Inverted Index Compression via Online Document Routing

Gal Lavee, Ronny Lempel, Edo Liberty, and Oren Somekh
Yahoo! Labs., Haifa, Israel
Email: {gallavee, rlempel, edo, orens}@yahoo-inc.com

ABSTRACT

Modern search engines are expected to make documents searchable shortly after they appear on the ever changing Web. To satisfy this requirement, the Web is frequently crawled. Due to the sheer size of their indexes, search engines distribute the crawled documents among thousands of servers in a scheme called *local index-partitioning*, such that each server indexes only several million pages. To ensure documents from the same host (e.g., `www.nytimes.com`) are distributed uniformly over the servers, for load balancing purposes, random routing of documents to servers is common. To expedite the time documents become searchable after being crawled, documents may be simply appended to the existing index partitions. However, indexing by merely appending documents, results in larger index sizes since document reordering for index compactness is no longer performed. This, in turn, degrades search query processing performance which depends heavily on index sizes.

A possible way to balance quick document indexing with efficient query processing, is to deploy *online document routing* strategies that are designed to reduce index sizes. This work considers the effects of several online document routing strategies on the aggregated partitioned index size. We show that there exists a tradeoff between the compression of a partitioned index and the distribution of documents from the same host across the index partitions (i.e., host distribution). We suggest and evaluate several online routing strategies with regard to their compression, host distribution, and complexity. In particular, we present a term based routing algorithm which is shown analytically to provide better compression results than the industry standard random routing scheme. In addition, our algorithm demonstrates comparable compression performance and host distribution while having much better running time complexity than other document routing heuristics. Our findings are validated by experimental evaluation performed on a large benchmark collection of Web pages.

WWW2011 Mar. 28 - Apr. 1, Hyderabad, India

Categories and Subject Descriptors

H.3.m [**Information Systems**]: Information Storage and Retrieval

General Terms

Algorithms

Keywords

Inverted Index, Index Compression, Index Partitioning, Document Routing, Online Algorithm

1. INTRODUCTION

Modern search engines serve enormous query loads over tens of billions of Web pages, and are expected to deliver fresh and relevant results to users in sub-second time. Due to the sheer size of the Web, search engines partition their index over thousands of servers, where each server typically processes only several millions documents. At query time, the top results retrieved from all servers are merged to produce the final results, which are returned to the user.

The main data structure used by search engines to efficiently map queries to Web pages is the *inverted index* [2, 23]. The inverted index contains a *postings list* for each unique term appearing in the corpus. The postings list of a term consists of the list of document identifiers[1] (docIds) containing it, which are typically sorted by increasing docId values. The list is represented by encoding the gaps (called *dGaps*) between successive docIds. Another data structure in an inverted index is the *lexicon*, or *dictionary*, which is a lookup table that for each term in the corpus, points to the postings list corresponding to it.

It is well known that index size has a major effect on query processing throughput. In addition to the direct reduction in memory and disk space, more compact indexes lead to savings in data transfers and increase the hit rate of memory caches [22, 21]. Therefore, a large body of work has focused on index compaction and compression methods. The structure described above leaves two main degrees of freedom for compression optimization. The first is the assignment of docIds to documents (also referred to as document reordering). The second is the actual encoding of the dGaps into bits (also referred to as dGap compression). The basic idea behind an effective docId assignment is to

[1]Although terms frequencies and offsets within the document occupy a major portion of modern inverted indexes, we focus here on the documents identifiers only.

assign "similar" documents with close docIds, hence, potentially reducing the dGaps since similar documents contain many common terms. Alas, the problem of finding the optimal docId assignment is NP-hard and so various heuristics have been proposed in the literature.

However, previous work has focused on compacting the inverted index of a single server, whereas large corpora are indexed over thousands of servers. A standard approach distributes the corpus in a random fashion across the servers, which in particular routes (with high probability) similar documents to different servers. While this adversely affects document reordering schemes, it creates a more balanced query processing workload across the servers and eases communication bottlenecks as pages of popular Web sites are scattered about evenly across the servers.

This work considers a real-time partitioned indexing scenario where newly arriving content must be indexed on some server and made available for search immediately upon arrival. This prohibits the application of time consuming state-of-the-art document reordering schemes. However, the partitioned setting allows a degree of freedom in the form of the policy that routes incoming documents to the various partitions. We thus apply *online document routing schemes* – instead of the industry standard random routing scheme – to reduce the aggregated size of the partitioned index. We further monitor and constrain the distribution of documents belonging to the same hosts across the partitions (referred to as *host distribution* hereafter). We tested our algorithms on the 25 million Web page TREC .gov2 collection. Our main contributions are the following:

- We introduce the idea of using online document routing for aggregated index size compression in locally partitioned indexes. As far as we are aware, this novel approach has not been studied previously.
- We propose and evaluate several online routing strategies with regard to their compression, host distribution, and complexity. We show that there exists a tradeoff between the compression of an partitioned index and host distribution across the partitions.
- In particular, we present a term-based routing algorithm. Adhering to a simple document generating model presented in [11], we show analytically that the algorithm provides better compression results than the industry standard random routing scheme. In addition, our algorithm demonstrates comparable compression performance and host distribution while having much better running time complexity than other document routing heuristics.

The rest of this work is organized as follows. Section 2 surveys related work. Section 3 formally defines our model, problem and metrics. Algorithms for document routing are presented in Section 4, with the term-based algorithm analyzed in Section 5. Our experimental results are reported in Section 6. We conclude in Section 7.

2. RELATED WORK

2.1 Index Partitioning

The sheer size of the Web, the enormous number of search queries, and the required low latency, enforce a distributed inverted index architecture [2, 4, 5, 23]. To support these requirements, both *distribution* and *replication* principles are applied. Replication (also known as *mirroring*) means making enough identical copies of the system so that the required query load can be served, and is beyond the scope of this work. Distribution means the way the inverted index is partitioned across a collection of nodes.

