Efficient Query Processing on Term-Based-Partitioned Inverted Indexes

Enver Kayaaslan[a,1], Cevdet Aykanat[a]

[a] *Computer Engineering Department, Bilkent University, Ankara, Turkey*

Abstract

In a shared-nothing, parallel text retrieval system, queries are processed over an inverted index that is partitioned among a number of index servers. In practice, the inverted index is either document-based or term-based partitioned, depending on properties of the underlying hardware infrastructure, query traffic, and some performance and availability constraints. In query processing on term-based-partitioned indexes, the high communication overhead incurred due to transfer of large amounts of information from index servers to the central broker forms a major performance bottleneck, deteriorating the scalability of the parallel text retrieval system. In this work, to alleviate this problem, we propose a novel combinatorial model that tries to assign concurrently accessed index entries to the same index servers, based on the inverted index access patterns extracted from past query logs. The model aims to minimize the communication overhead incurred by future queries, also enforcing a balance on workloads of index servers. We evaluate the performance of this model over a real-life text collection and a query log obtained from Yahoo!. Our results indicate significant reduction in the communication overhead, relative to a baseline strategy that only balances query workloads.

Keywords: parallel text retrieval, inverted index, term-based partitioning, parallel query processing, hypergraph partitioning

Email addresses: `enver@cs.bilkent.edu.tr` (Enver Kayaaslan), `aykanat@cs.bilkent.edu.tr` (Cevdet Aykanat)

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 query processing. 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. Typically, the query processing throughput of the system is increased by replicating search clusters to exploit inter-query parallelization. Processing of a query is carried out in parallel on the nodes of the search cluster. The number of nodes in the cluster and the size of the document collection together determine the response time to queries.

The most efficient way to process queries is to built an inverted index on the entire document collection and to process queries over this index. An inverted index maintains an inverted list for each term in the collection vocabulary. Each inverted list keeps the ids of documents in which the term corresponding to the list appears, together with some other auxiliary information (e.g., the frequency of the term in the document).

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

In document-based index partitioning, each index server is assigned a non-overlapping subset of documents, on which the server builds a local inverted index. 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 into 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 query processing throughput if the inverted index is mostly stored on the disk.

In term-based 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 posting lists corresponding to the terms assigned to the server. A query is processed only on the index servers that host at least one posting list associated with a query term. The index servers involved in processing the query return to the central broker their entire lists of documents matching the query terms they host. Since queries are typically very short, few index servers are involved in processing the query. Hence, term-based partitioning allows high concurrency, as multiple queries can be processed at the same time by different index servers. However, the large amounts of data transferred from index servers to the central broker forms a performance bottleneck, together with the poor load balance in query workloads of index servers.

The focus of this work is on term-based inverted index partitioning for the sake of efficient query processing. We propose a novel combinatorial model, based on hypergraph partitioning, to distribute the posting lists of an inverted index over a parallel text retrieval system. The proposed model assigns to the same index server posting lists that are likely to be accessed together[1], also trying to minimize the communication overhead that will be incurred by queries. In the mean time, the model tries to keep the load imbalance in query workloads of index servers as low as possible.

To verify the validity of the model, we conduct simulations using a real-life web document collection and a web query log obtained from Yahoo!. According to the results, the proposed index partitioning model achieves significant reduction in the fraction of queries processed by a single index server. We observe similar gains in the communication overhead, relative to a baseline partitioning strategy that only captures the workload balance.

The rest of the paper is organized as follows. Section 2 provides some background material. Term-based index partitioning techniques used in previous works are summarized in Section 3. Section 4 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. Experimental results are presented in Section 7. Section 8 concludes the paper.

2. Background

2.1. Term-based index partitioning

An inverted index contains a set of term and corresponding inverted list pairs $\mathcal{L} = \{(t_1, \mathcal{I}_1), (t_2, \mathcal{I}_2), \ldots (t_T, \mathcal{I}_T)\}$, where $T = |\mathcal{T}|$ is the size of the

[1]This information can be obtained from past query logs.

vocabulary $\mathcal{T}$ of the indexed document collection. Each posting $p \in \mathcal{I}_i$ keeps information about a document d_j in which term t_i appears. This information is used for ranking. In the simplest case, it includes a document id and the term's frequency in the document. In a parallel text retrieval system with K-processors, the postings of an inverted index is partitioned among a set of K index servers $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$. The partitioning is performed taking into account the computational load distribution on index servers.

