramming levels, the OR logic results in much slower response times and lower throughput values than the AND mode. In general, when the HS policy is employed, the ConHP and CutHP strategies both achieve considerably better performance than the rest of the combinations. As an example, when $m = 24$, about seven more queries can be processed every second, on average, with an average response time



(a) AND mode.



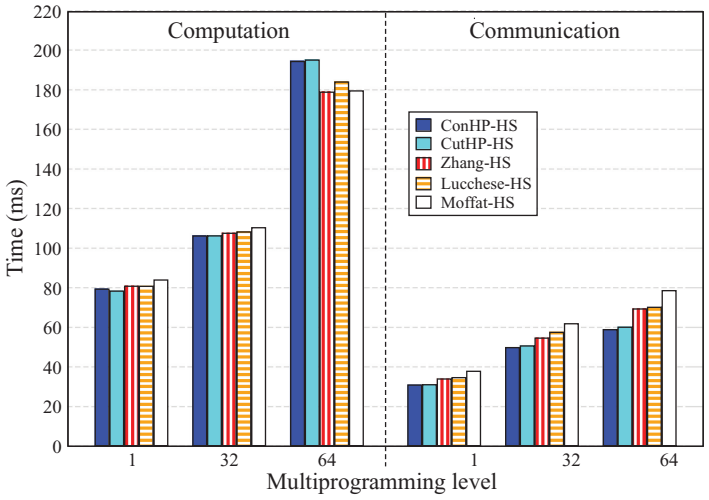
(b) OR mode.

Fig. 4. Average query response time versus query processing throughput at different multiprogramming levels ($m \in \{1, 8, 16, 24, 32, 40, 48, 56, 64\}$).

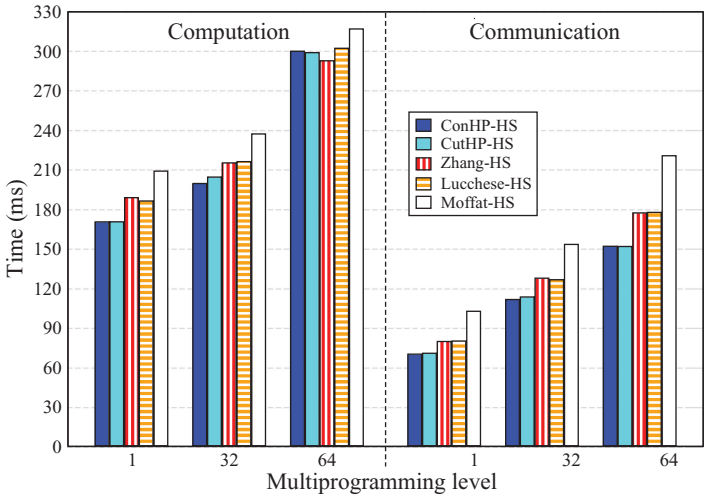
saving of 40 ms relative to the closest baseline Zhang. We also observe that ConHP performs slightly better than CutHP, but the performance gap is not significantly large.

In Figures 5(a) and 5(b), average query response times are dissected into two components as computation time and communication time, at selected multiprogramming levels ($m \in \{1, 32, 64\}$), for the AND and OR modes, respectively. The computation times include the time for fetching the inverted lists and scoring the documents. The communication times include the time for the preparation and transfer of the network messages.

According to Figure 5(a), the query response times in the AND logic are mainly determined by the computation time rather than the communication time. Especially at large multiprogramming levels (e.g., $m = 64$), the computation is the bottleneck as the load imbalance among the index servers becomes more pronounced. When $m = 64$,



(a) AND mode.



(b) OR mode.

Fig. 5. Computation and communication times at different multiprogramming levels ($m \in \{1, 8, 16, 24, 32, 40, 48, 56, 64\}$).

the computation times are relatively higher for the ConHP and CutHP strategies, which are less successful in load balancing. At lower multiprogramming levels, however, those two strategies lead to lower computation times as certain overheads are reduced by better placement of inverted lists on the index servers (e.g., less initialization overhead and better performance in query processing optimizations) and the computational load imbalance is relatively less important since the index servers are not heavily loaded. As expected, the ConHP and CutHP strategies lead to lower communication times since there are fewer hops in the communication and less data has to be communicated.