The two main strategies of partitioning an inverted index are *local index-partitioning* and *global index-partitioning* [4]. According to the local index-partitioning strategy (or *document based partition*), each node is responsible for a disjoint subset of documents in the collection. In the global index-partitioning strategy (or *term based partition*), terms are divided into subsets, such that each node stores postings lists only for a subset of the terms in the collection.

Due to various theoretical and practical considerations, modern large-scale search engines follow the local inverted index-partitioning strategy and distribute the documents across the nodes [4, 5]. Documents can be assigned to nodes using different policies. For example, the *hash distribution policy* allocates documents to nodes in a random fashion by hashing the documents' URLs to yield a node identifier [2]. Various other distribution policies such as *round-robin distribution* are also possible [14].

2.2 Inverted Index Compression

Here we focus on a single node of a local index-partitioning architecture. We consider a simplified model of an inverted index in which the postings list of term t holds the list of docIds containing t, sorted by increasing value. Denote the list by $d_1^t, d_2^t, \ldots, d_{n_t}^t$, where d_i^t denotes the docId of the i'th document containing t out of n_t such documents. The list is actually represented by encoding the first docId and the sequence of gaps (dGaps) between successive identifiers thereafter, i.e. $d_1^t, d_2^t - d_1^t, \ldots, d_{n_t}^t - d_{n_t-1}^t$. The two degrees of freedom available for compressing the size of the lists are (a) docId assignment; and (b) dGap encoding. As we focus on the former, we start by briefly reviewing the latter. dGap encoding techniques aim to compress a sequence of integers. The literature contains schemes that encode each gap individually, e.g. Gamma, Delta, Golomb-Rice [20] and Zeta [9] encodings, as well as schemes that encode certain blocks of gaps, e.g. PForDelta [22, 13] and Simple9 [1]. Additionally, the Interpolative Encoding scheme [15] is applied directly on the docIds rather than their dGaps, and works well for clustered term occurrences.

In general, the docId assignment problem seeks a permutation of the the documents that minimizes the inverted-index size under a specific dGap encoding scheme. As shown in [6], this problem is NP-hard and various heuristics are used to provide approximations.

The size of an inverted-index is a function of the dGaps. All effective dGap encoding techniques represent smaller numbers with fewer bits (about logarithmic in the number value). Hence, assigning docIds in a way which results in smaller dGaps is the key for better compression. This principle drives most works dealing with docId assignment, and they strive to assign close docIds to "similar" documents, i.e. documents that share many terms.

Technically, most works define a graph $G = (D, E)$, where D is the set of documents, and E is a set of edges representing the similarity between two documents $d_i, d_j \in D$. One line of work started by [16] traverses the graph G to find

the maximal weight path connecting all the nodes, assigning docIds accordingly. This is equivalent to the NP-Hard *traveling salesman problem* (TSP). Several TSP approximations were applied for docId assignment in [16, 7, 12]. In [16], a simple greedy nearest neighbors (GNN) approach is used to add one edge at a time. To reduce the computational load, [7] uses singular value decomposition (SVD) to reduce the dimensionality of the term-document matrix. To scale up TSP-based schemes [12] proposes a new framework based on computing TSP on a reduced sparse graph obtained through *locality sensitive hashing*.

In yet another line of work, the nodes of G are clustered according to their similarity and close docIds are assigned to the nodes (documents) within each cluster. A top-down approach is used in [8], where the whole collection is recursively split into sub-collections, inserting "similar" nodes into the same sub-collections. Then, the sub-collections are merged into an ordered group of nodes. A bottom-up approach called k-scan was proposed in [18]. A hybrid method which combines k-scan clustering and TSP for intra-cluster docId assignment is proposed by [6].

A different approach, which is both highly scalable and highly effective, was proposed for Web collections in [17]. It assigns docIds according to the lexicographically sorted order of the documents' URLs, utilizing the fact that URL similarity is a strong indicator of document similarity. The scheme was found to perform remarkably well on various Web collections indexed as a whole.

In all the aforementioned works, a heuristic of docId assignment or an encoding of dGaps were empirically tested against several collections and compared to the results of other works. In contrast, [11] analyzes the compressibility of a collection whose documents are generated by a simple probabilistic model in which terms are chosen independently from a given distribution.

2.3 Online Problems

Online algorithms propose solutions to problems where the input is not known in advance and must be processed as it arrives in a sequential "online" fashion [10]. At first glance, the document routing problem tackled in this paper resembles two classical online problems - the *Metrical Task System* (MTS) and the *L*oad Balancing (a.k.a. Job Scheduling) problems.

MTS is a general framework for online problems that generalizes several known online problems such as the paging problem and the k-servers problem [10]. An MTS is composed of two components. The first is a metric space $\mathcal{M} =< \mathcal{S}, \mathsf{d} >$, where $\mathcal{S}$ is a set of points in the space and $\mathsf{d} : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ is a metric over $\mathcal{S}$. The second component of an MTS is a set of tasks R. Each task $r \in R$ may be seen as a vector $< r_1, r_2, \ldots, r_{|\mathcal{S}|} >$ of costs where each entry, r_i, is the cost of processing the request in state $i \in S$. Since the requests must be processed in the order which they arrive, the cost of an MTS algorithm is then the cost of moving from one state to another plus the cost of processing each request in the current state, given by the formula:

$$ALG(\sigma) = \sum_{i=1}^{n} \mathsf{d}(ALG(i-1), ALG(i)) + \sum_{i=1}^{n} r_i(ALG(i)) \ ,$$

Here σ denotes a particular request sequence of size n and $ALG(i)$ denotes the state of the algorithms immediately before processing request i.

The *Load Balancing* problem seeks to assign arriving jobs requests to m possible machines (one machine per job). Each of the jobs is associated with a cost, and a load balancing algorithm's objective is to balance the cumulative cost of jobs assigned to each machine. We may think of a job j as a vector $r_j =< r_{j,1}, r_{j,2}, \ldots, r_{j,m} >$, where $r_{j,i}$ is the cost of job j on machine i [3].