In the term-based partitioning approach, each index server S_k locally keeps a subset $\mathcal{L}_k$ of the set $\mathcal{L}$ of all term and corresponding inverted list pairs, where

$$\mathcal{L}_1 \cup \mathcal{L}_2 \cup \ldots \cup \mathcal{L}_K = \mathcal{L} \tag{1}$$

with the condition that

$$\mathcal{L}_k \cap \mathcal{L}_\ell = \emptyset, \qquad \text{for } 1 \leq k, \ell \leq K, k \neq \ell. \tag{2}$$

In this partitioning technique, all processors are responsible for maintaining their own sets of terms, i.e., inverted lists are assigned to index servers as a whole. In the rest of the section, we provide some background on two alternative query processing schemes for term-based-partitioned inverted indexes.

2.2. Traditional query processing scheme

The traditional query processing scheme considered in many works (Tomasic and Garcia-Molina, 1993; Ribeiro-Neto and Barbosa, 1998; MacFarlane et al., 2000; Badue et al., 2001) involves a central broker, responsible for preprocessing the user query and issuing it to index servers in the search cluster. In this scheme, queries are processed as follows. First, the broker separates the user query $q = \{t_1, t_2, \ldots, t_{|q|}\}$ into a set $\{\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_K\}$ of K subqueries, in compliance with the way the index is partitioned. Each subquery $\hat{q}_k$ contains the query terms whose responsibility is assigned to index server S_k, i.e., $\hat{q}_k = \{t_i \in q \colon (t_i, \mathcal{I}_i) \in \mathcal{L}_k\}$. Then, the central broker issues the subqueries to index servers. Depending on the terms in the query, it is possible to have $q_k = \emptyset$, in which case no subquery is issued to S_k.

When an index server S_k receives a subquery $\hat{q}_k$, it accesses the disk and reads the inverted lists associated with the terms in $\hat{q}_k$, i.e., for each query term $t_i \in \hat{q}_k$, inverted list $\mathcal{I}_i$ is fetched from the disk. The information in the postings of fetched lists are used to first determine some documents that match $\hat{q}_k$. Typically, the matching may be performed in two different ways, based on the AND (disjunctive mode) or OR (conjunctive mode) logic. In case of the AND logic, documents that appear in at least one of the inverted lists related to the query is considered to be be a match, whereas a document is a match, in case of the OR logic, only if it appears in all inverted lists related to the query. Once the matching documents are found, they are scored by some relevance function (e.g., BM25) using the statistical information stored in the postings. This typically involves summing up for a document the score contributions of query terms. Finally, the documents are ranked in decreasing order of scores. We note that the techniques discussed in our work are independent of the scoring function used.

After S_k computes its partial ranking $\mathcal{A}_k$ (this typically contains document ids and scores), it transfers this information to the central broker. $\mathcal{A}_k$ is only a partial ranking because the document scores are not yet fully computed, as some terms that can contribute to the score of a document may be remotely stored by other index servers. To generate a global ranking of documents for the query, all partial rankings related to the query are gathered at the central broker. The central broker then merges the received partial rankings. In case of the OR logic, merging involves summing up the score entries corresponding to the same document and sorting the final scores in decreasing order. In case of the AND logic, only the documents that appear in all partial rankings are considered for the final ranking. Finally, the broker returns to the user the highest ranked k document ids, potentially together with some additional information (e.g., snippets).

There are three performance problems in this type of query processing.

- High communication volume: If the processing of a query is carried out by more then one index servers, the servers have to transfer their entire partial ranking information to the central broker. Especially, if the document matching is performed in the conjunctive mode, this implies that large volumes of data need to be transferred over the network., which implies a significant increase in query response times.

- Bottleneck at the central broker: Due to the high volume of data communicated to the central broker, the result merging step in the broker may form a bottleneck, especially under high query traffic volumes or when K is large. In those cases, the broker queue that stores partial ranking information received from index servers may start infinitely growing, limiting the peak sustainable query processing throughput.

- Imbalance in workloads of index servers: If the posting entries are not evenly distributed among the index servers, some index servers may be overloaded with processing queries while others are mostly idle. This, in turn, causes a throughput bottleneck in 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), as a remedy to the bottleneck problem at the central broker and also to achieve better workload balance in the system. In this scheme, for a given query, the set of index servers that are responsible for processing the query are determined as in the traditional scheme. One of the selected index servers is given the broker role. This broker server determines a routing sequence for remaining index servers, on which the query will be executed. The same server is also responsible for obtaining the final top k results and transferring them to the user or the part of the system responsible for presenting search results.

The processing of the query proceeds by obtaining the local partial scores of a server and combining them with the scores received from the previous server in the routing sequence. The computation of local scores is similar to that in the traditional scheme. They are combined with the previous scores in different ways, depending on the matching logic. In case of OR logic, each server simply updates the received document scores using the information in the information in the posting entries 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 broker, which initiated query processing.

