# IMPROVING THE EFFICIENCY OF DISTRIBUTED INFORMATION RETRIEVAL USING HYBRID INDEX PARTITIONING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FATİH HAFIZOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE 2018

Approval of the thesis:

**IMPROVING THE EFFICIENCY OF DISTRIBUTED INFORMATION RETRIEVAL USING HYBRID INDEX PARTITIONING**

submitted by **FATİH HAFIZOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halit Oğuztüzün _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. İsmail Sengör Altıngövde _____
Supervisor, **Computer Engineering, METU**

**Examining Committee Members:**

Prof. Dr. Özgür Ulusoy
Computer Engineering Dept., Bilkent University _____

Assoc. Prof. Dr. İsmail Sengör Altıngövde
Computer Engineering Dept., METU _____

Prof. Dr. Pınar Karagöz
Computer Engineering Dept., METU _____

**Date:** _____

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last Name: FATİH HAFIZOĞLU

Signature :

# ABSTRACT

## IMPROVING THE EFFICIENCY OF DISTRIBUTED INFORMATION RETRIEVAL USING HYBRID INDEX PARTITIONING

**HAFIZOĞLU, Fatih**
**M.S., Department of Computer Engineering**
**Supervisor : Assoc. Prof. Dr. İsmail Sengör Altıngövde**

**June 2018, 45 pages**

Selective search with traditional partitioning have advantages over exhaustive search in terms of total query cost. However, it can suffer from query latency and load imbalance for most of the time due to its nature. To overcome these issues, we proposed a new partitioning method in this thesis, namely Hybrid partitioning. Our studies shows that it is possible to obtain significant savings in query latency with this new partitioning methodology. In addition to that, query processing with Hybrid partitioning also achieves perfect load balancing and provides resource optimization, which is a key point for low resource environments.

Keywords: distributed search, selective search, document partitioning, document clustering

# ÖZ

## DAĞITIK BİLGİ GETİRME İÇİN MELEZ DOKÜMAN DAĞITIMI KULLANILARAK VERİMLİLİĞİN ARTTIRILMASI

**HAFIZOĞLU, Fatih**
**Yüksek Lisans, Bilgisayar Mühendisliği Bölümü**
**Tez Yöneticisi : Doç. Dr. İsmail Sengör Altıngövde**

**Haziran 2018 , 45 sayfa**

Geleneksel döküman dağıtımı kullanılarak yapılan seçmeli aramalar kapsamlı aramalarla kıyaslandığında toplam sorgu maliyeti açısından avantajlıdır. Fakat bu yöntem sorgu bekleme süresi ve yük dağılımı açısından çoğu zaman kötü sonuçlar verebilmektedir. Bu tezde bu sorunlarla baş edebilmek için melez döküman dağıtımı ismiyle yeni bir döküman dağıtımı yöntemi öneriyoruz. Çalışmalarımız melez döküman dağıtımı yöntemiyle sorgu bekleme süresi açısından ciddi kazançlar sağlamanın mümkün olabileceğini göstermektedir. Buna ek olarak, melez döküman dağıtımı yöntemı yük dağılımını da dengede tutabilmektedir ve kaynak kullanımında optimizasyon sağmamaktadır. Bu özellik kısıtlı kaynaklı ortamlarda çok önemlidir.

Anahtar Kelimeler: dağıtık arama, seçmeli arama, döküman dağıtımı, döküman kümeleme

*To my family and friends...*

# ACKNOWLEDGMENTS

I would like to thank my supervisor Assoc. Prof. Dr. İsmail Sengör Altıngövde for his guidance, help and enthusiastic approach during our work. It was a great honor and great opportunity for me to work with him.

I would also like to thank my friend and co-worker Emre Can Küçükoğlu. I appreciate his help and support.

And my parents who has been with me all my life and whose support me even in the worst moments. I cannot tell how lucky I am to have them. Thank you, mom and dad.

Few people have the privilege of having supportive and giving friends. I am lucky to have many of them. I would like to express my gratitude to Arda Taşcı, Deniz Çelik, Fatih Kurt, Ugur Temiz, Haluk Aktas, Ahmet Ozan Olgun and Hilal Gundog for being amazing friends and their constant support.