The major difference between the document routing problem and either MTS or Load Balancing stems from the fact that the cost associated with appending document d to index partition j depends on the sequence of documents previously routed to j. This doesn't fit at all within the Load Balancing framework, and causes a factorial explosion of states in MTS that renders its computation infeasible and its competitiveness non-existent. Therefore, we cannot transfer the analytical results of MTS or Load Balancing to our setting. Nevertheless, load balancing approaches and intuitions may still be applied and assessed empirically (see Section 4).

3. FORMAL PROBLEM DEFINITION

The online problem setting we tackle is the following. The system consists of a document dispatcher and m (initially empty) index partitions. Documents, modeled as bags of words, arrive in some arbitrary sequence to the dispatcher (e.g. as the result of a large scale distributed crawl policy). Each document belongs to some Web-host. The dispatcher must immediately route each document to one of the m partitions, whereby that partition simply appends the document to its existing inverted index. Each index consists of (1) a dictionary, and (2) inverted lists per term. The inverted lists encode document IDs only, i.e. no intra-document offsets or any other payload.

For example, consider a document d containing terms $t_1, \ldots, t_k$ that is appended as the j'th document of partition ℓ. d will be assigned docId j on ℓ. Any of $t_1, \ldots, t_k$ that has not appeared in any of the previous $j-1$ documents routed to ℓ will be added to ℓ's dictionary, and docId j will then be appended - by dGap encoding - to the k postings lists corresponding to the terms.

The dispatcher attempts to minimize the overall index size, i.e. the sum of index sizes on all partitions, while maintaining some balance between the number of pages from each host residing on each partition. Specifically, our algorithms will typically attempt to minimize the overall index size subject to some constraint on the skewness allowed in the distribution of same-host pages across the m partitions.

Since documents are routed and indexed one at a time (no buffering is allowed, either at the dispatcher or on the partitions), only single dGap encoding schemes apply. Specifically, our experiments use Delta encoding. Block-based dGap encoding schemes such as PForDelta and Simple9 are not applicable.

The following subsections formally define our metrics for index size and host distribution across partitions.

3.1 The Bits per Posting Metric

Let a corpus with $\mathcal{N}$ overall postings be indexed across m partitions, and let T_i denote the set of distinct terms on the i'th partition. Let t be a term appearing in n_t documents in some partition, and denote those docIds by $d_1^t < d_2^t < \ldots < d_{n_t}^t$. Assume all postings lists are encoded using Delta encoding. Then, the overall size of all postings lists on the

i'th partition, $\mathcal{P}_i$, is given by

$$\mathcal{P}_i = \sum_{t \in T_i} \left(\delta(d_1^t) + \sum_{j=2}^{n_t} \delta(d_j^t - d_{j-1}^t) \right) ,$$

where $\delta(k)$ is the length (in bits) of the Delta encoding of the integer k:

$$\delta(k) = 1 + \lfloor \log_2 k \rfloor + 2\lfloor \log_2(1 + \lfloor \log_2 k \rfloor) \rfloor . \qquad (1)$$

The overall size of the postings across the m partitions, $\mathcal{P}$, is defined as

$$\mathcal{P} = \sum_{i=1}^{m} \mathcal{P}_i .$$

We further define the overhead $\mathcal{OH}$ of a partitioned index as the space taken by the m dictionaries of the individual partitions. Each entry of the i'th dictionary is a pointer into the sequence of postings lists on the i'th server, and hence requires $\log_2 \mathcal{P}_i$ bits. Overall,

$$\mathcal{OH} = \sum_{i=1}^{m} |T_i| \log_2 \mathcal{P}_i .$$

Finally, the *bits per posting* metric comes in two flavors, with and without overhead. Those are simply $\frac{\mathcal{P}+\mathcal{OH}}{\mathcal{N}}$ and $\frac{\mathcal{P}}{\mathcal{N}}$, respectively.

3.2 Host-Distribution Measure

We evaluate the distribution of pages belonging to N_h hosts across m partitions using the *Chi squared* (χ^2) distribution [19]. Let $N_{h,i}$ denote the number of documents from host h on partition i, and denote by N_i the total number of documents on partition i. Also, let p_h denote the proportion of the documents belonging to host h of the total number of documents, N. The following *host-balancing* value B is distributed χ^2 with $(m-1)(N_h-1)$ degrees of freedom:

$$B = \sum_{i=1}^{m} \sum_{h=1}^{N_h} \frac{(N_{h,i} - N_i \cdot p_h)^2}{N_i \cdot p_h}$$

Essentially, this expression checks the null hypothesis that the host distribution resulted from a random routing of documents to the partitions. Intuitively, the value corresponding to a document routing scheme that approximately preserves the overall host distribution on each of the servers will be low.

When the number of degrees of freedom is large, the exact χ^2 distribution is difficult to compute. Hence we use a standard Normal approximation, where the mean is the number of degrees of freedom and the variance is twice this number. We will thus report the value

$$\frac{B - (m-1)(N_h-1)}{\sqrt{2(m-1)(N_h-1)}} ,$$

which is approximately distributed $\mathcal{N}(0,1)$.

4. ALGORITHMS

This section described several Algorithms for online document routing in the framework defined in Section 3.

Random document routing.

The *random* algorithm, commonly used in search engines, simply routes each incoming document to a partition chosen uniformly and independently at random.

Greedy document routing.

The *greedy* algorithm routes each document d to the partition whose inverted lists will grow by the least amount when appending d to its existing index.

Let d contain the set of terms T_d and denote by $\phi(j,t)$ the dGap created in the postings list of term $t \in T_d$ on partition $j, j \in \{1, 2, \ldots m\}$ upon appending d to the current state of partition j. Note that $\phi(j,t)$ ranges from 1 to the number of documents already routed to partition j (in case term t has not yet appeared there).