In a sense, the pipelined scheme sacrifices inter-query parallelism in exchange of intra-query concurrency. Since the role of the central broker is distributed across the index servers, the result merging bottleneck in the traditional scheme is mostly avoided. Moreover, pipelined query processing allows for more fine-grained load balancing. Finally, in case of the AND logic, the total volume of communication may be reduced, relative to the traditional scheme. Unfortunately, when the pipelined scheme is compared to query processing on document-based-partitioned indexes, the following continues to be problems.

- High communication volume: In case of the OR logic, the volume of data transferred between the index servers can be quite high. In practice, the volume is identical to that observed in the traditional scheme.
- High number of network hops: In case of long queries, the routing sequence may involve many index servers. The high number of network hops and messages may increase query processing times.
- Workload imbalance: THe pipelined scheme significantly improves the workload balance of the system, relative to the traditional approach. However, the imbalance remains as a problem at the micro-scale.

3. Previous Work

The index partitioning problem is investigated by a number of works. These works mainly differ in their assumptions about the underlying architecture, matching logic, and partitioning model. There are also differences in adopted ranking models, datasets, and experimental methodologies. Herein, we specifically focus on works related to term-based partitioning (Tomasic and Garcia-Molina, 1993; Jeong and Omiecinski, 1995; Ribeiro-Neto and Barbosa, 1998; MacFarlane et al., 2000; Badue et al., 2001; Cambazoglu et al., 2006; Moffat et al., 2006, 2007; Lucchese et al., 2007), omitting the findings about document-based partitioning (Tomasic and Garcia-Molina, 1993; Jeong and Omiecinski, 1995; Ribeiro-Neto and Barbosa, 1998; MacFarlane et al., 2000; Badue et al., 2001; Cambazoglu et al., 2006; Badue et al., 2007).

Tomasic and Garcia-Molina (1993) evaluate a simple term-based partitioning approach on a share-nothing parallel architecture where each node is assumed to have multiple I/O buses with multiple disks attached to each bus. In their approach, terms are evenly partitioned across the disks (though it is not explicitly stated how partitions are obtained) and queries are processed using the AND logic. The traditional query processing scheme is extended by a novel prefetching technique, composed of two phases. In the first phase, an initial partial ranking is obtained from the index server that hosts the query term with the shortest posting list or from the index server that covers the largest number of query terms. In the second phase, subqueries are issued to remaining index servers together with this initial ranking. In each contacted index server, the generated partial rankings are intersected with the provided initial ranking so that 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 posting list size skewness and observe the impact of this skewness on query processing performance. They propose two load balancing heuristics for term-based index partitioning. Their first heuristic distributes the lists to servers taking into account the number of postings in lists so that each server keeps similar amounts of posting entries. The second heuristic, in addition to posting list sizes, takes into account the access frequencies of lists. Their simulations indicate that, in terms of query processing throughput, term-based partitioning scales well with query traffic, especially in case of low skewness.

Ribeiro-Neto and Barbosa (1998) evaluate term-based partitioning for batch query processing on a shared-nothing parallel system. This work adopts a simple partitioning strategy where terms are evenly distributed to index servers in lexicographical order. The results confirm the previous work in that term-based partitioning is more feasible in case of a fast network.

MacFarlane et al. (2000) evaluate the term-based partitioning scheme on a shared-nothing parallel architecture. In their experiments, they use a real-life document collection with an index supporting position information. For posting list assignment, they try three different frequency criteria, which are tried to be balanced. This work is the first to observe the performance bottleneck in the central broker. Badue et al. (2001) repeats on a small number of processors a similar study to that of MacFarlane et al. (2000). The results favor term-based partitioning, especially when the number of processors is larger than the average number of terms in queries. Cambazoglu et al. (2006) conduct yet another study, with the assumption that the index is completely stored on the disk. It is shown that term-based partitioning has better throughput scalability with increasing number of processors, relative to document-based partitioning.

Moffat et al. (2007) propose a novel query processing approach for term-based-partitioned indexes. In the proposed approach, partial rankings are computed and passed among the set of nodes that host the query terms, in a pipelined fashion. This approach, although it may increase query processing times, is shown to achieve good throughput values relative to the traditional query processing approach, which relies on passing posting lists to a central broker, on term-based-partitioned indexes.

