Indexing Strategies for Graceful Degradation of Search Quality

Shuai Ding
Polytechnic Institute of NYU
sding@cis.poly.edu

Sreenivas Gollapudi
Microsoft Research
Mountain View, CA, USA
sreenig@microsoft.com

Samuel Ieong
Microsoft Research
Mountain View, CA, USA
saieong@microsoft.com

Krishnaram Kenthapadi
Microsoft Research
Mountain View, CA, USA
krishnaram.kenthapadi@microsoft.com

Alexandros Ntoulas
Microsoft Research
Mountain View, CA, USA
antoulas@microsoft.com

ABSTRACT

Large web search engines process billions of queries each day over tens of billions of documents with often very stringent requirements for a user's search experience, in particular, low latency and highly relevant search results. Index generation and serving are key to satisfying both these requirements. For example, the load to search engines can vary drastically when popular events happen around the world. In the case when the load is exceeding what the search engine can serve, queries will get dropped. This results in an ungraceful degradation in search quality. Another example that could increase the query load and affect the user's search experience are ambiguous queries which often result in the execution of multiple query alterations in the back end.

In this paper, we look into the problem of designing robust indexing strategies, i.e. strategies that allow for a graceful degradation of search quality in both the above scenarios. We study the problems of index generation and serving using the notions of document allocation, server selection, and document replication. We explore the space of efficient algorithms for these problems and empirically corroborate with existing theory that it is hard to optimally solve the allocation and selection problems without any replication. We propose a greedy replication algorithm and study its performance under different choices of allocation and selection. Further, we show that under random selection and allocation, our algorithm is optimal.

Categories and Subject Descriptors

H.3.3 [**Information Storage And Retrieval**]: Information Search and Retrieval

General Terms

Search Engine, Index Generation, Index Serve, Performance, Algorithm, Experiment

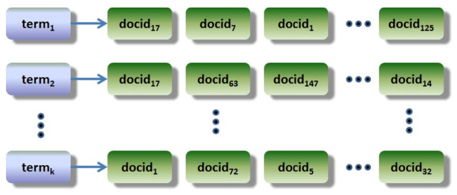

Figure 1: An inverted index over documents and terms

Keywords

Search Quality, Indexing Strategies, Graceful Degradation

1. INTRODUCTION

As the content of the web grows richer, more users rely on search engines to locate relevant information, making search engines one of the most used applications of modern information technology. User experience on a search engine is largely governed by two factors: *latency*, which measures how fast the results are returned to the user, and *accuracy*, which measures how relevant the results are to the query.

The performance of a search index is crucial to a search engine's goal of providing good user experience. Today's commercial search engines employ an inverted index structure that allows efficient retrieval of documents containing a particular term (or word). An inverted index over a collection of terms and the underlying documents consists of a set of inverted lists, one for each term, and often sorted by the relevance of the document for that term [14] (see Figure 1).

The indices of commercial search engines are extremely large and consume a lot of data center resources including CPU, memory, and power. Two important problems involving search indexes that affect search engine performance are:

1. **Index generation**: This problem deals with *allocation* of documents to index servers and building an inverted index on each index server. A naive allocation could be a random assignment of documents to index servers using simple hashing schemes.

2. **Index serving**: At query answering time, each *selected* index server returns a set of top results for the user query, which are then typically aggregated and ranked by a ranking module in the search engine. Under naive allocation, all index servers are typically queried because there is no knowledge of the

index servers to which the top documents for the given query are allocated.

In this paper, we study the inherent trade-off between index selection and allocation strategies on the performance of the index in terms of the quality of the returned search results. More specifically, our goal is to identify strategies that are robust under heavy query loads, i.e. they do not drop queries but instead they allow for a graceful degradation of the quality of results. We begin by observing that under the naive allocation and selection strategies (which we collectively refer to henceforth as indexing strategies), there can be a drop in performance of the system under increased query loads as well as other scenarios such as server failures. Before we proceed to discuss these observations, we need to define what we mean by performance of an index. Generally, a good indexing strategy has to satisfy the following properties even in the presence of unusual spikes in query loads. We will define these notions more formally in Section 2.

(1) **Quality:** The indexing strategy has to surface highly relevant results to user queries.

(2) **High throughput/Low latency:** The indexing strategy should support high throughput of queries.

(3) **Availability:** The documents and the associated inverted index should be available to answer the user queries effectively.

One can clearly observe that the naive indexing strategies do not satisfy all the above properties. Under an unusually high query load, in the absence of any explicit mapping of documents to index servers, all servers need to be queried and therefore there is no way for the search engine but to drop queries that the index cannot handle above its designed peak load. This leads to a drop in the quality of the results since there are queries for which no results are produced. In another scenario when there are node failures, which in principle are common in large distributed systems, property-based allocation (see Section 6) could lead to a drastic drop in quality if the server on which the top documents reside crashes. Often, search engines mitigate this problem by keeping multiple copies of the index (*replication*) at the cost of increased resource usage such as memory and power. In another scenario, all major search engines apply extensive query rewriting techniques[1] to get better search results for the user's queries. Thus, a single user query often results in multiple queries to the index. Again, under the naive selection and allocation strategies, these multiple queries are typically processed *sequentially* on *all* the index server nodes possibly resulting in higher latency for the user query.

One important signal that can be exploited for designing effective indexing strategies is the knowledge of the query workload and the quality of the documents that the search engine typically serves. Both the query frequency and document quality have a direct effect on the indexing strategies. Given the skew typically observed in search query frequencies and the skew in the quality of documents for a given query,[2] one can design more synergistic allocation and selection strategies that can a) gracefully handle high query loads; b) handle node failures; and c) reduce any hot spots (i.e. cases where one or few of the servers see a relatively high fraction of the queries) in the index.

[1]For example, the query 'camera' may be changed to 'digital camera', 'digital photography', 'video camera' and so forth.

[2]Typically observed as a power-law or log-normal distribution.

1.1 Motivation and Objective

Due to the issues with the naive indexing strategies mentioned above, one natural question to ask is: *Can we use less machines in the cluster to answer user queries and also achieve acceptable search quality?* We will show that answers to this question come from more synergistic consideration of allocation and selection strategies. Let us begin by motivating the need for query workload and document-quality aware indexing strategies. Most search engine users only care about the top-k results where k is a number often comparable to the number of index servers available for serving the query results. Therefore, it is likely that only a (small) subset of servers really contribute to serving a query, as explained by the following balls-and-bins argument [13]. Given k balls (top-k results for a query) and n bins (servers in the cluster), the expected number of empty bins after k uniformly random tossings is:

$$n * \left(\frac{n-1}{n}\right)^k \approx \frac{n}{e}\ (when\ k\ equals\ to\ n).$$

In other words, only about 65% of the servers contain a document in the top-k results when k is equal to n. As n grows, the fraction of non-contributing servers also grows. Figure 2 illustrates the growth rate when $k = 10$.