Finally, I would like to thank my girlfriend, Ece Sahin. My world is brighter and more beautiful because of you. Thank you.

This work is partially funded by the Ministry of Science, Industry & Technology of Turkey and Huawei Inc. under the grant no 0441.STZ.2013-2.

# TABLE OF CONTENTS

# LIST OF TABLES

TABLES

# LIST OF FIGURES

FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| BM25 | Best Matching - 25 |
| CSI | Centralized Sample Index |
| IR | Information Retrieval |
| DIR | Distributed Information Retrieval |
| KL | Kullback–Leibler |
| KLD | Kullback–Leibler Divergence |
| CORI | Collection Retrieval Inference Network |
| REDDE | Relevant Document Distribution Estimation |
| CRCS | Central-Rank-Based Collection Selection |
| CRCS(l) | Linear Central-Rank-Based Collection Selection |
| CRCS(e) | Exponential Central-Rank-Based Collection Selection |

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

Traditionally, a centralized index is used to search for a query inside a collection of documents. However, it is realized that keeping one-large index is not feasible in the real world because collection sizes are increasing exponentially. As a result of this, it is necessary to partition the collection and distribute the search task. With the help of the distribution, it will be possible to parallelize the search process.

Different partitioning methods are considered for enabling parallelization of the search process, straight forward method being the random partitioning. In this method, documents are randomly assigned to physical shards and a central broker is responsible to forward the query to all shards, collect the results from them and produce the final result by merging the partials. The entire process is considered as exhaustive search where all documents including query terms are processed.

Creating partitions based on the topics of documents is another method studied in IR community for many years. Since determining topics of the queries is also possible, only documents with relevant topics could be searched in this system. In this case, the broker needs to determine the relevant shards and forward queries to those shards only. This process, previously known as cluster-based retrieval [25] [1] for centralized IR scenario, is called selective search [24]. The purpose of the selective search is to produce high quality results while reducing the query processing cost.

Effectiveness of the selective search is examined in earlier works and findings show

that quality of the results is usually close to exhaustive search [24]. In addition, it is able to reduce total cost (i.e. total number of documents processed to produce final results) by scoring documents with relevant topics only. However, even if it reduces the total cost, efficiency is still questionable in terms of query latency. Since all documents with a particular topic are put together, it may cause increases in query latency. Besides, it is likely that there will be a load imbalance because the popularity of the topics is not same.

## 1.2 Contribution of Thesis

Query latency and load imbalance are the two main issues in selective search. Our main contribution in this work is to propose a solution that both improves query latency and provides load balance across shards.

As a solution, we developed a new partitioning method namely Hybrid partitioning. In this partitioning, we obtain the topics of documents but we do not use them to determine physical locations of them. Instead, we still assign documents to shards in a random manner. Inside shards, we used topics of documents to build cluster-skipping indexes [12] [1]. Efficiency improvements with cluster-skipping indexes have shown in centralized IR scenarios. In this work, we applied the same logic to distributed search setup.

We have used TREC and AOL query sets along with 4 different clustering strategies of the collection in order to be able to test partitioning methods in a variety of setups. We applied exhaustive search with random partitioning and selective search with Topic and Hybrid partitioning in each experiment and compare the results. In these experiments, we show significant savings compared to the traditional selective search in terms of the query latency while ensuring load balance.

Note that, we have presented our primarily findings in [19] and extended our experiments in this thesis.

## 1.3 Organization of Thesis

The organization of the thesis is as follows: Chapter 2 presents background information about the selective search. The common approaches in partitioning and resource selection methods are presented in detail. In Chapter 3, we present the proposed methods to improve the efficiency of the selective search. In Chapter 4, the details of the experimental setup (i.e., query sets, datasets, evaluation metrics and resource selection methods) are explained. Chapter 5 provides the experimental results and in Chapter 6, we present our conclusions and point to possible future work directions.

# CHAPTER 2

# RELATED WORK

In this chapter, we first present the document clustering approaches studied in the literature. Next, we review the methods for resource selection, which is one of the key stages in the selective search.