The change in the size of the inverted lists of partition j upon appending document d, $\Delta(j,d)$ is thus given by:

$$\Delta(j,d) = \sum_{t \in T_d} \delta(\phi(j,t)) , \qquad (2)$$

where $\delta(\cdot)$ denotes the Delta dGap encoding function (1).

Load-balancing document routing.

The *load balancing* algorithm is inspired by the relationship of our problem to the online load balancing problem of permanent jobs on unrelated machines. In this algorithm, which is based on [3], we start with an initial estimate L of the maximum load (i.e., the maximum aggregated index size). We then calculate the normalized *load* $\tilde{\ell}_{j,d} = \ell_{j,d}/L$ and normalized cost of adding the next document d, $C(j,d) = \Delta(j,d)/L$, for each partition $j = 1 \ldots m$. Here $\Delta(j,d)$ denotes the cost of adding document d to partition j (see equation (2)), and $\ell_{j,d}$ denotes the size of partition j assuming document d will be routed to it.

Next, we compute the following cost function for all partitions:

$$a^{\tilde{\ell}_{j,d}+C(j,d)} - a^{\tilde{\ell}_{j,d}} \text{ , where } a = 1 + \frac{1}{1.1} ,$$

and route the document to the partition of lowest cost. As more documents are added to the index, it is possible that the initial guess of the maximum load, L, will be exceeded. In this case we simply update our maximum load estimate to $2L$. More precisely, we check if $\ell_{j,d} + \Delta(j,d) > \beta L$, where β is a parameter (see [3]). For the unrelated machines load balancing problem, this algorithm promises an $O(\log m)$ competitive solution. Of course, as our document routing algorithm is different, this analytical guarantee does not hold.

Term based document routing.

The *term-based* algorithm associates a disjoint set of "representing" terms with every partition. Each new document is routed to the partition with which it has the most overlapping terms. Section 5 proves that this algorithm achieves better compression than random routing, the current de facto standard.

The algorithm requires a strategy for associating the terms in such a way that the probability of a particular term to "represent" each partition is about equal, as is the total probability mass of terms assigned to each partition. Due to the Power-law nature of term popularities, random association of terms to partitions creates partitions with imbalanced

probability mass of their assigned terms. We therefore used a heuristic approach which first sorts the terms in the corpus according to frequency[2], and then associates them in a "zig zag" pattern, i.e. the m highest frequency terms are associated with partitions $1..m$ in ascending order; the next m highest frequency terms are associated with partitions $m...1$ in descending order; and so on. After this initial phase, we use iterative swapping of the highest frequency term from the partition with the maximum cumulative frequency (over all terms associated with it) with the lowest frequency term from the partition with the minimum total frequency. This process is allowed to iterate until convergence of the difference between the maximal cumulative frequency and the minimal cumulative frequency.

Note that in practice there's no need to associate all terms in the corpus among the servers. Our experiments will leave most terms unassigned, effectively ignoring them when making routing decisions.

4.1 Adding Host-Balancing Constraints

We extend both the Greedy and Term-based routing algorithms by limiting the number of documents from each host that are assigned to any single partition[3]. The resulting routing schemes are called *Constrained Greedy* and *Constrained Term-based*, respectively.

For each host h of (estimated) size n_h, we bound the number of its documents which may be routed to any single partition. Then, upon arrival of a document of host h, the constrained algorithm (either Greedy or Term-based) simply considers only partitions on which the number of already-routed documents belonging to host h is below the bound.

We apply two flavors of bounds, denoted $b_1(h)$ and $b_2(h)$, to each host:

$$b_1(h) = \max\left\{\left\lceil \alpha\frac{n_h}{m}\right\rceil, 3\right\} \quad (3)$$

$$b_2(h) = \max\left\{\left\lceil \frac{n_h}{m} + \alpha\sqrt{\frac{n_h}{m}}\right\rceil, 3\right\} \quad (4)$$

The first bound flavor requires a slack value of $\alpha > 1$, to allow each partition to hold some multiplicative factor of its "fair share" of documents from host h. Our experiments use $\alpha \in \{1.05, 1.1, 1.2\}$. The second bound defines the slack in terms of the (approximate) standard deviation of the random routing of h's documents across the m partitions. Our experiments use $\alpha \in \{1, 2, 3\}$. With either flavor, we always allow partitions to hold 3 documents of each host. This relaxes our constraints over hosts with very few pages relative to the number of partitions.

4.2 Implementation Notes

The Random routing algorithm is obviously trivial to implement, with each routing decision being made by the dispatcher in $O(1)$. All other algorithms compute some value for routing document d to each of the m partitions, where the value depends on the set of terms T_d contained in d.

The *Greedy* and *Load Balancing* algorithms each compute a cost, which requires examining all $|T_d|$ terms in the document to be appended, for each of the m partitions. These computations can be done in a centralized fashion in the dispatcher, or in a distributed fashion with an extra round of communication between the dispatcher and each of the partitions.

For the centralized computation, the dispatcher must keep track of the number of documents routed to each partition, as well as the serial number of the last document routed to each partition for each term in the corpus. Effectively, this means keeping a global dictionary with m entries per term. Let $\mathcal{T}$ denote the number of distinct terms in the corpus. The centralized computation requires $O(m\mathcal{T})$ memory at the dispatcher and $O(m\,|T_d|)$ time upon dispatching d.

In the distributed computation, the dispatcher sends d to each partition, which computes its local cost and sends it back to the dispatcher, who then sends an indexing request to the "cheapest" partition. Each local cost computation requires $O(|T_d|)$ time and requires $O(\mathcal{T})$ extra memory per partition, as the last document ID per term needs to be kept in the local dictionary[4].