Based on the above observations, we propose the following objective.

> *Given a collection of documents* **D** *and* **n** *servers, find strategies for replication, allocation, and selection such that the quality of results retrieved from* **m** *(* $<$ **n***) servers is maximized.*

1.2 Contribution

In this paper, we address the above index generation problem by considering how to replicate and allocate documents across servers, and the index serving problem by considering how to select servers for query execution. We explore different allocation and selection strategies and their combinations for improved index performance. Overall, in this paper we make the following contributions:

1. We study different allocation and selection strategies for robust indexing. We propose a new selection strategy which outperforms previous selection strategies in search quality.

2. We evaluate our indexing strategies based on speed, quality and availability (or load balancing). Previous work has mostly focused on studying the speed and quality of index. In our work we also study the availability of the results.

3. We study how to employ replication strategies in combination with allocation and selection strategies in order to create indexes with high speed, quality and availability. We study different replication strategies and we propose a greedy replication strategy that is optimal when combined with specific allocation and selection strategies.

4. We experimentally evaluate our indexing strategies on a web dataset of 25.2 million documents. Our results demonstrate that improvement can achieve compared to existing strategies.

The rest of our paper is organized as follows: in Section 2 we discuss the preliminaries and the performance metric for our problem setting. Section 3 and Section 4 study different allocation and selection techniques and their combinations, while Section 5 discusses replication strategies. In Section 6 we present our experimental results and Section 7 discusses the related work. Section 8 concludes.

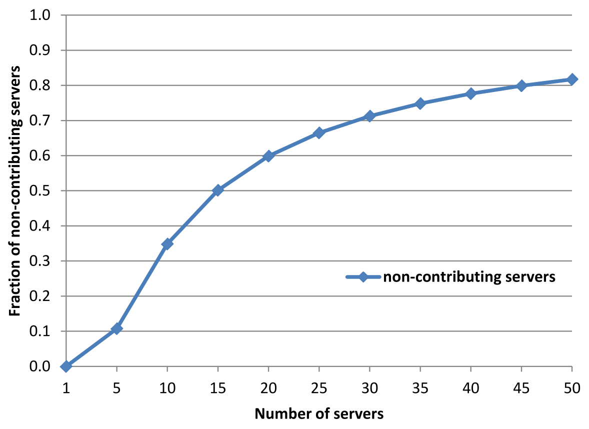

Figure 2: Expected percentage of non-contributing servers. x-axis is the total number of servers, y-axis is the expected percentage of servers which do not contain documents in the top-10 results

2. MODEL

We denote the set of queries by Q, and for each query q, its frequency, or the expected number of times q is queried against the search engine, by $f(q)$. We denote the set of documents to be indexed by D. For each document $d \in D$, there exists a *score* $s(q, d)$ that reflects the relevance of document d is to query q. We treat the scores as input to our problem formulation, and explain the one we used in experiments in the relevant section. For a given query q and a collection of documents $C \subseteq D$, we denote the k documents with the highest score by $B(q, C, k)$. We denote the set of machines, or servers, by $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$. Documents are stored on these machines.

We study the questions of replication, allocation, and selection in this work. Formally, replication is the question of how many copies of a particular document $d \in D$ we make available across all the servers. We denote this by its replication factor $R(d)$. Allocation is the question of which $R(d)$ servers get assigned a copy of document d. The set of servers that got allocated document d is denoted by $A(d)$, and we view a server M_i as a collection of documents. In particular, this allows us to meaningfully take the union over machines, as this is simply the union of the underlying documents stored in the machines. Both replication and allocation are performed *offline*. Selection is the question of which servers are used to answer a query at *runtime*. Given a query q, and a desired number of servers m, we select $S(q, m)$ to answer the query. Under normal circumstances, we expect $m = n$; however, as motivated previously, due to machine failures or abnormally high query load, we may want to select only a subset of servers to answer a query. Indeed, our focus is on situations where $m < n$, possibly substantially so. A system is determined by the choice of replication, allocation, and selection strategies, (R, A, S).

From the users' perspective, given a query q, the set of top-k documents retrieved when m machines are used is

$$docs(R, A, S, q, m, k) = B(q, \{\cup S(q, m)\}, k) \ .$$

This is determined not only by selection, but also by replication and allocation as they in turn determine what documents are stored on which servers. When it is clear from context, we simply write $docs(q, m, k)$.

Next, we formalize the three desired properties of a system that we outlined in Section 1. We begin with *quality*. For a given query q, we define the quality of the retrieved results by the sum total of the scores of the top-k documents retrieved.

$$qual(q, m, k) = \sum_{d \in docs(q,m,k)} s(q, d) \ .$$

The expected quality of a system, when m servers are used to retrieve top-k results, is given by

$$qual(R, A, S, m, k) = \sum_{q \in Q} f(q) qual(q, m, k) \ .$$

High throughput or *low latency* implies that no index server becomes a hotspot, i.e., a bottleneck, in the presence of query load spikes or server failures. In other words, no index server should be selected to answer queries all the time. We can make this formal by measuring the load of a machine, in terms of the number of queries it has to answer. We define load of the machine as follows.

$$load(M_i, m) = \sum_{q \in Q, M_i \in S(q,m)} f(q) \ .$$

In other words, load of server M_i is determined by the expected number of queries it has to answer. The load of a system is determined by the most loaded machine, i.e.,

$$load(S, m) = \max_i load(M_i, m) \ .$$

Under this definition, a system with smaller load will have lower latency, and all else being equal, we prefer a system with smaller load. Note that load is determined only by selection, and how many machines are used to answer queries.

A useful way of thinking about load is as follows. Consider the baseline of using all servers to answer all queries. The load on each server will then be $\sum_{q \in Q} f(q)$. If we manage to reduce the load on the most loaded machine by half, then we can handle twice the number of queries without an increase in latency. Of course, if this is achieved by selecting only a subset of servers to answer queries, we may suffer from an associated drop in quality. Indeed, this is the subject of investigation in this paper.