## 2.1 Document Clustering for Distributed IR

The volume of the information on the Internet is increasing rapidly. In addition to this, a large number of users are trying to access this information at the same time. Storing such a volume of information and keeping it available require distributed solutions.

Barroso et al. explain in their work that in order to build an effective solution, Google employs random partitioning for sharding and randomly assigns documents to the nodes that are distributed worldwide [7]. Indeed, partitioning collections to multiple nodes is a popular approach applied by both search engines and academic studies [8, 9, 26, 27]. Cahoon et al. evaluate different setups in prior researches. Setups of which sizes up to 128 GB distributed to servers that each contain approximately 1 GB. Single central broker used to manage these servers and servers are responsible for producing results locally. In simulations, they parametrize number of CPUs, number of disks and network utilization and evaluate different setups. Both previous studies and theirs share the common architecture with the primary goal of providing load balance and reducing response time at the same time. Results show that distributing documents among nodes can improve both. However, entire collection needs to be searched in order to produce final results.

Another distributed search approach followed in earlier studies is to cluster documents with their topical similarities. The main idea here is motivated by the *Cluster Hypothesis* introduced by Van Rijsbergen, which suggests that "Closely associated documents tend to be relevant to the same requests" [35]. That is, documents that are in the same clusters are most likely to be relevant to the query if their cluster is determined as relevant to the query in comparison to the documents in other clusters. Determining the relevant clusters is called resource selection phase in distributed information retrieval and will be described in next sub-section. Here, we focus on how to cluster documents based on their similarities.

**Algorithm 1** Clustering Algorithm with KL

**Input:** Collection C, Number of Shards K

**Output:** Clusters CL

1: RSD ← RANDSAMPLE(C, K) // Randomly sample K documents from C
2: CENTROIDS ← INITCENTROIDS(RSD) // Initialize centroids
3: **for** $D \in C$ **do**
4:  **for** $k \in \{1, \ldots, K\}$ **do**
5:  FIND $DIST(CL_i, D)$
6:  **end for**
7:  Assign $d$ to $CL_n$ where $CL_n$ is the closest cluster
8: **end for**
9: **if** first phase **then**
10:  CENTROIDS ← GENERATECENTROINDS(CL) // Re-generate centroids
11:  **goto** 2
12: **end if**

**return** $CL$

One of the challenging problems of clustering documents is the efficiency. Xu and Croft introduce an efficient clustering technique in their work [36] with linear time complexity. They employ a two-pass K-Means algorithm to cluster documents. In the first phase, random *K* documents are selected as the initial centroids of the clusters and for each remaining document, their algorithm (shown in Algorithm 1) finds the closest cluster for that document and assigns it to that cluster. In the second phase, results of the first phase are taken as the centroids of the clusters and the same process

is applied again. With the second phase, they intend to reduce the effect of random initials selected in the first phase and possible mistakes caused by them. In short, their approach applies the K-Means with only two passes. In order to find the distance between a document and a cluster, they use Kullback-Liebler divergence metric in Equation 2.1.

$$DIST(C^i, D) = \sum_{w \in D} \frac{c(w, D)}{|D|} log \frac{c(w, D)/|D|}{(c(w, D) + c(w, C^i))/(|D| + |C^i|)} \tag{2.1}$$

where

- $c(w, D)$ and $c(w, C^i)$ denote the number of occurrences of word $w$ on document $D$ and cluster $C^i$, respectively,
- $|D|$ and $|C^i|$ denote the size of document $D$ and cluster $Ci$, respectively.

In another study, Puppin et al. use query logs to cluster documents [29]. In their method, a query log applied with exhaustive search and obtained results are used for representing documents. Documents are weighted by the queries that retrieved them. Then, they co-cluster queries and documents accordingly. There are some major drawbacks in their method. First of all, some documents were not retrieved by any of the queries and they are grouped as another cluster. This issue could be solved by using more comprehensive query log but this time complexity will increase since it requires more and more queries to comprise all documents. Secondly, their method highly depends on the query log. Even if the query log comprises all the documents, it may still create unbalanced results in terms of document recalls and this will effect clustering results.