Turning to the *Term-based* algorithm, the centralized computation requires the dispatcher to keep track of the terms associated with each partition. This requires $O(\mathcal{T})$ space. The routing decision can then be achieved in $O(|T_d| + m)$ time while maintaining m additional counters, for overall space complexity of $O(\mathcal{T} + m)$. Therefore, comparing centralized implementations, this algorithm requires about m times less time and space than the *Greedy* and *Load Balancing* algorithms. This becomes significant as m grows. In the distributed computation, each partition can track its own set of associated terms at the cost of $O(\mathcal{T})$ space across all partitions, or roughly $O(\mathcal{T}/m)$per partition.

In addition to the above resources, the *Constrained* versions of the Greedy and Term-based algorithms must also store the N_h host size estimates (or bounds), as well as the number of pages per host that have been routed to each partition. This entails some extra $O(mN_h)$ memory. This memory can be kept at the dispatcher, or (in case of a distributed implementation) each partition may maintain its own counters for the number of pages per host it holds.

5. TERM BASED DOCUMENT ROUTING ALGORITHM - ANALYSIS

This section shows that the Term-based routing algorithm results in a smaller aggregated index size than that of the industry standard random routing scheme under the simplified document generation model of [11].

According to the model of [11], the N documents of the corpus may include up to L unique terms. Documents are independently generated as follows: each of the L terms is chosen independently with replacement, according to some frequency rank distribution $\{p_i\}$ (e.g., power-law or double-Pareto distributions) defined over the vocabulary T. The resulting corpus can be described by a $|T| \times N$ boolean term-doc matrix that has a 1 in entry (i, j) if term i is included in document j.

Let S_i denote the r.v. corresponding to the size of the postings list that encodes the ith row of the term-doc matrix (i.e. term t_i), when applying some single dGap encoding (e.g., Delta encoding). In addition, let $P_i \triangleq Pr(t_i \in d) = 1 - (1 - p_i)^L$ denote the probability that term t_i is included in document d. Moreover, for $1 \leq g \leq N$, let $X_{g,i}$ be the

[2]We assume that approximate term statistics are known from previous crawls.

[3]We assume here that the sizes of the hosts are approximately known due to previous crawls.

[4]While this ID is already encoded in the inverted index, decoding it requires decoding a long sequence of dGaps.

r.v. denoting the number of dGaps of length g in row i. According to [11, Thm. 2]

$$S(N,P_i) = \mathbb{E}[S_i] = \sum_{g=1}^{N} \delta(g)\mathbb{E}[X_{g,i}] \ , \tag{5}$$

where

$$\mathbb{E}[X_{g,i}] = P_i(1-P_i)^{g-1} + P_i^2(1-P_i)^{g-1}(n-g) \ , \tag{6}$$

and $\delta(g)$ is the number of bits needed to encode a dGap of length g. In particular, [11, Sec. 7] shows that $S(N,P_i)$ achieves the entropy bound for every term t_i that satisfies[5] $w\left(\frac{\log N}{N}\right) = P_i = o(1)$. Hence, for these terms the expected size can be approximated by[6] $S(N,P_i) \approx N\ H(P_i)$.

Let $\hat{T}$ denote the subset of terms that satisfy the conditions of [11, Sec. 7] and randomly divide it into m disjoint sets of size $|\hat{T}|/m$ (a set for each partition). We represent the sets by vectors $\{V_\ell\}_{\ell=1}^m$ with $V_\ell(i) = 1$ if t_i is included in the ℓth set and $V_\ell(i) = 0$ otherwise. A document d is assigned to the kth partition if it maximizes the dot product between the document term vector and the corresponding partition-defining term vector, i.e.

$$k = \arg\max_{\ell} \{V_\ell \cdot d\} \ . \tag{7}$$

We note that since the terms of $\hat{T}$ are randomly and evenly divided between the partitions, the m dot products $\{V_\ell \cdot d\}_{\ell=1}^m$ are identically distributed random variables.

Proposition 1 *For the above document generating model, the Term-based document routing algorithm achieves smaller expected aggregated index size than that of the Random document routing scheme.*

PROOF. We start by examining how the probability of a term being included in a document changes given that the document was assigned to a certain partition according to the Term-based routing algorithm. Focusing on the ℓth partition,

$$\begin{aligned} P_{i,\ell} \triangleq Pr(t_i \in d | d \in \ell) &\overset{(a)}{=} \frac{Pr(d \in \ell | t_i \in d)}{Pr(d \in \ell)} Pr(t_i \in d) \\ &= \beta_{i,\ell} P_i, \end{aligned} \tag{8}$$

where (a) is achieved by applying Bayes' theorem; $P_i \triangleq Pr(t_i \in d)$, and $\beta_{i,\ell} \triangleq Pr(d \in \ell | t_i \in d)/Pr(d \in \ell)$. We note that since the partition "representing" terms were divided randomly and equally among the partitions (hence, the decision dot products are identically distributed r.v.'s), then the probability that a document d is routed to the ℓth partition $Pr(d \in \ell) \approx 1/m$. Examining the definition of $\beta_{i,\ell}$ we observe that three cases should be considered: (1) the term t_i is a member of the set which "represents" the ℓth partition, i.e., $V_\ell(i) = 1$ - in this case it is easily verified that $\beta_{i,\ell} > 1$; (2) the term t_i is a member of the set which "represents" some other partition $\ell' \neq \ell$ - in this case it is easily verified that $\beta_{i,\ell} < 1$; and (3) the term t_i is not "representing" any partition $t_i \notin \hat{T}$ - in this case $\beta_{i,\ell} = 1$. Hence, the probability that a term t_i appears in a document which was routed to partition ℓ increases in case t_i "represents" the ℓth partition, decreases in case t_i "represents" some other partition, and remains unchanged otherwise. Moreover, since every document is always routed to one of the servers, we have that

$$\sum_{\ell=1}^{m} Pr(d \in \ell) P_{i,\ell} = P_i \approx \frac{1}{m} \sum_{\ell=1}^{m} P_{i,\ell} \ . \tag{9}$$