Moffat et al. (2006) conduct a separate study to investigate the load imbalance problem in pipelined query evaluation (Moffat et al., 2007) on term-based partitioned indexes. That work evaluates a few alternatives for estimation of nodes' workloads and used a simple "smallest fit" heuristic for load balancing, coupled with replication of most popular posting lists on multiple nodes. Although simulations indicate improved load balancing, in practice, the throughput values remained inferior to query processing on document-based-partitioned indexes due to unexpected peaks in load imbalance at micro-scale, i.e., the load balance problem is not completely solved.

Lucchese et al. (2007) propose a greedy heuristic that tries to assign terms co-occurring in queries to the same index servers to construct a term-based-partitioned index. In their heuristic, the terms are iteratively assigned to parts trying to optimize a performance objective function that combines query throughput and average query processing time, scaled by a relative importance factor. Compared to the baseline random assignment and bin packing approaches (Moffat et al., 2006), some improvement is observed in query locality, i.e., the number of index servers involved in processing is reduced, on average.

Zhang and Suel (2007) describe heuristics for term-based index partitioning and posting list replication, assuming the AND logic in query processing. The evaluated heuristics, based on graph partitioning and greedy assignment of terms to index servers, tried to assign posting lists that are likely to produce short intersections to the same processor. This approach is shown to reduce the volume of data communicated to the central broker during query processing.

4. Formal problem definition

Let f be a matching function that maps a query to a set of documents. That is, $f(q)$ represents the set of documents matching to query q. We are given a set $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$ of K servers, a vocabulary $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ of n terms and a stream $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$ of m queries. Each term $t_i \in \mathcal{T}$ is associated with a posting list $\mathcal{I}_i$ and each query $q_j \in \mathcal{Q}$ is composed of terms in vocabulary, i.e., $q_j \subseteq \mathcal{T}$. Given these definitions, we define a term partition Φ as follows.

Definition 1 (Term partition). *A term partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$ is a partition of vocabulary $\mathcal{T}$, where $\mathcal{T}_k$ refers to the set of terms assigned to server S_k and the local index $\mathcal{L}_k$ at S_k can be formulated as,*

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

Definition 2 (Hitting set of a query Lucchese et al. (2007)). *For a term partition Φ, the hitting set $h(q, \Phi)$ of a query $q \in \mathcal{Q}$ is defined as the set of servers to which there are some terms assigned in q, i.e.,*

$$h(q, \Phi) = \{S_k \in \mathcal{S} : q \cap \mathcal{T}_k \neq \emptyset\} \tag{4}$$

Let $c(q, \Phi)$ denote the communication overhead in processing q with respect to term partition Φ. Moreover, Let $\hat{q}_{jk}$ denote the subquery to be processed in server S_k for a query $q_j \in \mathcal{Q}$, i.e., $\hat{q}_{jk} = q_j \cap T_k$. For latency dominant systems, the communication overhead is determined by the number of messages and can be formulated as,

$$c(q, \Phi) = 2|h(q, \Phi)| \tag{5}$$

In systems, where communication volume is more determinant, the network overhead depends on the parallel query processing scheme. In the naive approach, the communication overhead can be modeled as,

$$c(q, \Phi) = \begin{cases} 0 & |h(q, \Phi)| = 1 \\ \sum_k |f(\hat{q}_k, \mathcal{D})| & o.w. \end{cases} \tag{6}$$

whereas for pipeline query processing scheme the network overhead can be approximated as the minimum size of the shortest matching set of subqueries Zhang and Suel (2007), i.e.,

$$c(q, \Phi) = \begin{cases} 0 & |h(q, \Phi)| = 1 \\ \min_k |f(\hat{q}_k, \mathcal{D})| & o.w. \end{cases} \tag{7}$$

In processing of every subquery $\hat{q}$, each term $t \in \hat{q}$ incurs a load proportional to its posting list size. This implies that each term $t_i \in \mathcal{T}$ introduces load $p_i \times |\mathcal{I}_i|$, where p_i is the count how many times t_i occurs in the query stream $\mathcal{Q}$. Hence, the overall load $L_k(\Phi)$ of a server S_k with respect to a given term partition Φ can be formulated as,

$$L_k(\Phi) = \sum_{t_i \in \mathcal{T}_k} p_i \times |\mathcal{I}_i| \tag{8}$$

Definition 3 (ε-balance on workload). *Given sets $\mathcal{S}$, $\mathcal{T}$ and $\mathcal{Q}$, a term partition Φ is said to achieve an ε-balance on workload if overall load $L_k(\Phi)$ of each server S_k satisfies following constraint,*

$$L_k(\Phi) \leq L_{avg}(\Phi)(1+\varepsilon) \tag{9}$$

where $L_{avg}(\Phi)$ refers to the average overall load of servers.