Finally, *availability* implies that *most* of the documents should still be accessible during server failures for *any* query. It suggests that machine failures should not lead to drastic decrease in the quality of results. We approximate the potential damage when server M_i fails by the expected contribution of M_i to answering queries. Specifically, we define the value of server M_i in terms of the expected number of top-k documents it contributes, weighted by the number of times the document is replicated, namely,

$$value(M_i, R, m, k) = \sum_{q \in Q} f(q) \sum_{d \in (docs(q,m,k) \cap M_i)} \frac{1}{R(d)} \ .$$

One way to interpret the definition of value of a server is as follows. The baseline is determined by the documents the system retrieves in the absence of machine failures, $docs(q, m, k)$. If a server fails, then it leads to losing certain documents, captured by the set $(docs(q, m, k) \cap M_i)$. For these documents, the actual loss is determined by how many copies of this document is available in the system. The loss is 100% if there is only one copy ($R(d) = 1$), but less if it is replicated more. Hence, we weigh the loss by $\frac{1}{R(d)}$.

The availability of a system is determined by the loss in value in the worst-case failure

$$loss(R, A, S, m, k) = \max_i value(M_i, m, k) \ .$$

Under this definition, a system with smaller loss will have higher availability, and all else being equal, we prefer a system with smaller loss. For simplicity and to ensure that *loss* depends only on replication and allocation, we choose $m = n$ in our computations. Alternative formalization that captures random or multiple failures is possible; the current one is chosen for simplicity and ease of interpretation.

In summary, we are interested in choosing replication, allocation, and selection strategies so as to *maximize quality*, *minimize load*, and *minimize loss*. There are trade-offs among these conflicting objectives, which constitute the focus of this study.

3. DOCUMENT ALLOCATION

Allocation studies the problem of how to assign documents to servers. It plays an important role in determining the availability and the quality of the system. We propose a balanced allocation strategy that exploits the query work load and the document quality.

Instead of assigning documents uniformly at random and achieving good availability in expectation, one can attempt to explicitly minimize the worst-case loss of the system. Recall that to maximize availability, our objective is to:

$$\min loss(R, A, S, m, k) \ .$$

However, optimizing this objective requires knowing the choice of selection strategy. To work around this difficulty, instead of allocating the documents in the top-k results as balanced as possible among the servers, we approximate the value of a document by its expected score to a random query,

$$v(d) = \sum_{q \in Q} f(q) s(q, d)$$

We then want to evenly spread the scores for documents among servers. This is known as the *"makespan problem"* in job scheduling [2, 3]. To this end, we utilize a solution similar to the Longest Processing Time (LPT) algorithm in [8]. More specifically, we allocate the documents greedily to the server with the lowest sum total value of documents. This algorithm is essentially a 4/3 approximation of the optimal [8]. The complete algorithm is given as follows:

```
**foreach** $M_i \in \mathcal{M}$ **do** $v(M_i) \leftarrow 0$;
Sort $D$ in descending order of $value(d)$;
**foreach** $d \in D$ **do**
    Find $M_i$ with smallest $v(M_i)$;
    Assign $d$ to $M_i$;
    $v(M_i) \leftarrow v(M_i) + v(d)$;
**end**
```

Algorithm 1: BALANCEDALLOCATION Algorithm

Later, in our experimental section we compare the BALANCEDALLOCATION algorithm with two well-known allocation techniques, more specifically random allocation and property-based allocation [10, 11, 16, 19].

4. SERVER SELECTION

Selection studies the problem of which servers are chosen to answer a query. It plays a central role in determining the speed of the system. Together with allocation, they jointly determine the quality of the system. It is typically implemented with two processes: an offline process that precomputes certain statistics, and a runtime process that uses the statistics to select the servers to answer a query at runtime.

If one cares only about quality, the best strategy is to always select all servers. However, this may be infeasible in case of abnormally high workloads as the alternative will be to ignore the queries of some users entirely. In this section, we study selection strategies where the fraction of servers that can be used to answer the query is specified as an input. We propose a selection strategy based on the distribution of document quality for any given query. We approximate the distribution by computing the histogram of term-level scores. In our experimental section, we will compare our proposed selection strategy with those in the literature, i.e. ReDDe [11] and Gloss [9].

In our histogram-based method, we select servers with the highest estimated number of relevant documents to the query. This number is estimated using histogram of term-level scores of each server, similar to [18]. Here, we implicitly assume that the relevance score of a document equals the sum of the scores of the document to terms in the query. For each term t, let $H(t, M_i)$ denote the *histogram* of scores of the documents stored on server M_i for term t. We assume that the histogram is stored at the finest level—a list of scores in decreasing order; in practice, the list is approximated by pairs of (range, count), but the approach can be suitably adapted. We assume that the histogram only stores the non-zero scores, and denote its length by L_t. In the following, we treat the histogram as an array.

Consider some two-term query $q = \{t_1, t_2\}$. Assuming term independence, one can estimate the number of documents on server M_i for which the sum of scores exceed threshold τ using Algorithm 2. The m servers with the highest estimates are selected to answer the query.

```
**input** : Histograms $H(t_1, M_i)$, $H(t_2, M_i)$, threshold $\tau$
**output**: Estimated no. of relevant docs $\geq \tau$ on $M_i$
$E(\{t_1, t_2\}, \tau, M_i) \leftarrow 0$;
**for** $i_1 \leftarrow 1$ **to** $L_{t_1}$ **do**
    $s_1 \leftarrow H(t_1, M_i)[i]$;
    Search for $s_2$ in $H(t_2, M_i)$ where $s_1 + s_2 \geq \tau$;
    Let $i_2$ be the position of $s_2$ in $H(t_2, M_i)$;
    $E(\{t_1, t_2\}, \tau, M_i) \leftarrow E(\{t_1, t_2\}, \tau, M_i) + \frac{i_1 i_2}{|M_i|}$;
**end**
**return** $E(\{t_1, t_2\}, \tau, M_i)$
```

Algorithm 2: HISTOGRAMSELECTION Algorithm

Note that we can perform certain optimizations to speed up the algorithm, such as reducing the search space using the fact that the position in the second histogram is monotonically non-increasing, and to perform early termination when the scores of the remaining documents are too low. The algorithm can easily be generalized to queries with more than two terms.

The algorithm requires an input threshold that can be estimated with the help of the entire histogram for the terms, $H(t_1, D)$ and $H(t_2, D)$. We start with a very high threshold $\overline{\tau}$, and use Algorithm 2 to estimate the number of documents exceeding this threshold for the entire document collection. We repeatedly lower the threshold until the number of documents is at least k, the desired number of documents to return. The threshold is then used in the selection process.