According to Figure 5(b), in case of OR logic, the communication is relatively more pronounced although the computation time still dominates the overall response time. The proposed strategies are still better in reducing the communication overheads.

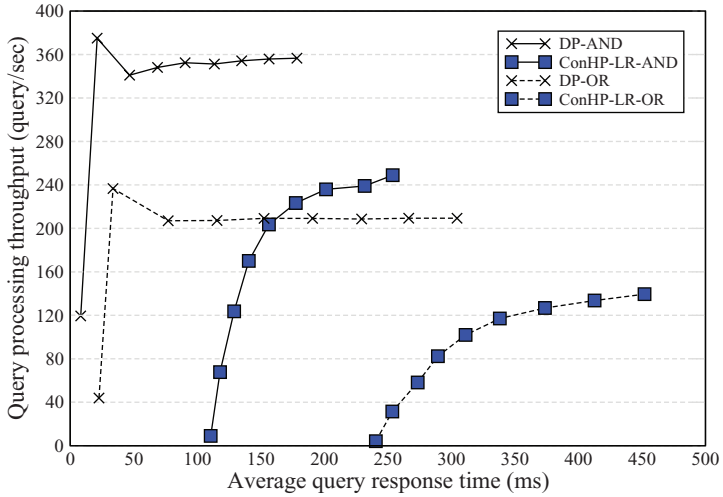


Fig. 6. Document-based partitioning versus the best performing term-based partitioning technique (ConHP-LR).

More specifically, when $m = 64$, the ConHP and CutHP strategies incur about 31% less communication overhead than Moffat and 14% less communication overhead than Zhang and Lucchese. Also, at low multiprogramming levels, some savings are observed in the average computation times.

7.3. Comparison with Document-Based Partitioning

In this section, we compare the best-performing term-based index partitioning strategy (ConHP) with document-based index partitioning (DP). In Figure 6, we display the query response times and throughput values observed at different multiprogramming levels, for both AND and OR modes. According to the figure, the throughput achieved by the DP strategy saturates at much lower multiprogramming levels. The decrease in the throughput as we increase m from 8 to 16 is potentially due to the bottleneck emerging in disk accesses as there are too many queries per index server. In general, if we compare the performance at the same multiprogramming levels, the performance attained by DP is quite superior to that attained by ConHP.

We next conduct experiments in which the test queries are of a certain length ($|q| \in \{1, 2, 3, 4, 5, > 5\}$). The results reported in Table II indicate that the superior performance of DP is mainly because the pipelined query processing scheme performs quite poorly for very long queries. The poor performance in case of long queries is mainly due to the high communication overhead and because the query processing optimizations are less effective. Moreover, in term-based partitioning, increasing query length tends to increase the load imbalance because the variation in inverted list sizes is higher. This implies much slower response times for long queries compared to document-based partitioning, which does a good job in balancing the query processing load across the index servers. With short queries (e.g., one- or two-term queries) and especially at high multiprogramming levels (e.g., $m = 64$), the performance gap between the proposed term-based partitioning model and DP is relatively smaller.

In Table II, it is also interesting to observe that, at high multiprogramming levels, two-term queries lead to better throughput values compared to single-term queries although single-term queries have a lower average response latency. This is mainly due to the effect of query processing optimizations (e.g., skipping), which are only applicable