Problem 1 (Term-based index partitioning problem). *Given a set $\mathcal{S}$ of servers, a vocabulary $\mathcal{T}$ of terms with associated posting lists, a query stream $\mathcal{Q}$ and an balance parameter $\varepsilon \geq 0$, find a term partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$, which induces the index partitioning $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_K\}$ such that the ε-balance on workload is achieved while the total communication overhead $\Psi(\Phi)$ is minimized, where*

$$\Psi(\Phi) = \sum_{q_j \in Q} c(q_j, \Phi) \tag{10}$$

5. Term-based Index Partitioning Model

The proposed model formulates the term-based index partitioning problem as a hypergraph partitioning problem. Hence, we first provide an overview of hypergraph partitioning basics before presenting the proposed model.

5.1. Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$ Berge (1985). Each net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$. The set of vertices connected by a net n_j are called the pins of net n_j and represented as $Pins(n_j)$. The degree of a net n_j is equal to the number of vertices it connects, i.e., $deg(n_j) = |Pins(n_j)|$. Similarly, the nets connecting a vertex v_i are called the nets of a vertex. Each vertex $v_i \in \mathcal{V}$ is associated with a weight w_i and each net $n_j \in \mathcal{N}$ is associated with a cost c_j.$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition if each part $\mathcal{V}_k$ is nonempty, parts are pairwise disjoint, and the union of 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 Λ_j of a net n_j is the set of parts connected by n_j. The connectivity $\lambda_j = |\Lambda_j|$ of a net n_j is equal to the number of parts connected by n_j. A net n_j with connectivity $\lambda_j = 1$ is an internal net, whereas it is an external (or cut) net if its connectivity is more than one.

In Π, the weight of a part is equal to the sum of the weights of vertices in that part. A partition Π is said to be balanced if each part $\mathcal{V}_k$ satisfies the balance criteria

$$W_k \leq W_{\text{avg}}(1+\epsilon), \qquad \text{for } k=1,2,\ldots,K, \tag{11}$$

where the weight W_k of a part $\mathcal{V}_k$ is defined as the sum of the weights w_i of the vertices in that part, W_{avg} is the weight that each part should have in case of perfect balance, and ϵ is the maximum imbalance ratio allowed.

Given all these definitions, the K-way hypergraph partitioning problem Alpert and Kahng (1995) can be defined as finding a partition Π for a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ such that the balance criteria on part weights (Equation 11) is maintained while a function defined over the nets is optimized. There are several objective functions developed and used in the literature. The metrics used by the term-based partitioning model are the cutnet and λ connectivity metric defined as follows.

$$\chi(\Pi) = \sum_{n_i \in \mathcal{N}_{\text{cut}}} c_i, \tag{12}$$

$$\chi(\Pi) = \sum_{n_i \in \mathcal{N}} c_i \lambda_i, \tag{13}$$

5.2. Model

We represent the query set as a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$. In this representation, each term $t_i \in \mathcal{T}$ is represented by a vertex $u_i \in \mathcal{U}$. There introduced a net $n_j \in \mathcal{N}$ for each query $q_j \in \mathcal{Q}$. A vertex u_i is a pin of a net n_j if and only if term t_i appears in query q_j, i.e. $t_i \in q_j$, Note that the degrees of nets are quite small due to less number of terms appearing in a query (2.76 in average). Each vertex u_i is associated with a weight $w_i = p_i \times |\mathcal{I}_i|$ which refers to the load to be introduced by t_i. We assign a unit cost to each net n_j, i.e., $c_j = 1$.

In a K-way partition $\Pi = (\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K)$ of hypergraph $\mathcal{H}$, each vertex part $\mathcal{U}_k$ corresponds to the subset $\mathcal{T}_k$ of terms assigned to index server S_k. If a net n_j is cut, then query q_j incurs transfer cost of partial scores, otherwise it does not incur such network cost. In this model, balancing the part weights W_k according to the balance criteria in Eq. 11 effectively balances the workload among index servers in order to achieve ε-balance as discussed in Eq. 9. Minimizing the cost according to cutnet metric (Eq. 12)

Table 1: Fraction of queries

	Avg	Uniq	1	2	3	4	≥ 5
$\mathtt{S}_2$	2.76	0.15	0.12	0.36	0.31	0.14	0.08
$\mathtt{S}_3$	2.76	0.15	0.12	0.36	0.31	0.14	0.08

accurately captures minimizing number of queries incurring transfer cost of partial scores and thus approximates minimization of total communication overhead $\Psi(\Phi)$ when communication is modeled as in Equations 6 and 7. Whereas, in order to minimize the number of messages (when the communication overhead is modeled using Equation 5), minimizing the cost according to connectivity metric (Eq. 13) happens to correctly minimize total communication overhead.