5. ALLOCATING MULTIPLE COPIES

Previous work [10, 9, 16] mainly focuses on indexing strategies which try to use less machines and return good search results, without studying the availability of the index. As discussed before, the availability is a very important factor in designing search engine indexing, especially in a large cluster where disk failure is very common. Given the importance of the index availability, in this section we study different replication strategies. Replication studies the problem of how many copies of each document we make available across all the servers. It plays a key role in determining the availability and the quality of the system. Replication, together with allocation and selection, determines the quality of the system.

We assume that the amount of additional space available for replication is specified by a parameter $C > 0$ (typically C could be a fraction, say, 0.1 or 0.2). The total space available over all servers can accommodate a total of $(1 + C)|D|$ documents. Each server is assumed to have the same capacity, that is, it can accommodate $(1+C)|D|/n$ documents. The goal of replication strategy is to determine the replication function $R()$, so that for each document $d \in D$, $R(d)$ copies are made available across all servers. We require every document to be replicated at least once, that is, $R(d) \geq 1 \ \forall d \in D$. Hence we study how to assign the additional space ($C|D|$) to documents in D. To enable better understanding of replication and for ease of analysis and experimentation, we separate the replication strategy from the allocation strategy, that is, we determine the replication factor of a document independent of which servers would receive the copies of the document. We study four replication strategies below.

We first state and discuss a result that motivates our study of replication.

THEOREM 1. *For a given document replication vector, under random selection, the expected quality of the results returned does not have any dependence on the allocation strategy.*

In other words, the theorem implies that if we use the simple and easy-to-implement strategy of random selection, the allocation strategy does not affect the quality of the results. However, the replication strategy is important since it determines if a top document for a query will be part of the results retrieved and thus indeed affects the quality of the results. As a corollary, when there is no replication ($R(d) = 1 \ \forall d \in D$), if random server selection is used, we can allocate documents with the goal of maximizing availability, that is, minimizing $loss$.

The proof of the theorem follows from the following intuition. Since m different servers are selected at random, for a query and any document d, the probability that it will be included in $docs(q, m, k)$ does not depend on the specific choice of servers where the $R(d)$ copies of d are allocated. In other words, due to the randomness in selection, the probability that there is overlap between the set of m servers randomly selected and the set of $R(d)$ servers where d is allocated depends only on $n, m, R(d)$ and *not* on the specific choices (ids) of the $R(d)$ servers. Thus, under random selection, the allocation strategy does not influence the quality of results.

5.1 Uniform Replication

A naive way to replicate documents is to treat all the documents equally and distribute the additional space equally amongst all documents. First each document is replicated $(1 + \lfloor C \rfloor)$ times, and then with probability $(C - \lfloor C \rfloor)$, each document is allowed an additional copy. When $0 < C < 1$, this process is equivalent to including a second copy of a document with probability C. We call this replication strategy *uniform replication*. As uniform replication strategy is easy and natural to implement, we use it as our baseline in experiments.

5.2 Quality-Based Replication

Instead of treating all the documents equally, we next consider a strategy that gives preference to documents with high *value*. The intuition is that high value documents are more likely to be among the top k results of queries and hence it is desirable to replicate them more often. In quality-based replication, we first assign one copy of every document, and allocate the remaining $C|D|$ space greedily based on the document value, that is, consider the documents in the descending order of value and replicate each such document on all servers until there is no more space available. Recall that the value of a document d is defined as:

$$v(d) = \sum_{q \in Q} f(q)s(q,d) \ .$$

Algorithm QUALITYBIASEDREPLICATION formally describes this replication strategy.

```
foreach $d \in D$ do
    $v(d) \leftarrow \sum_{q \in Q} f(q)s(q,d)$;
end
$w \leftarrow \lfloor \frac{(C-1)|D|}{n-1} \rfloor$;
Let $d_1, d_2, \ldots, d_{|D|}$ represent the documents sorted in the
descending order of $v(d)$;
for $i = 1 \ldots w$ do
    $R(d_i) \leftarrow n$;
end
$R(d_{w+1}) \leftarrow (C-1)|D| + 1 - (n-1)w$;
for $i = w+2 \ldots |D|$ do
    $R(d_i) \leftarrow 1$;
end
```

Algorithm 3: QUALITYBIASEDREPLICATION algorithm

5.3 Workload-Aware Replication

In *workload-aware replication*, we assume that the workload is known, and hence the number of available servers (m) to answer a query is also known. To improve upon quality-based replication, instead of replicating a high value document on all servers, we only need to replicate on $n + 1 - m$ servers. Since we select m out of n servers for each query, we are *guaranteed* to select at least one server containing the high value document.

Algorithm WORKLOADAWAREREPLICATION is similar to QUALITYBIASEDREPLICATION, the only difference being that the maximum replication factor is smaller ($n+1-m$ instead of n), hence the available space permits maximal replication of more (high-valued) documents.

```
foreach $d \in D$ do
    $v(d) \leftarrow \sum_{q \in Q} f(q)s(q,d)$;
end
$w \leftarrow \lfloor \frac{(C-1)|D|}{n-m} \rfloor$;
Let $d_1, d_2, \ldots, d_{|D|}$ represent the documents sorted in the
descending order of $v(d)$;
for $i = 1 \ldots w$ do
    $R(d_i) \leftarrow n + 1 - m$;
end
$R(d_{w+1}) \leftarrow (C-1)|D| + 1 - (n-m)w$;
for $i = w+2 \ldots |D|$ do
    $R(d_i) \leftarrow 1$;
end
```

Algorithm 4: WORKLOADAWAREREPLICATION algorithm

5.4 Greedy Replication

Our final replication strategy is motivated by the following observation. As more copies of a document become available, intuitively the incremental benefit of an additional copy follows diminishing return. In fact, we implicitly used this fact in workload-aware replication, by observing that beyond $n + 1 - m$ copies, an additional copy of a document carries no benefit.

Formally we analyze the probability that a document replicated $R(d)$ times will be selected (or "hit") when m servers are probed randomly. We observe that this probability depends only on the number of servers n, as well as m and $R(d)$, and hence denote it as $Pr(hit(d)|n, m, R(d))$. Then,

$$Pr(hit(d)|n, m, R(d)) = 1 - \prod_{i=0}^{m-1} \left(1 - \frac{R(d)}{n-i}\right) \qquad (1)$$

For given choices of n and m, consider the successive differences in the above probability as $R(d)$ increases, and denote it as: $\delta(R(d)) = Pr(hit(d)|n, m, R(d)) - Pr(hit(d)|n, m, R(d)-1)$. We remark that $\delta(R(d))$ decreases as $R(d)$ increases. Table 1 shows how the values behave for $n = 10$ and $m = 2$ as $R(d)$ is increased from 1 to 10. For example, it is more beneficial to provide an extra copy when there are just 2 copies as compared to when there are already 7 copies.