. Intuitively, this states that the overall probability of a term in the corpus remains unchanged regardless of the partitioning. We complete the proof by showing that the expected aggregated size difference between the Random routing and the Term-based routing setups is positive

$$\begin{aligned} \Delta &\overset{(a)}{\approx} \sum_{\ell=1}^{m} \sum_{t_i \in T} (S(N/m, P_i) - S(N/m, P_{i,\ell})) \\ &\overset{(b)}{=} \sum_{\ell=1}^{m} \sum_{t_i \in \hat{T}} (S(N/m, P_i) - S(N/m, P_{i,\ell})) \\ &\overset{(c)}{\approx} \sum_{t_i \in \hat{T}} \sum_{\ell=1}^{m} \frac{N}{m} (H(P_i) - H(P_{i,\ell})) \\ &= N \sum_{t_i \in \hat{T}} \left(H(P_i) - \frac{1}{m} \sum_{\ell=1}^{m} H(P_{i,\ell}) \right) \\ &\overset{(d)}{>} N \sum_{t_i \in \hat{T}} \left(H(P_i) - H\left(\frac{1}{m} \sum_{\ell=1}^{m} P_{i,\ell} \right) \right) = 0 \ , \end{aligned} \tag{10}$$

where (a) holds assuming each partition for both setups includes approximately N/m documents due to the *strong law of large numbers* and since partitions are homogeneous; (b) is achieved since we show that $P_{i,\ell} = P_i$ for all terms $t_i \notin \hat{T}$; (c) is achieved since $S(N,P_i) \approx N\ H(P_i)$ for $t_i \in \hat{T}$; (d) holds since $H(\cdot)$ is a strictly concave function; and the last equality is due to (9). □

To corroborate the results of the last Proposition, we generated a corpus using the document generation model of [11], selected the partition "representing" term sets, applied term based document routing, and appended the routed documents to their partitions using Delta encoding. The resulting aggregated bits per posting were compared to those calculated for an identical system using the Random document routing scheme. The comparison shows a consistent although minor reduction in the aggregated index size when Term-based routing is used. This small improvement, while obeying Proposition 1, is far from the 20% improvement achieved by the Term-based routing when applied to the .gov2 corpus (see Section 6). We conjecture that the performance gap stems from the simplified nature of the document generation model used to produce the synthetic corpus, as assuming independent documents of independent terms does not capture the inherent similarity between real documents belonging to the same host (the properties which make URL document ordering so successful [17]). Hence, the probability for .gov2 documents of the same host ending up in the same partition is much higher than that of two arbitrary documents generated by the model of [11]. Accordingly, as indicated by our experiments, the difference (10) is expected to be much larger for real life documents.

[5] Practically it means terms with not "too high" and not "too low" probability.

[6] Here $H(q)$ denotes the binary entropy function $H(q) = -q\log_2 q - (1-q)\log_2(1-q)$.

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

We use the TREC .gov2 Web corpus, a collection of about 25.2 million pages crawled from the gov domain, for the experiments[7]. After parsing, tokenizing, and removing all empty documents, we are left with 24.9 million documents across 17,000 hosts, 74.5 million distinct terms, and 5,705.2 million postings (distinct term appearances in documents).

Our experiments apply a random order of documents arrival and measure the resulting aggregated bits per posting metric for every algorithm (see section 3.1). As mentioned earlier, we use the Delta dGap encoding scheme throughout our experiments.

6.2 Index Compression Results

Figure 1 plots the Bits per Posting measure achieved by the different online algorithms as functions of the number of servers m. The left plot focuses on the inverted lists alone (without the overhead incurred by the dictionaries), and shows a downward trend in the number of bits per posting needed to encode the partitioned index as the number of servers increases. This trend was observed in all algorithms and is mainly an artifact of the additional degrees of freedom present when there are more partitions to choose from when routing. However, random routing also enjoys the increasing number of machines - this can be understood intuitively by considering the extreme case where there are as many servers as documents and each server uses only 1 bit to encode each posting.

Turning to the right side of the figure which takes into account the dictionary-induced overhead, there is an upward trend in the amount of bits per posting needed to encode the partitioned index. This is caused by overlapping term sets across the various partitions - overlapping terms require multiple dictionary entries, up to m entries in the worst (and realistic for many words) case.

The relative performance of the various algorithms is the same in both plots, and so the following discussion is applicable to both. At the two extremes, Random routing upper bounds the compression results, whereas Greedy routing trumps all algorithms and saves about a third of the space as compared with Random. It manages to decrease the overall index size (i.e. including overhead) for most of the growth in the number of partitions. The reason for this is that the greedy algorithm manages to assign documents with similar term sets to the same server, keeping the overlap in term sets across the partitions relatively small.

As we further discuss in Section 6.3, Greedy and Random routing also perform at the two extremes in terms of host distribution, but the Random performs best (quite intuitively) while Greedy performs worst, as same-host documents tend to share many terms and will thus tend to be routed by Greedy to the same small set of partitions, rather than uniformly across all servers.

The Load Balancing algorithm shows result very similar to those of the random algorithm, for a low number of partitions ($m < 200$). Around 200 servers, its curve starts to improve on that of Random's, achieving fewer bits per posting. For $m = 1000$, Load Balancing beats out the constrained versions of the Greedy and Term-based routing schemes, but still loses to the unconstrained versions.

The Term-based algorithm whose curve is plotted in Figure 1 is based on the association of about 8 million terms to the various partitions. The terms are those whose frequencies in the corpus range from 5 to 10^6. It achieves about 20% improvement in index size over the Random algorithm at $m = 1000$.

The constrained versions of both the Greedy and Term-based schemes attempt to decrease index size while not deviating too much from a balanced allocation of each host's pages across the partitions. Qualitatively, they achieve about the average compression of Random and their non-constrained version. Note that the curves shown in Figure 1 use found flavor $b_1(h)$ with $\alpha = 1.2$, as defined in Section 4.1. Section 6.3 will further analyze the constrained versions of the algorithms.