6. Experimental Setup

We sampled 1.7M web pages predicted by a proprietary classifier as belonging to the `.fr` domain. We also sampled about 6.3M queries from the France (`FR`) front-end of Yahoo! web search during three consecutive days (query sets $\mathtt{S}_1$, $\mathtt{S}_2$, and $\mathtt{S}_3$). Queries in $\mathtt{S}_2$ are used as training set to obtain cooccurrance relations of terms, whereas $\mathtt{S}_3$ is used for evaluation.

To create a more realistic setup, we assumed a query result cache with infinite capacity Cambazoglu et al. (2010), i.e., only unique queries are used by the model and also in evaluation. Both for training $\mathtt{S}_2$ and test $\mathtt{S}_3$ sets, the queries of previous day, $\mathtt{S}_1$ and test $\mathtt{S}_2$ respectively, is used to warm up the result cache, where hitrate performs around 69%. Although filtered by the result cache, the occurance frequencies of terms in queries still present a clear power-low distribution as shown in (Skobeltsyn et al., 2008, Fig. 5). The query size distribution, calculated over miss queries, of the train and test query log is given in Table 1.

We used the hypergraph partitioning tool PaToH Catalyurek and Aykanat (1999) with its default parameters (except imbalance constraint, which we set as 5%) to partition the hypergraphs. We varied the number of parts such that $K \in \{4, 8, 12, 16\}$ and scaled the number of documents linearly with K, i.e., $K \times 100$K documents are used in a partitioning run with K servers. The number of unique terms collections are given in Table 2.

We resort to replication in order to moderate the skewed distribution of term occurences by placing most load-intensive 100 terms to all nodes Moffat et al. (2006) at a cost of 20%, 47%, 73%, and 100% increase in the total index

Table 2: Vocabulary size

K	index	voc
4	124M	4253647
8	249M	6927509
12	368M	9074506
16	492M	11200564

size for $K = 1,2,3$, and 4, respectively. Whenever a query containes replicated terms, processing tasks of replicated terms are assigned to servers online. We employed four different approaches all of which assignes all replicated terms of a query into one center. First approach (referred as `GLB`), inherited from Moffat et al. (2006), pickes the currently least loaded server, which in turn results in high load balance. This approach has high potantial to perturb coherency of query terms. Hence, a more conservative approach (referred as `LOC`) selects the currently least loaded server among the ones unreplicated terms of the query are assigned. Additionally, we employ two other approaches, referred as `MAX` and `MIN`, where we select the server to which the unreplicated query term with highest and lowest amount of postings is assigned, respectively. Whenever all the terms of a query are those of replicated ones, four approaches behave like `GLB`. Some load intensive terms can cause severe imbalance Moffat et al. (2006). However, replication of most-load intensive terms, when combined with dynamic load balancing technique, yields a good workload balance. Replicated terms are not taken into account when finding partitions.

Training set $\mathtt{S}_2$ is used to extract popularity values of terms in order to calculate estimated workloads. We assume that queries are processed in conjunctive (AND) mode. Hence, a query is processed only if all its terms occur in the vocabulary. The fraction of such queries ranges between 83% and 87% (over missed queries), where the amount differs according to size of collection. All reported results are given over miss and vocabulary queries.

As the baseline index partitioning technique, we used bin packing approach, where terms are assigned to servers as in greedy bin packing according to their load-intensivities. The baseline technique and the proposed hypergraph-partitioning-based model are denoted as `BIN` and `HP`, respectively. All reported results, related to HP-based model, are averages of five runs.

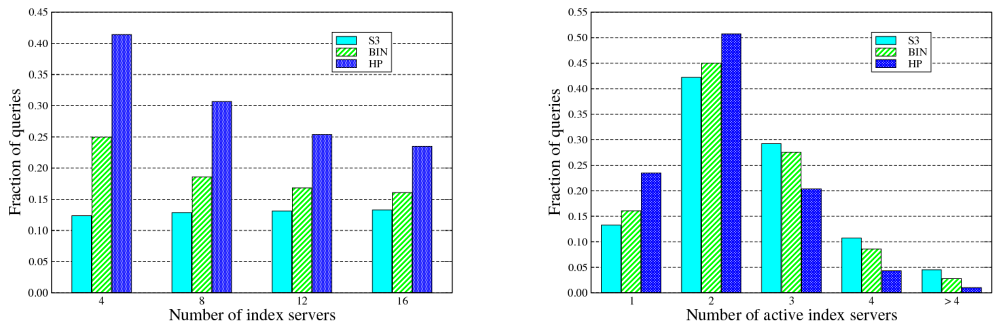

Figure 1: Fraction of locally processed queries (left) and fraction of queries with a given number of active index servers (right) among all queries.