Kulkarni and Callan also employ the K-Means clustering approach in their studies [22, 23].They have modified some parts of the method described in [36] (Algorithm 1 and Equation 2.1). Instead of calculating distances between cluster centroids and documents with KL divergence, they used the symmetric version of the method to calculate the similarities between them. While doing this, they also fixed some issues of Equation 2.1. First of all, they noticed that the equation is heavily biased on clusters with fewer lengths. Secondly, since the equation is not symmetric, centroids of the

clusters have very little effect on calculated distance while the importance of a term inside documents dominates the distance value. And finally, they draw attention that term weighting is not available in this formula. There should be inverse document frequency and inverse collection frequency in the formula to calculate more accurate values. In the light of these observations, they propose a new equation to calculate the similarity of a document to a cluster centroid (Equation 2.2).

$$SIM(C^i, D) = \sum_{w \in C^i \bigcap D} {p_c}^i(w) log \frac{p_d(w)}{\lambda p_B(w)} + p_d(w) log \frac{{p_c}^i(w)}{\lambda p_B(w))} \tag{2.2}$$

$${p_c}^i(w) = c(w, C^i) / \sum_{w'} c(w^{'}, C^i) \tag{2.3}$$

$$p_d(w) = (1 - \lambda) c(w, D) \sum_{w'} c(w^{'}, D) + \lambda p_B(w) \tag{2.4}$$

where

- $c(w, D)$ and $c(w, C^i)$ denote the number of occurrences of word $w$ on document $D$ and cluster $C^i$, respectively,
- $p_B(w)$ is the inverse frequency of *w* inside the collection.

Even if it has a linear time complexity, applying K-Means algorithm to a large collection could be still very costly. Kulkarni and Callan have made another change in here and instead of applying K-Means to the entire set, they decided to apply it to a sampled subset of the collection. Once the centroids of the clusters are determined over the sample, remaining documents could be clustered more easily. In their algorithm (Algorithm 2), K-Means is applied to a randomly selected subset in 5 runs to determine centroids of the clusters with setting centroids randomly at the start. Reducing the set which K-Means applied to also provides the opportunity for multiple runs with adding affordable costs. Then they dispatched the remaining documents to clusters based on their similarities with cluster centroids. While calculating the

similarity between a document to cluster in both centroid determining phase and dispatching phase, the modified version of Kullback-Liebler divergence (Equation 2.3) is used in their work.

**Algorithm 2** Sample Based Clustering Algorithm with KL-SIM

**Input:** Collection C, Number of Shards K, Sample Percentage P

**Output:** Clusters CL

1: SC ← RANDSAMPLE(C, SIZE(C) * P) // Randomly sample %P of documents from C
2: RSD ← RANDSAMPLE(SC, K) // Randomly sample K documents from SC
3: CENTROIDS ← INITCENTROIDS(RSD) // Initialize centroids
4: **for** $D \in SC$ **do**
5:  **for** $k \in \{1, \ldots, K\}$ **do**
6:  FIND $SIM(CL_i, D)$
7:  **end for**
8:  Assign $d$ to $CL_n$ where $CL_n$ is the most similar cluster
9: **end for**
10: **if** In first fourth phase **then**
11:  CENTROIDS ← GENERATECENTROINDS(CL) // Re-generate centroids
12:  **goto** 4
13: **end if**
14: **for** $D \in C \backslash SC$ **do**
15:  **for** $k \in \{1, \ldots, K\}$ **do**
16:  FIND $SIM(CL_i, D)$
17:  **end for**
18:  Assign $d$ to $CL_n$ where $CL_n$ is the most similar cluster
19: **end for**

**return** $CL$

Dai et al. show that random decisions in K-Means clustering may result in misclustered documents [17]. In order to improve the clustering strategy and prevent possible mistakes due to random decisions made, they revise the methodology and propose two new strategies in [18].

The first one is mentioned as query-driven clustering initialization algorithm (QInit) and developed for preventing possible errors related to random cluster centroid initializations. In this method, they extract terms from query logs and cluster each term with its corresponding word embeddings. Thus, terms for similar user search topics are grouped together.