Hence given two documents d_1 and d_2 with $v(d_1)$ slightly larger than $v(d_2)$, we prefer their replication counts to be close. Under workload-aware replication, it is possible that d_1 is replicated $n + 1 - m$ times and d_2 just once.

Combining the above desired property with the desire to give preference to high value documents, we devise the following optimization objective for replication.

$$\text{Maximize} \sum_{d \in D} v(d) \cdot Pr(hit(d)|n, m, R(d))$$

$$\text{s.t.} \sum_{d \in D} R(d) \leq (1 + C)|D| \text{ and } R(d) \geq 1 \;\; \forall d \in D.$$

This optimization problem consists of $|D|$ variables ($R(d)$'s), with an objective function that is extremely non-linear (in fact, has degree m dependence on each variable). Because of this complexity, at first, this problem seems impossible to solve optimally or to even approximate.

We next propose a greedy algorithm and surprisingly show that it achieves the optimal solution. The key insight is to notice that even though $Pr(hit(d)|n, m, R(d))$ is non-linear in $R(d)$, there are only n meaningful choices for $R(d)$ for any document d and thus only n possible values for $Pr(hit(d)|n, m, R(d))$ or $\delta(R(d))$. For each of $C|D|$ steps, the algorithm picks a document with maximum incremental benefit to the objective, that is, with maximum $v(d) \cdot \delta(R(d))$ and assigns an additional copy to the document.

This algorithm can be efficiently implemented by maintaining a binary heap of the incremental gain ($v(d) \cdot \delta(R(d) + 1)$) values for every document. At each step, exactly one document is assigned an additional copy, and thus only its gain value needs to be updated. As this update requires $O(\log |D|)$ steps in the worst case, the algorithm has time complexity $O(C|D| \log |D|)$.

We next show that the greedy algorithm is optimal, i.e., the locally optimal choices gives a globally optimal solution.

THEOREM 2. *Algorithm* GREEDYREPLICATION *finds the optimal replication vector with respect to the replication optimization objective.*

```
Compute the vector $(\delta(1), \delta(2), \ldots, \delta(n))$ using Eqn 1;
foreach $d \in D$ do
    $v(d) \leftarrow \sum_{q \in Q} f(q)s(q, d)$;
    $R(d_i) \leftarrow 1$;
end
for $i = 1 \ldots C|D|$ do
    Choose the document that can contribute the maximal
    incremental gain to the objective, that is,
    $d^* \leftarrow \arg\max_{d \in D}(v(d) \cdot \delta(R(d) + 1))$;
    $R(d^*) \leftarrow R(d^*) + 1$;
end
```

Algorithm 5: GREEDYREPLICATION algorithm

The basic intuition is that the incremental gain in quality by replicating a document decreases as a document is replicated more times. Further, the gain from replicating a document is independent on how many times the other documents are replicated. Therefore, when presented with a choice of documents to replicate, one can proceed greedily. A formal argument is provided below.

PROOF. The proof follows by contradiction. Suppose GREEDYREPLICATION is not optimal, so that the objective evaluated on the solution ALG computed by GREEDYREPLICATION is less than the optimal value. Then for any optimal solution OPT, we can always find two documents $d_1, d_2 \in D$ on which ALG and OPT differ in opposite directions, that is, $R_{OPT}(d_1) \geq R_{ALG}(d_1) + 1$ and $R_{ALG}(d_2) \geq R_{OPT}(d_2) + 1$. If there is more than one optimal solution, we fix OPT as the one with smallest number of copies for d_1.

Define a new valid assignment DIF that differs from OPT on just d_1 and d_2: $R_{DIF}(d_1) = R_{OPT}(d_1) - 1$ (has one less copy of d_1) and $R_{DIF}(d_2) = R_{OPT}(d_2) + 1$ (has one more copy of d_2). It follows that DIF is not optimal, and hence we have:

$$v(d_1) \cdot \delta(R_{OPT}(d_1)) > v(d_2) \cdot \delta(R_{OPT}(d_2) + 1) \qquad (2)$$

Since $\delta()$ is a decreasing function, using $R_{OPT}(d_1) \geq R_{ALG}(d_1) + 1$ and $R_{ALG}(d_2) \geq R_{OPT}(d_2) + 1$, we have:

$$v(d_1) \cdot \delta(R_{ALG}(d_1) + 1) \geq v(d_1) \cdot \delta(R_{OPT}(d_1)) \qquad (3)$$

$$v(d_2) \cdot \delta(R_{OPT}(d_2) + 1) \geq v(d_2) \cdot \delta(R_{ALG}(d_2)) \qquad (4)$$

Combining equations 2, 3 and 4, we get:

$$v(d_1) \cdot \delta(R_{ALG}(d_1) + 1) > v(d_2) \cdot \delta(R_{ALG}(d_2)), \qquad (5)$$

implying that when algorithm GREEDYREPLICATION chose to include the last copy of d_2, d_1 would have been a better choice. This is a contradiction since our algorithm is greedy. $\square$

6. EXPERIMENTAL EVALUATION

6.1 Dataset and Performance Metric

In this section we experimentally evaluate the performance of our algorithms on a real-world dataset based on the notions of quality, availability and speed that we discussed earlier. To serve as our collection of documents we used the *Gov2* dataset from TREC's Terabyte Track [1] which consists of 25.2 million Web pages from the `.gov` domain. To serve as our query load for our various policies we used a random subset of the query log of a major search engine comprising 100 thousand queries. In order to have a setting

$R(d)$	1	2	3	4	5	6	7	8	9	10
$Pr(hit(d)\|n, m, R(d))$	0.20	0.37	0.53	0.66	0.77	0.86	0.93	0.97	1.00	1.00
$\delta(R(d))$	0.20	0.17	0.16	0.13	0.11	0.09	0.07	0.04	0.03	0.00

Table 1: Diminishing return observed with more copies of a document ($n = 10$, $m = 2$)

close to that of a realistic search engine, we performed our experiments on a cluster of 50 machines.