7. Experimental Results

In the figures presented in this section, `BIN` represents the baseline approach based on bin packing, whereas `HP` implies the hypergraph-partitioning-based (HP-based) method where the objective is to minimize cutsize according to cutnet metric (Eq. 12).

Figure 1 (left) presents the fraction of locally processed, i.e., processed at single index server, queries among all. `S3` represents the hypotetical lower bound of locality due to single-term queries in test set $\mathtt{S}_3$. Bin packing approach (`BIN`) achieves considerably better locality than lower bound (`S3`), since the query sizes are not sufficiently small relative to number of index servers. Decreasing gap between two with increasing number of index servers verifies that reason. Locality decreases with increasing number of index servers, which is a natural result since query size remains same while the number of index servers increases. As seen in the figure, the HP-based approach effectively improve locality. We also note that locality relative to baseline also degrades which is because of that locality converges to theoretical lower bound with increasing number of index servers. Figure 1 (right) presents the fraction of queries with a given number s of index servers among all queries, for $s = 1$, 2, 3, 4 and $s > 4$ when the number of index servers is equal to 16. Moreover, `S3` represents the fraction of queries with a given number of terms among all queries (Skobeltsyn et al., 2008, Fig. 3). As seen in the figure, remarkably more queries process on less number of index servers when HP-based approch is applied.

Figure 2 illustrates savings (one figure for each cost model) in communi-

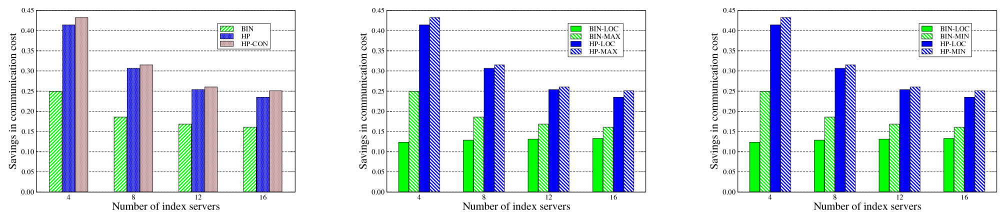

Figure 2: Savings in communication overhead where cost is modeled as in Equation 5 (left most), Equation 6 (middle), and Equation 7 (right most), as normalized to those of `BIN-GLB`.

cation overhead as normalized to those of bin packing approach with global randomization of replicated terms, referred as `BIN-GLB`. Figures reveal that huge savings in network overhead can be obtained by preserving term coherency of the queries. The savings decreases with increasing number of index servers for both baseline and HP-based method, which can be explained by a similar reasoning that of locality. The left most figure shows the normalized savings in the number of messages (Eq. 5). In this figure, `HP-CON` refers to the hypergraph partitioning objected to minimize cutsize according to connectivity metric (Eq. 13). As seen in the figure, hypergraph-partitioning-based approaches achieve significant savings in communication overhead in this scenario.

In Figure 2, the middle and right most ones represent normalized savings in communication overhead when modeled according to Equations 6 and 7, respectively. `BIN-LOC` and `HP-LOC` refer to the cases that local randomization of replicated terms is applied. Whereas the cases in which replicated terms of a query is processed on the index server with query term having most/least postings are referred as `BIN-MAX/MIN` and `HP-MAX/MIN`. The figure in the middle shows savings when queries are processed in the naive approach, whereas the right most figure presents how much saving can be foreseen in pipelined approach. Both figures confirm that important gains can be achieved in network bandwidth by proposed HP-based approach validating hypergraph partitioning model.

The improved savings in communication costs come at a cost imbalance on workloads. Table 3 illustrates how much degredation is observed at workload balance. Each row represents experiments with different number of index servers (K). Former five columns are results related to baseline bin packing approach whereas latter columns are for hypergraph-partitioning-

Table 3: Imbalance values

	Bin packing					Hypergraph Partitioning				
K	GLB	LOC	MAX	MIN	Storage	GLB	LOC	MAX	MIN	Storage
4	1.00	1.00	1.03	1.01	1.05	1.00	1.04	1.11	1.12	1.12
8	1.00	1.00	1.03	1.01	1.08	1.00	1.06	1.13	1.17	1.26
12	1.00	1.00	1.05	1.02	1.11	1.00	1.06	1.19	1.25	1.32
16	1.00	1.00	1.09	1.04	1.16	1.00	1.08	1.19	1.26	1.32

based method. For each of two groups, we present imbalance values for all four replicated-term assignment strategies. Fifth column of each group represents storage imbalance values that are side results of assignments. As seen in the figure, `GLB` approaches achieve perfect balance independent of the method (at a cost of higher communication), which also reflects the dominancy of the replicated-terms. LOC method behave similar to `GLB` method in bin packing approach, with conserving any locality that may occur for a query. Thanks to balancing constraint in the HP model, we observe satisfactory amount of workload imbalance (lower than 10%). Storage imbalances are important side results and observed at admissable amounts, i.e., up to 16% and 32% for bin packing and HP-based approaches, respectively.