In addition to QInit, they also propose query-biased similarity metric (QKLD) in their work. It is based on symmetric KLD also used in [22]. They suggest that Equation 2.2 works well in terms of topics in document corpus but it is insufficient in reflecting user query topics. To reflect the latter, they add a weight to the equation that shows the importance of a term in Equation 2.5. In particular, $(\alpha_q(w) + b)$ part of the equation adds the term importance in query log to the equation, while the remaining of it has no difference than Equation 2.2.

$$SIM_{QKLD}(C^i, D) = \sum_{w \in C^i \bigcap D} (\alpha_q(w)+b) \times ({p_c}^i(w) log \frac{p_d(w)}{\lambda p_B(w)} + p_d(w) log \frac{{p_c}^i(w)}{\lambda p_B(w))}) \tag{2.5}$$

$$\alpha_q(w) = log(wf_{w,Q} + 1) \times log(\frac{|D|}{df_{w,D}} + 1) \tag{2.6}$$

where

- $wf_{w,Q}$ and $df_{w,D}$ denote the term frequencies of word $w$ on query log and documents in the corpus, respectively,
- $|D|$ is the number of documents in total, and
- $b$ is the smoothing parameter.

With QInit and QKLD, they produce 4 different clustering for the dataset with combinations of them. These are constructed as follows:

- **KLD-Rand:** Random seeds in cluster centroid initialization and KLD similarity metric used in Equation 2.2.

- **KLD-QInit:** Query-driven cluster centroid initialization and KLD similarity metric used in Equation 2.2.
- **QKLD-Rand:** Random seeds in cluster centroid initialization and QKLD similarity metric used in Equation 2.5.
- **QKLD-QInit:** Query-driven cluster centroid initialization and QKLD similarity metric used in Equation 2.5.

In their experiments, they found that KLD-QInit clusters have no differences than KLD-Rand clusters and exclude it from the testbed. Other than this, they obtain significant improvements in both efficiency and effectiveness with these new methodologies. The most effective and efficient results are obtained with QKLD-QInit which uses both QInit and QKLD in cluster centroid initialization and similarity function respectively. Since the clustering structure they have produced is available online[1] we used them in our work to evaluate the performance of our index partitioning approaches.

## 2.2 Resource Selection

Another important phase of distributed information retrieval with selective search is resource selection (i.e., finding the relevant clusters). Even if perfect clustering could be achieved, it will be meaningless without a proper logic to identify the clusters that should be conducted for a given query. On this subject, we will cover 6 different resource selection algorithms in this section. These are categorized as large-document models and small-document models. As an example of large-document models, CORI [10] will be discussed. On the other hand, Redde [34], CRCS(e) and CRCS(l) [32], GAVG [31] and Redde.top [5, 6] will be covered as examples of small-document resource selection approaches.

In CORI, a cluster is represented as bag-of-words that are the concatenation of the containing documents [10]. Since each collection is represented as a document, cluster selection algorithm works just like a document retrieval algorithm with taking the

[1] http://boston.lti.cs.cmu.edu/appendices/CIKM2016-Dai/

weight of the term in the collection as term frequency and inverse collection frequency as inverse document frequency in the BM25 formula. The result of the resource selection algorithm is the rankings of the documents that are representative of certain clusters. Then, top-K clusters can easily be selected and the query can be forwarded those clusters.

The motivation for the emergence small-document models is the observation that CORI is not working well with mixed sized collections [34]. In order to overcome this issue, Si and Callan proposed a new method, namely, Redde that inspired many more methods afterward.

In Redde, a set of documents is randomly selected from each cluster. These documents grouped as a special cluster. Then, an index is created for this cluster. This index is referred as Central Sample Index (CSI) and plays a big role in resource selection algorithms of small-document models. In query processing phase, query is first processed over CSI. Next, clusters are scored based on the number of documents it has in the top-N results. At last, scores of the clusters are scaled with the proportion of the size of sampled documents to the size of the original cluster. The formula for calculating scores of the clusters is given in Equation 2.7.

$$SCORE(C^i, q) = \frac{|C^i|}{|S^i|} \times \sum_{d \in S^i, d \in R} 1 \tag{2.7}$$

where

- $S^i$ is the sampled documents from cluster $C^i$,
- $R$ is the top *N* result obtained from CSI,
- $|C^i|$ and $|S^i|$ are the sizes of the cluster and sampled documents of the cluster respectively.