Table II. Performance with Varying Query Length

		Response latency (ms)				Throughput (query/sec)			
		AND		OR		AND		OR	
m	$\|q\|$	CutHP-LR	DP	CutHP-LR	DP	CutHP-LR	DP	CutHP-LR	DP
1	1	50.9	11.1	50.9	11.1	19.6	89.3	19.6	89.3
	2	59.9	7.1	69.5	13.5	16.6	138.4	14.4	73.3
	3	111.1	8.8	133.6	18.1	9.0	111.5	7.5	54.7
	4	136.8	9.7	278.6	26.0	7.3	101.9	3.6	38.1
	5	151.3	10.7	481.1	36.1	6.6	92.2	2.1	27.6
	> 5	161.3	12.0	1,077.6	56.6	6.2	82.8	0.9	17.6
32	1	66.5	75.7	66.5	75.7	247.3	368.8	247.3	368.8
	2	88.9	71.5	97.9	94.1	348.5	439.2	304.4	334.1
	3	154.9	91.0	180.9	130.0	203.1	348.7	173.1	244.4
	4	206.5	118.6	373.0	184.6	149.2	267.6	82.9	172.4
	5	234.5	153.9	637.6	258.1	131.5	205.4	48.2	122.8
	> 5	271.3	197.4	1,394.8	392.4	111.3	158.6	21.7	80.3
64	1	86.6	122.8	86.6	122.8	216.6	349.7	216.6	349.7
	2	141.7	130.0	153.0	182.6	418.1	475.5	375.6	338.7
	3	245.9	179.8	277.7	252.5	253.4	349.8	223.1	249.4
	4	341.4	233.4	547.6	370.6	177.8	270.0	111.0	170.6
	5	386.9	301.5	898.9	506.7	155.7	207.0	66.3	123.7
	> 5	449.0	388.4	1,902.8	766.7	129.4	157.3	30.3	80.7

Table III. Performance with Varying Number of Index Servers

		Response latency (ms)				Throughput (query/sec)			
		AND		OR		AND		OR	
m	K	ConHP-LR	DP	ConHP-LR	DP	ConHP-LR	DP	ConHP-LR	DP
8	2	126.1	56.5	214.9	77.8	63.4	141.2	37.1	102.6
	4	114.7	38.1	236.7	55.3	69.6	209.6	33.7	144.2
	8	117.9	21.2	253.5	33.7	67.7	375.0	31.5	236.6
16	2	215.0	150.9	320.7	205.8	74.3	105.9	49.7	77.6
	4	138.3	93.1	265.1	136.7	115.4	171.5	60.0	116.9
	8	129.0	46.8	273.5	77.1	123.7	341.0	58.2	207.0
32	2	428.6	302.3	588.6	416.3	74.5	105.6	54.1	76.7
	4	229.3	184.2	387.2	268.6	138.5	173.2	81.7	118.9
	8	156.4	90.5	311.2	152.6	203.6	352.4	101.9	209.2
64	2	802.6	604.8	1,158.3	837.3	79.2	105.3	54.9	76.2
	4	462.8	360.5	727.9	535.8	136.4	176.5	86.6	119.0
	8	253.8	178.4	452.0	304.5	249.0	356.6	139.4	209.3

in case of multiterm queries. When processing two-term queries whose inverted lists are located in the same index server, the postings in the shorter list can be used to skip the postings in the longer list. Hence, such two-term queries can be processed faster than single-term queries, resulting in increased throughput. On the other hand, the average response latency of two-term queries is higher than that of single-term queries due to the larger communication overhead incurred in case of two-term queries whose lists are located in different index servers.

As a final experiment, we evaluate the performance of the ConHP and DP strategies with varying number of index servers ($K \in \{2, 4, 8\}$). According to the results in Table III, DP shows better scalability than ConHP with increasing number of servers. The main reason is the presence of intraquery parallelism in document-based partitioning, which prevents high load imbalance rates as the number of index servers

increases. On the other hand, interquery parallelism, which is available in term-based partitioning, does not help much against the increasing load imbalance. Hence, ConHP does not scale well with increasing number of index servers, especially when the multiprogramming level is low (see the decrease in the throughput values as we increase K from four to eight, when $m = 8$). However, it has relatively good scalability at high multiprogramming levels (e.g., $m = 64$). In general, we observe that the performance gap between ConHP and DP is lower at high multiprogramming levels and small number of index servers. Both strategies scale better under the AND logic assumption than the OR logic assumption.