Other results (not presented here) reveals that the Term-based routing is not so sensitive with regards to the number of terms associated with the various partitions. Experiments with 6 and 4 millions terms of the unconstrained and constrained Term-based routing yields similar index sizes as those presented here for the 8 million terms version.

6.3 Host Distribution Results

This section further studies the trade-off between the quality of compression and the balanced routing of same-host documents across the different partitions. The conclusion from the data points given below is one of "no free lunch". Most of the redundancy in the document stream that is exploited by the better-compressing routing schemes stems from the ability to route same-host documents to a small set of partitions, rather than scattering them evenly across the partitions. This is yet another indication of why reordering Web collections by URL sorting[17] is so successful.

Table 1 shows the normalized Chi-squared host distribution values, as defined in Section 3.2, for the routing schemes presented in Figure 1, for several values of m. The schemes are listed in the order of their compression effectiveness, from worst (Random, left) to best (Greedy, right). Observe the following:

- Random routing, as expected, achieves values that are typical of a Normal$(0, 1)$ random variable.
- The tradeoff between compression (Figure 1) and host-balancing (Table 1) is near-perfect. The relative order of the schemes in terms of the two measures is almost entirely reversed. The sole exception is that the Constrained Greedy scheme outperforms the Constrained Term-based scheme in both measures[8].
- Whenever the Load Balancing scheme achieves random-level host-balancing ($m = 10, 40, 100$), it is basically similar to Random routing in terms of compression. Other than those three points, all other schemes distribute same-host pages with an imbalance that is statistically impossible to achieve with random routing.
- The constrained versions of the Greedy and Term-based schemes achieve a host-balancing value that is

[7]We note that since it was crawled almost to exhaustion, .gov2 is very "dense" and includes almost all relevant pages [12].

[8]The difference between Random and Load Balancing for $m = 10, 40$ is just a random fluctuation and does not contradict the above statement.

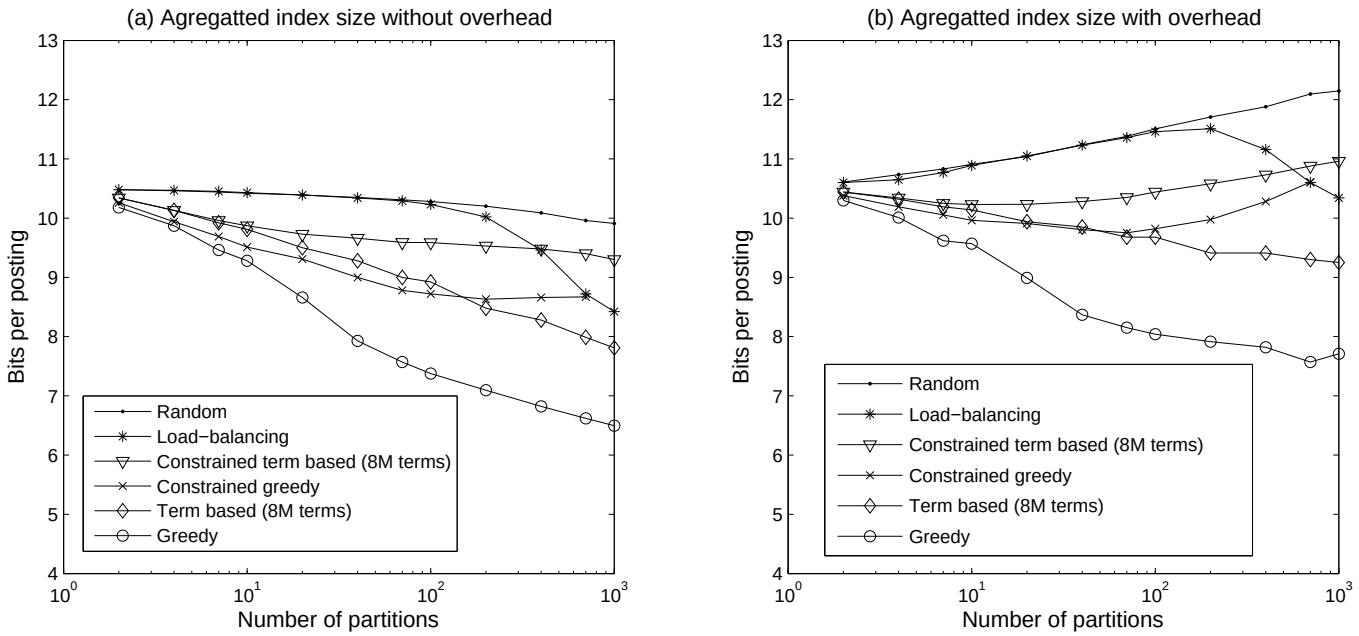

Figure 1: Bits per posting as function of the number of partitions for different document routing strategies and Delta encoding applied to .gov2 corpus, (a) without and (b) with overhead.

Scheme m	Random	Load Balancing	Constrained Term-based (8M)	Constrained Greedy	Term-based (8M)	Greedy
10	0.06	-0.47	3314	2050	77229	177828
40	0.96	-1.49	1607	1390	97814	389281
100	-1.31	2.59	734	825	123029	382995
400	0.79	303	897	427	111377	428874
700	-1.88	1820	1704	457	113754	433871

Table 1: Normalized host-distribution values per scheme, as functions of the number of partitions m

orders of magnitude lower (i.e. more balanced) than the corresponding non-constrained schemes, while sacrificing about half of the compression improvement over Random routing. While they still distribute same-host pages in a manner that is statistically very far from random, they are also very far from the skewed distribution of the non-constrained versions.