The exact ranking function that each search engine employs is closely guarded secret. In this paper we define the score of a document d to query q similar to section 2 as:

$$score(q, d) = \sum_{term\ t\ in\ q} score(t, d)$$

To compute the term relevance we used the BM25 scores [17].

Finally, to measure the quality performance of our policies we used the mean precision at k. That is, for a given allocation, selection, replication policy configuration, we compute the average fraction of top-k documents returned when the policy is applied compared to the top-k when no policy is applied. More formally, for a query load of m queries and for a configuration p:

$$Quality(p) = \frac{1}{m} \sum_{i=1}^{m} \frac{|Res_p(q_i) \cup Res_{OPT}(q_i)|}{|Res_{OPT}(q_i)|}$$

where $Res_p(q_i)$ is the top-k result set of configuration p for query q_i, and $Res_{OPT}(q_i)$ is the top-k result set for query q_i when no policy is applied (i.e. the query is sent to every machine). In our experiments we used $k = 10$.

6.2 Comparison of Allocation Strategies

We compare BALANCEDALLOCATION strategy with two well-known document allocation paradigms.

6.2.1 Random Allocation

A simple allocation strategy is to assign documents uniformly at random to the servers. Random allocation typically achieves good availability as each server receives in expectation the same number of relevant documents for each query. Formally, under random allocation, and assuming that each server has the same probability of serving a query, the expected value of each of the server is the same, i.e., for any two servers M_i and M_j,

$$\mathbb{E}[value(M_i, R, m, k)] = \mathbb{E}[value(M_j, R, m, k)] \ .$$

Since the total value of the system is fixed, this means that the loss is minimized in expectation.

6.2.2 Property-Based Allocation

A popular class of allocation strategies operate by first grouping the documents by certain properties, and then assigning documents with the same (or similar) property value to the same server.

Several property-based allocation strategies have been proposed in the past literature. This includes (1) source-based allocation that is based on treating the domain of a document as its property, used as a baseline for evaluation in [19]; (2) query-driven allocation that is based on grouping together documents that are used to answer similar queries [16]; and (3) topic-based allocation that is based on clustering documents around topics using KL-divergence as the distance function [11, 10].

Property-based allocation strategy typically does not guarantee good availability, as it may happen that many valuable documents get assigned to the same server. To understand better the performance of such allocation strategies, we implemented and compared the source-based allocation method with other allocation strategies in our experiment. This allocation sorts all the pages based on the URLs, then groups consecutive documents and assigns each group to the same server.

6.3 Comparison of Selection Strategies

We compare the HISTOGRAMSELECTION strategy with three strategies, viz., random selection, selection based on a random subset of documents, and a centralized strategy.

6.3.1 Random selection

Random selection is a pure runtime strategy that does not require the storage of any precomputed statistics. Given a query, we randomly select a subset of machines to answer the query.

6.3.2 ReDDe

ReDDe is proposed in [11]. During the offline process, a random sample of documents of about 0.1% of the total size of the collection are drawn, and an index over these samples are computed and stored in a central server. At runtime, given a query, this central index is first queried, and the subset of machines is chosen based on the result to the first query.

6.3.3 Gloss

Gloss is proposed in [9]. During the offline process, for each server, for each term, the average score of the documents for the term for the server is computed and stored in a central server. At runtime a score for each server is computed assuming independence, and the servers with the highest scores are selected.

6.4 Evaluation of Result Quality

We start our experimental evaluation by comparing the quality of results of different combinations of policies. To this end, we first loaded the documents to the cluster of 50 machines using different allocation policies. Then, we ran the queries against the cluster using different combinations of allocation and selection policies. In order to have a consistent view among the runs of the configurations and minimize variations, we evaluated the effect of the work load increase by using the same set of queries over all configurations while restricting our policies to direct each query to a progressively decreasing number of machines (denoting a corresponding increase in the work load). For each combination and selection policies we report the *Quality* metric as defined in the previous section.

We show the results in Figures 3, 4 and 5, each one corresponding to an allocation policy with different lines corresponding to different selection policies. The horizontal axis represents the workload expressed as a multiple of the initial workload when we can afford to use all 50 machines. For example, a workload of 2.0 implies that we have twice the amount of workload and hence we will use half of our machines to answer each query. The vertical axis corresponds to the *Quality* of a given allocation, selection configuration. To better evaluate the performance of our policies we also report the quality of an optimal selection policy (denoted OPT) which selects the best possible combination of machines for a given allocation

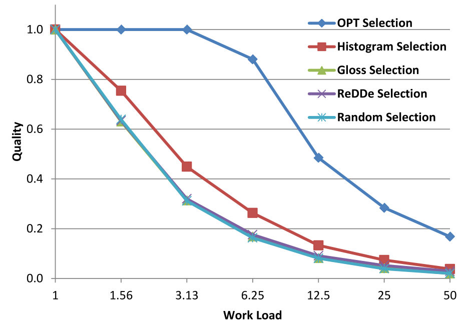

Figure 3: Quality of random allocation with different selection strategies under increasing work load.

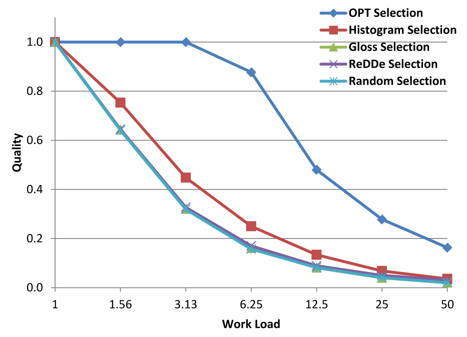

Figure 4: Quality of LPT allocation with different selection strategies under increasing work load.

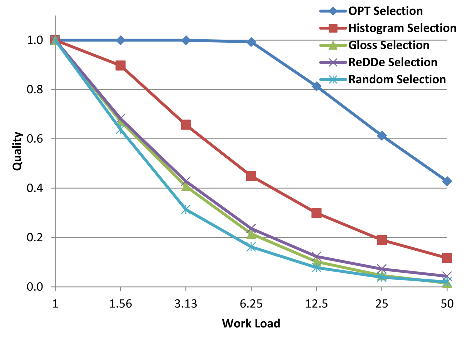

Figure 5: Quality of source-based allocation with different selection strategies under increasing work load.

policy. Also, following [11], for the ReDDe selection policy we used 0.1% of the index to store the necessary statistics.