We believe that term-based partitioning is likely to be less scalable compared to document-based partitioning with increasing collection sizes. This is because, in term-based partitioning, the inverted lists are assigned to index servers as a whole. Growing collection sizes imply larger inverted list sizes. Hence, we can expect to have high response latencies in online query processing. This is one of the reasons for commercial search engines to prefer document-based partitioning [Barroso et al. 2003].

8. CONCLUSION

We proposed a novel term-based inverted index partitioning model. The proposed model aims to reduce the communication costs incurred in query processing while maintaining the computational load balance of the index servers. Through extensive experiments, we demonstrated that the proposed model can yield better query processing performance compared to the term-based index partitioning strategies previously proposed in literature. Compared to document-based index partitioning, however, there is still a large performance gap that remains.

A possible extension to our work is a multiconstraint model, where the storage load imbalance can be captured as an additional constraint in the model. Another possible extension is to replace the adopted replication heuristic, which replicates frequently accessed inverted lists on all index servers, with the recently proposed techniques that couple hypergraph partitioning with replication [Selvitopi et al. 2012]. This may lead to further reduction in communication costs.

ACKNOWLEDGMENTS

We thank the anonymous reviewers, whose comments have greatly improved the quality of the article.

REFERENCES

Alpert, C. J. and Kahng, A. B. 1995. Recent directions in netlist partitioning: a survey. *Integration VLSI J. 19*, 1–2, 1–81.

Aykanat, C., Cambazoglu, B. B., and Uçar, B. 2008. Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices. *J. Parallel Distrib. Comput. 68*, 5, 609–625.

Badue, C., Ribeiro-Neto, B., Baeza-Yates, R., and Ziviani, N. 2001. Distributed query processing using partitioned inverted files. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval*. 10–20.

Badue, C. S., Baeza-Yates, R., Ribeiro-Neto, B., Ziviani, A., and Ziviani, N. 2007. Analyzing imbalance among homogeneous index servers in a web search system. *Inf. Process. Manage. 43*, 3, 592–608.

Barroso, L. A., Dean, J., and Hölzle, U. 2003. Web search for a planet: the Google cluster architecture. *IEEE Micro 23*, 2, 22–28.

Berge, C. 1985. *Graphs and Hypergraphs*. Elsevier Science Ltd.

Cambazoglu, B. B. 2006. Models and algorithms for parallel text retrieval. Ph.D. dissertation. Department of Computer Engineering, Bilkent University.

Cambazoglu, B. B. and Aykanat, C. 2006. A term-based inverted index organization for communication-efficient parallel query processing. In *Proceedings of the IFIP International Conference on Network and Parallel Computing*. 104–109.

Cambazoglu, B. B. and Baeza-Yates, R. 2011. Scalability challenges in web search engines. In *Advanced Topics in Information Retrieval*, Information Retrieval Series, vol. 33, Springer, 27–50.

Cambazoglu, B. B., Catal, A., and Aykanat, C. 2006. Effect of inverted index partitioning schemes on performance of query processing in parallel text retrieval systems. In *Proceedings of the 21st International Conference on Computer and Information Sciences*. 717–725.

Cambazoglu, B. B., Junqueira, F. P., Plachouras, V., Banachowski, S., Cui, B., Lim, S., and Bridge, B. 2010. A refreshing perspective of search engine caching. In *Proceedings of the 19th International Conference on World Wide Web*. 181–190.

Catalyurek, U. and Aykanat, C. 1999. Hypergraph-partitioning-based decomposition for parallel sparsematrix vector multiplication. *IEEE Trans. Parallel Distrib. Systems 10*, 7, 673–693.

Gan, Q. and Suel, T. 2009. Improved techniques for result caching in web search engines. In *Proceedings of the 18th International Conference on World Wide Web*. 431–440.