8. Conclusions

References

Alpert, C. J., Kahng, A. B., August 1995. Recent directions in netlist partitioning: a survey. Integr. VLSI J. 19, 1–81.
URL `http://portal.acm.org/citation.cfm?id=214838.214841`

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

Badue, C. S., Baeza-Yates, R., Ribeiro-Neto, B., Ziviani, A., Ziviani, N., May 2007. Analyzing imbalance among homogeneous index servers in a web search system. Information Processing & Management 43, 592–608.
URL `http://portal.acm.org/citation.cfm?id=1224561.1224707`

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

Cambazoglu, B. B., Catal, A., Aykanat, C., 2006. Effect of inverted index partitioning schemes on performance of query processing in parallel text retrieval systems. In: ISCIS. pp. 717–725.

Cambazoglu, B. B., Junqueira, F. P., Plachouras, V., Banachowski, S., Cui, B., Lim, S., Bridge, B., 2010. A refreshing perspective of search engine caching. In: Proceedings of the 19th international conference on World wide web. WWW '10. ACM, New York, NY, USA, pp. 181–190.
URL `http://doi.acm.org/10.1145/1772690.1772710`

Catalyurek, U., Aykanat, C., July 1999. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst. 10, 673–693.
URL `http://portal.acm.org/citation.cfm?id=311796.311798`

Jeong, B.-S., Omiecinski, E., 1995. Inverted file partitioning schemes in multiple disk systems. IEEE Transactions on Parallel and Distributed Systems 6 (2), 142–153.

Lucchese, C., Orlando, S., Perego, R., 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. InfoScale '07. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, pp. 43:1–43:9.
URL `http://portal.acm.org/citation.cfm?id=1366804.1366860`

MacFarlane, A., McCann, J. A., Robertson, S. E., 2000. Parallel search using partitioned inverted files. In: Proceedings of the 7th International Symposium on String Processing and Information Retrieval. IEEE Computer Society, Washington, DC, USA, pp. 209–220.
URL `http://portal.acm.org/citation.cfm?id=829519.830829`

Moffat, A., Webber, W., 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. SIGIR '06. ACM, New York, NY, USA, pp. 348–355.
URL `http://doi.acm.org/10.1145/1148170.1148232`

Moffat, A., Webber, W., Zobel, J., Baeza-Yates, R., June 2007. A pipelined architecture for distributed text query evaluation. Information Retrieval 10, 205–231.
URL `http://portal.acm.org/citation.cfm?id=1265488.1265490`

Ribeiro-Neto, B. A., Barbosa, R. A., 1998. Query performance for tightly coupled distributed digital libraries. In: Proceedings of the 3rd ACM Conference on Digital Libraries. DL '98. ACM, New York, NY, USA, pp. 182–190.
URL `http://doi.acm.org/10.1145/276675.276695`

Skobeltsyn, G., Junqueira, F., Plachouras, V., Baeza-Yates, R., 2008. Resin: a combination of results caching and index pruning for high-performance web search engines. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '08. ACM, New York, NY, USA, pp. 131–138.
URL `http://doi.acm.org/10.1145/1390334.1390359`

Tomasic, A., 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. PDIS '93. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 8–17.
URL `http://portal.acm.org/citation.cfm?id=382019.382408`

Xi, W., Sornil, O., Fox, E. A., July 2002a. Hybrid partition inverted files for large-scale digital libraries. In: Proceedings of the Digital Library: IT Opportunities and Challenges in the New Millennium. Beijing Library Press, Beijing, China, pp. 404–418.

Xi, W., Sornil, O., Luo, M., Fox, E. A., 2002b. Hybrid partition inverted files: experimental validation. In: Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries. ECDL '02. Springer-Verlag, London, UK, pp. 422–431.
URL `http://portal.acm.org/citation.cfm?id=646635.700061`

Zhang, J., Suel, T., March 2007. Optimized inverted list assignment in distributed search engine architectures. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International. pp. 1 –10.