The general observation regarding the performance of different configurations is that they perform worse than optimal. Additionally, as expected, the achieved quality of the policies decreases as the work load increases. Regarding the random allocation shown in Figure 3, we observe that the Gloss, ReDDe and Random selection policies achieve similar performance. Since documents are evenly spread out among machines, Gloss and ReDDe cannot make good predictions. Our Histogram selection is slightly better overall, but it comes with the additional cost in space and computation time as we discussed in Section 2. Additionally, its performance difference is diminishing as the work load increases.

The LPT allocation policy's goal is to allocate the documents in a very balanced way across all servers. Due to this fact, as we can see from Figure 4, all the selection policies perform similarly to each other. Additionally, their overall performance is slightly worse than the random allocation, which is again an artifact of the more balanced allocation that LPT achieves.

Finally, regarding the Source-based allocation policy shown in Figure 5, we observe that the techniques taking advantage of statistics on the documents on each machine in general outperform the random selection. In this case, ReDDe and Gloss perform similarly with ReDDe slightly better than Gloss. This is due to the fact that Source-based allocation in its attempt to maintain the average score of documents the same across machines may create an imbalance in the number of documents allocated among machines for a given term, which benefits ReDDe.

We also conducted two statistical tests to determine the significance of the differences in quality of the results under different workloads, selection strategy, and allocation strategy. Under a repeated-measure analysis of variance (ANOVA), where each query is treated as a subject that is tested under different combinations of workloads, selection, and allocation, all of these three factors are statistically significant in determining the quality of the retrieved results (at p-value of < 0.0001). Histogram selection and Source-based allocation has especially large positive effect in the quality of the retrieved results across all workloads. Under a paired t-test between all pairs of different conditions, almost all of the differences in the quality of the results are statistically different at p-value of < 0.0001. There are two cases where the results are statistically insignificant: (1) when random selection strategy is paired up with any allocation strategy, which is expected as random selection does not select servers based on statistics of the documents stored; and (2) between random selection and Gloss when allocation is either random or LPT, which is due to both allocation strategy gives rise to servers with close to identical average scores, in which server selection under Gloss is more or less random.

In summary, based on our observations regarding the quality of the different allocation and selection policies we conclude that the Random allocation and Random selection combination is a good candidate due to its simplicity of implementation. Histogram-based selection does perform better than Random selection, however if this particular selection policy is employed it is best combined with Source-based allocation. In the next section we study the availability of these two policies.

6.5 Evaluation of Availability

Another important characteristic of an effective indexing strategy is high availability. Given the scale of search engine indexes, it is important for the system to continue providing good results even under the presence of node failures. In this experiment we study the availability of the indexes created under the various allocation poli-

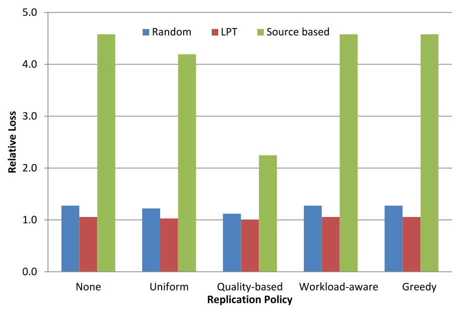

Figure 6: Loss function indicating availability for different allocation and replication policies.

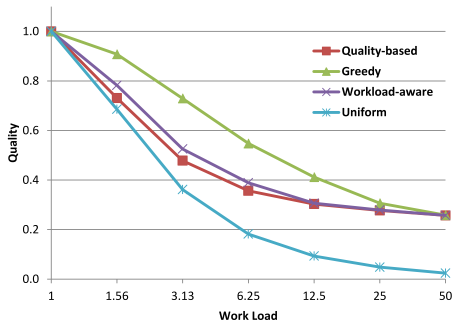

Figure 7: Quality for different replication policies under Random allocation and Random selection.

cies that we used in our previous experiment. To this end, for the three allocation policies, namely Random, LPT and Source-based we computed the loss value for our query set as defined in Section 2. We show the result in the first group of bars of Figure 6. The vertical axis shows the relative loss compared to the average loss across all servers. The higher the value (i.e. the loss), the more concentrated the good documents are in one particular server and thus the lower the availability and the more the system will suffer in the worst case node failure.

From the graph, we observe that the allocation policy with the minimum loss is the LPT allocation policy. This is expected since this particular policy makes an explicit attempt to balance the documents across the machines. The Random allocation policy also achieves reasonably good loss value as it is slightly higher than the LPT. Finally, although the source-based allocation policy was shown to perform very well in terms of quality in our previous experiment, its performance is the worst among the three policies in terms of availability. Its relative loss is almost 4 times higher than that of LPT and Random allocation policies.

6.5.1 *Benefit of Replication*

One way that is typically used to increase availability is the replication of the documents within the index. As additional copies of the same document exist in the system, the availability is expected to be higher since the replicated documents can still be found in case of a node failure. To study this effect, we repeated our availability experiment with the different replication policies that we discussed in Section 5. All replication policies were allowed an additional of 20% (i.e. $C = 0.2$) of the index to be used for replicating documents. We show the relative loss for the Uniform, Quality-based, Workload-aware and Greedy replication policies in the rightmost 4 groups of bars of Figure 6 respectively.

Our first observation comes from the fact that the availability remains almost the same for the Random and LPT allocation policies across the board. The availability of the Source-based allocation is increased when it is combined with the Quality-based replication policy (which also improves slightly the Random allocation). Unfortunately however, the benefits of Source-based allocation in terms of quality are not paired with high availability since it has a loss value twice as high as Random and LPT, even when combined with the best replication policy. On the contrary, the Random allocation with Random selection is more balanced in terms of both quality and availability.

Since we have allowed additional space for replicated documents in our index, we have also affected quality as more versions of the same document are available in the system. To this end, we studied the quality of the Random-allocation and Random-selection combination[3] under different replication policies. We show the results in Figure 7 with the horizontal axis representing workload and the vertical axis the quality as before. We observe that, overall, the quality of the results has increased which is due to the fact that the selection policy can now find more good quality documents. Overall, the Greedy replication policy performs the best.

6.6 Evaluation of Speed

We now turn to study the performance of the different policies in terms of query throughput. One of our main goals in designing an effective indexing mechanism is to be able to accommodate increases in the query load (possibly with some loss in quality) but without suffering delays to the queries served by the system.

To this end, we assume a setting where our system is operating at a query load close to peak capacity and where a selection policy will take charge to select a subset of the machines to serve an incoming query. In our experiment, we set this point to be 1000 queries per second for each of the 50 machines. Then, we increased the incoming query load to the system and we report the number of queries per second directed to the machine that received the maximum number of queries. The higher this number, the higher the expected number of dropped queries since the machine receiving the queries may not be able to serve them.