Jeong, B.-S. and Omiecinski, E. 1995. Inverted file partitioning schemes in multiple disk systems. *IEEE Trans. Parallel Distrib. Systems 6*, 2, 142–153.

Jonassen, S. and Bratsberg, S. E. 2009. Impact of the query model and system settings on performance of distributed inverted indexes. In *Proceedings of the Norsk Informatikkonferance*. 143–154.

Jonassen, S. and Bratsberg, S. E. 2010. A combined semi-pipelined query processing architecture for distributed full-text retrieval. In *Proceedings of the 11th International Conference on Web Information Systems Engineering*. 587–601.

Jonassen, S. and Bratsberg, S. E. 2012a. Improving the performance of pipelined query processing with skipping. In *Proceedings of the 13th International Conference on Web Information Systems Engineering*. 1–15.

Jonassen, S. and Bratsberg, S. E. 2012b. Intra-query concurrent pipelined processing for distributed full-text retrieval. In *Proceedings of the 34th European Conference on Advances in Information Retrieval*. 413–425.

Karypis, G. and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 1, 359–392.

Kucukyilmaz, T., Turk, A., and Aykanat, C. 2012. A parallel framework for in-memory construction of term-partitioned inverted indexes. *Comput. J. 55*, 11, 1317–1330.

Li, J., Loo, B., Hellerstein, J., Kaashoek, F., Karger, D., and Morris, R. 2003. On the feasibility of peer-to-peer web indexing and search. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*. 207–215.

Lucchese, C., Orlando, S., Perego, R., and Silvestri, F. 2007. Mining query logs to optimize index partitioning in parallel web search engines. In *Proceedings of the 2nd International Conference on Scalable Information Systems*. 43:1–43:9.

Ma, Y.-C., Chen, T.-F., and Chung, C.-P. 2002. Posting file partitioning and parallel information retrieval. *J. Syst. Soft. 63*, 2, 113–127.

Ma, Y.-C., Chung, C.-P., and Chen, T.-F. 2011. Load and storage balanced posting file partitioning for parallel information retrieval. *J. Syst. Soft. 84*, 5, 864–884.

MacFarlane, A., McCann, J. A., and Robertson, S. E. 2000. Parallel search using partitioned inverted files. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval*. 209–220.

Moffat, A., Webber, W., and Zobel, J. 2006. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 348–355.

Moffat, A., Webber, W., Zobel, J., and Baeza-Yates, R. 2007. A pipelined architecture for distributed text query evaluation. *Inf. Retriev. 10*, 3, 205–231.

Ribeiro-Neto, B. A. and Barbosa, R. A. 1998. Query performance for tightly coupled distributed digital libraries. In *Proceedings of the 3rd ACM Conference on Digital Libraries*. 182–190.

Ribeiro-Neto, B. A., Kitajima, J. P., Navarro, G., Sant'Ana, C. R. G., and Ziviani, N. 1998. Parallel generation of inverted files for distributed text collections. In *Proceedings of the 18th International Conference of the Chilean Society of Computer Science*. 149–157.

Selvitopi, R. O., Turk, A., and Aykanat, C. 2012. Replicated partitioning for undirected hypergraphs. *J. Parallel and Distrib. Comput. 72*, 4, 547–563.

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

Tomasic, A. and Garcia-Molina, H. 1993. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*. 8–17.

WEBBER, W. 2007. Design and evaluation of a pipelined distributed information retrieval architecture. Master's thesis. University of Melbourne.

YAN, H., DING, S., AND SUEL, T. 2009. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th International Conference on World Wide Web*. 401–410.

ZHANG, J. AND SUEL, T. 2005. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*. 225–233.

ZHANG, J. AND SUEL, T. 2007. Optimized inverted list assignment in distributed search engine architectures. In *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*. 1–10.

ZOBEL, J. AND MOFFAT, A. 2006. Inverted files for text search engines. *ACM Comput. Surv. 38*, 2, Article 6.

Received January 2012; revised August 2012, January 2013; accepted March 2013