Next, Table 2 checks the sensitivity of both measures, Bits per Posting and host-balancing, to the bound flavor and value of α as defined in Section 4.1, with the Constrained Term-based scheme over $m = 400$ partitions. The sensitivity is quite minor[9]. Still, it preserves the trend of the normalized host-distribution value being larger when the Bits per Posting value is lower.

Constraint type	Bits per Posting	Normalized host-distribution
$\max\left\{\left\lceil\frac{n_h}{m}+3\sqrt{\frac{n_h}{m}}\right\rceil,3\right\}$	9.47	1284
$\max\left\{\left\lceil\frac{n_h}{m}+2\sqrt{\frac{n_h}{m}}\right\rceil,3\right\}$	9.48	917
$\max\left\{\left\lceil 1.2\frac{n_h}{m}\right\rceil,3\right\}$	9.48	897
$\max\left\{\left\lceil\frac{n_h}{m}+\sqrt{\frac{n_h}{m}}\right\rceil,3\right\}$	9.49	562
$\max\left\{\left\lceil 1.1\frac{n_h}{m}\right\rceil,3\right\}$	9.495	428
$\max\left\{\left\lceil 1.05\frac{n_h}{m}\right\rceil,3\right\}$	9.5	325

Table 2: Normalized host-distribution and bits per postings (without overhead) values, with the Constrained Term-based scheme (8 million terms) over $m = 400$ partitions, as functions of the constraint parameter α.

7. CONCLUSIONS

This work introduced the problem of reducing a partitioned index size in a low-latency indexing setting via document routing. In such settings, incoming documents must be made searchable quickly, and time-consuming document reordering algorithms cannot be applied. The industry standard routes documents randomly to the partitions, evenly distributing same-host pages across the partitions, which is advantageous for query-time performance. However, this results in indexes which do not compress well. We frame document routing as an online problem, and present document routing schemes that result in indexes that are up to

[9]This was true also for the Constrained Greedy scheme, whose results are not shown

30% more compact. These algorithms can be run in both centralized and distributed fashions.

We prove that one such scheme – Term-based routing – yields better compression than Random routing over a corpus model appearing in the literature. Its advantages are accentuated on real-life corpora, where same-host documents exhibit high similarity between them. Term-based routing is also lightweight in terms of time and space complexity as compared with the other non-random routing schemes.

For each routing scheme, we explored how same-host pages are spread across the partitions, and found a clear trade-off between compression and a balanced host distribution. This holds also when the routing algorithms are constrained to not exceed certain imbalance criteria. Essentially, this shows that much of the redundancy that routing algorithms exploit in Web collections stems from routing each host's pages unevenly, to a small set of partitions.

We believe that low-latency partitioned index systems offer additional trade-offs between issues that are well understood in batch indexing settings, and plan to pursue such trade-offs in future work. In addition, we plan to investigate weighted versions of the Term-based routing scheme. These weights, which will be functions of the terms' frequencies, will refine the current intersection-based score of a document with respect to each partition.

8. REFERENCES

[1] V. Anh and A. Moffat. Inverted index compression using word-aligned binary codes. In *in Proc. Information Retrieval*, pages 8(1):151–166, Jan 2005.

[2] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the web. *ACM Trans. Interet Technol.*, 1(1):2–43, 2001.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[4] C. Badue, R. Baeza-yates, B. Ribeiro-neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *In Proc. of the 9th String Processing and Information Retrieval Symposium (SPIRE'01)*, pages 10–20. IEEE CS Press, 2001.

[5] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, April 2003.

[6] R. Blanco. *Index Compression for Information Retrieval Systems*. PhD thesis, University of A Coruña, 2008.

[7] R. Blanco and R. Barreiro. Tsp and cluster-based solutions to the reassignment of document identifiers. *Journal of Information Retrieval (IR)*, 9(4), Sep. 2006. 499-517.

[8] D. Blandford and G. Blelloch. Index compression through document reordering. *Data Compression Conference*, 0:0342, 2002.

[9] P. Boldi and S. Vigna. Codes for the world wide web. *Internet Mathematics*, 2(4):405–427, 2005.

[10] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[11] F. Chierichetti, R. Kumar, and P. Raghavan. Compressed web indexes. In *in Proc. WWW 2009*, pages 451–460, North-Carolina, USA, Apr. 20-24 2009.

[12] S. Ding, J. Attenberg, and T. Suel. Scalable techniques for document identifier assignment. In *in Proc. WWW 2010*, pages 311–320, North-Carolina, USA, Apr. 26-30 2010.

[13] S. Háman. Super-scalar database compression between ram and cpu-cache. Master's thesis, Centrum voor Wiskunde en Informatica (CWI), 2005.

[14] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. Webbase: A repository of web pages. In *in Proc. of the Ninth International World-Wide Web Conference (WWW'00)*, pages 277–293, May 2000.

[15] A. Mofat and L. Stuiver. Binary interpolative coding for effective index compression. *Inf. Retr.*, 3(1):25–47, 2000.

[16] W. Shieh, T. Chen, J. Shann, and C. Chung. Inverted file compression through document identifier reassignment. *Inf. Processing and Management*, 39(1):117i£¡131, 2003.

[17] F. Silvestri. Sorting out the document identifier assignment problem. In *In Proc. of 29th European Conference on IR Research (ECIR'07)*, pages 101–112, 2007.

[18] F. Silvestri, R. Perego, and S. Orlando. Assigning document identifiers to enhance compressibility of web search engines indexes. In *in Proc. of the 2004 ACM Symposium on Applied Computing (SAC'04)*, pages 600–605. ACM Press, 2004.

[19] G. Snedecor and W. Cochran. *Statistical Methods*. Iowa State University Press, eighth edition edition, 1989.

[20] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann, 1999.

[21] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *in Proc. WWW 2009*, Madrid, Spain, Apr.20-24 2009.

[22] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *in Proc. WWW 2008*, pages 387–396, Beijing, China, 2008.

[23] M. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), Jul.