We report the result on Figure 8. We report our results for the combinations of Random allocation with Random selection, LPT allocation with Histogram selection and Source-based allocation with Histogram selection. Note that the presence of a replication policy does not affect the outcome of our experiment since the same set of machines will be selected. The horizontal axis represents the query load and the vertical axis shows the number of queries per second reaching the most loaded machine under a given query load.

Our first observation is that, overall, as the work load increases the maximum work load at one server remains more or less stable for Random and LPT. For these two policies, about half of the machines got a query load that was higher than the current operating work load of 1000 queries per second. In the case of Source-based, the maximum query load that the busiest machine receives increases proportionally to the increase in the query load in the system. For example, when the work load increases by 6 the maximum query load in the busiest machine increases almost 5 times. Overall, the combination of Random allocation with Random selection performs the best with the maximum load being only 1.4% higher than the current operating work load.

This last observation makes the combination of Random allocation, Random selection and Greedy Replication policies a very good all-around candidate for organizing an index. This combina-

[3]The observations for the remaining policy combinations are similar; we omit the graphs due to space constraints.

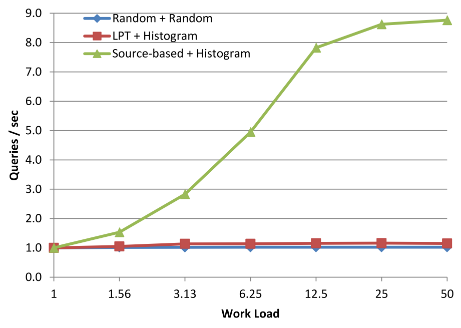

Figure 8: Throughput in thousand queries per second for increasing work loads.

tion is simple to implement and maintain, and it achieves a very good balance between quality, availability and speed.

7. RELATED WORK

In web search literature, there is a plethora of work on document allocation policies across *shards* of index servers. This work can be broadly classified into two categories. One that exploits relationship between documents and the other that is based on relationship between queries. The relationship between documents can be based on either semantic (e.g., *topic-based*) [19] or on other features (e.g., *source-based*) [19]. In their work on document allocation based on topic-based clustering, Xu and Croft [19, 11] used a two-pass K-means clustering algorithm and a KL-divergence distance metric to organize a collection into 100 topical clusters. Several property-based allocation strategies have been proposed in the past literature. This includes a source-based allocation that is based on treating the domain of a document as its property, used as a baseline for evaluation in [19] and a query-driven allocation that is based on grouping together documents that are used to answer similar queries [16].

In the literature related to selection strategies, people have broadly considered exhaustive search and selective search strategies. In the space of selective search strategies, Kulkarni and Callan [11] proposed a two-query approach in which the first query is used to estimate the number of relevant results on each index server. The second query is executed against the servers chosen based on the results of the first query. Gravano and Garcia-Molina [9] proposed a centralized solution for database servers in which a central server keeps appropriate term and server statistics. These statistics are computed offline and at query time, a score for each server is computed assuming independence, and the servers with the highest scores are selected. Ricardo Baeza-Yates et al. studied the feasibility of geographically building multi-site search systems in [4] and B. Barla Cambazoglu etc. proposed query forwarding techniques under geographically distributed indexes [7]. Also in [5] YouSearch is proposed to efficiently search personal web servers.

There has been work on improved load balancing techniques using some degree of replication of documents stored on the servers [12, 6]. Pitoura *et al* [15] have studied the trade-off between load-balancing and latency using replication in P2P systems.

8. CONCLUSIONS

In this study, we explored the space of algorithms for index serving and generation strategies. We measured the performance of our strategies using three natural properties of a search index, *viz.*, quality, latency, and availability. First, we showed that the straightforward random allocation and selection strategies indeed produced good quality results comparable to the more complicated but better strategies such as source-based allocation and histogram selection. Second, we showed the benefit of replication on availability of the search index in surfacing relevant results to the user's query. We empirically showed that without some degree of replication, naive strategies of allocation and selection do not help in producing top-k results with some quality guarantees. In particular, we observed that for a given set of allocation and selection strategies, our greedy replication strategy helps in surfacing the results with the highest quality. Finally, we report that no replication strategy affects the throughput significantly and that the combination of random allocation and selection reduces throughput the least. Overall, we observe that the combination of random allocation and selection combined with greedy replication are a good choice of strategies for index organization.

9. REFERENCES

[1] GOV2 Dataset, TREC Terabyte Track. `http://www-nlpir.nist.gov/projects/terabyte/`.

[2] Job Shop Scheduling . `http://wikipedia.org/wiki/Job_shop_scheduling`.

[3] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

[4] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Plachouras, and L. Telloli. On the feasibility of multi-site web search engines. In *CIKM*, 2009.

[5] M. Bawa, R. J. B. Jr., S. Rajagopalan, and E. J. Shekita. Make it fresh, make it quick - searching a network of personal webservers. In *12th International World Wide Web Conference (WWW2003)*, 2003.

[6] J. W. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *IPTPS*, pages 80–87, 2003.

[7] B. B. Cambazoglu, E. Varol, E. Kayaaslan, C. Aykanat, and R. Baeza-Yates. Query forwarding in geographically distributed search engines. In *SIGIR*, 2010.

[8] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[9] L. Gravano and H. Garcia-molina. Generalizing gioss to vector-space databases and broker hierarchies. In *VLDB*, pages 78–89, 1995.

[10] A. Kulkarni and J. Callan. Document allocation policies for selective searching of distributed indexes. In *CIKM*, 2010.

[11] A. Kulkarni and J. Callan. Topic-based index partitions for efficient and effective selective search. In *SIGIR 2010 Workshop on Large-Scale Distributed Information Retrieval*, 2010.

[12] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[13] R. Motwani and P. Raghavan. Randomized algorithms. 1995.

[14] M. Persin, J. Zobel, and R. Sacks-davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47:749–764, 1996.

[15] T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, load balancing and efficient range query processing in dhts. In *EDBT*, pages 131–148, 2006.

[16] D. Puppin, F. Silvestri, and D. Laforenza. Query-driven document partitioning and collection selection. In *INFOSCALE 2006: Proc. of the first International Conference on Scalable Information Systems*, pages 107–117, 2006.

[17] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 42–49, 2004.

[18] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, pages 648–659, 2004.

[19] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *Proc. of SIGIR*, pages 254–261, 1999.