To improve Redde, Shokouhi proposed two new methods, namely, CRCS(l) and CRCS(e) [32]. They followed the same path as Redde but changed the scoring algorithm of the clusters. In their work, they stated that not each document of the top *N* CSI result should make the same contribution to the cluster score and ranks of these

documents should be taken into consideration. To enable this, two new equations are introduced in their work. First one, CRCS(l), takes the position of the document linearly in its formula shown in Equation 2.8. On the other hand, CRCS(e) takes the rank exponentially in Equation 2.9.

$$SCORE(C^i, q) = \frac{|C^i|}{|S^i|} \times \sum_{d \in S^i, d \in R} N - j \tag{2.8}$$

$$SCORE(C^i, q) = \frac{|C^i|}{|S^i|} \times \sum_{d \in S^i, d \in R} \alpha \epsilon^{-\beta \times j} \tag{2.9}$$

where

- $j$ is the rank of document $d$ in CSI result, and
- $\alpha$ and $\beta$ are the coefficient parameters which are set to 1.2 and 2.8, respectively, in their experiments.

Instead of using document ranks, Seo and Croft decided to use the scores of the documents to calculate the scores of the clusters. To do this, they took the first *m* results for each cluster in CSI result and calculated the geometric average of the scores of these documents. If a cluster has less than *m* documents in CSI result, the minimum scored document added until reaching *m*. The formula for calculating cluster scores is given in Equation 2.10.

$$SCORE(C^i, q) = (\prod_{d \in S^i, d \in \text{top m of} S^i \in R} P(d, q))^{\frac{1}{m}} \tag{2.10}$$

where

- $P(d, q)$ is the score of document $d$ in query $q$

Another method that uses the relevance scores of the documents in resource selection is proposed by Arguello et al. and later called as Redde.top [5, 6]. This is also a variant of Redde. Instead of using ranks or geometric averages of the scores of the

documents, it uses the summation of the scores. The equation of cluster scoring formula is given in Equation 2.11

$$SCORE(C^i, q) = \frac{|C^i|}{|S^i|} \times \sum_{d \in S^i, d \in R} P(d, q) \tag{2.11}$$

In our experiments, we use Redde as it is widely used in related works that focus on selective search [22, 24].

## 2.3 Cluster Skipping Indexes

Cluster skipping indexes are initially introduced by Can et al. in [12]. In their work, they proposed a new structure to store inverted index files in order to improve the performance of cluster based retrieval. In this new structure, cluster membership information of documents are also stored in index files along with postings. With this way, it becomes possible to process documents from best clusters and skip the others.

Later, Altingovde et al. adapted this structure to environments with compression [1]. This structure is also used in dynamic pruning [2] and recommendation systems [3]. However, our implementation is the first one that applies this idea to selective search within a distributed information retrieval setup.

# CHAPTER 3

# IMPROVING PARTITIONING TECHNIQUES FOR SELECTIVE SEARCH

This chapter presents the new techniques proposed in this thesis in order to improve selective search. In the first section, we will discuss selective search along with the techniques used in general. Next, Hybrid partitioning method will be explained in detail. Finally, we will introduce the distributed CSI method to further improve search efficiency.

## 3.1 Selective Search

Distributed search is a well-known approach for searching large textual collections with partitioning documents to multiple shards. Several methods are proposed to partition the documents and the index across physical shards.

Random partitioning is one of them and it assigns documents to shards uniformly at random [11]. It is the most common method used in industry. Search process in the Random partitioned setup is simple as shown in Figure 3.1. The colors in the figure show the topics of the documents. For random partitioned setup, topics of the documents are not important. It just demonstrates that different documents are kept together in this method. When a query is submitted to the broker, it directly forwards the query to shards and waits for top-k results from each shard. After getting the results from all shards, broker merges them and produces the final result. Since it searches all the documents inside all the shards, this method is considered as exhaustive search over the entire collection and produces the same results as if it is conducted

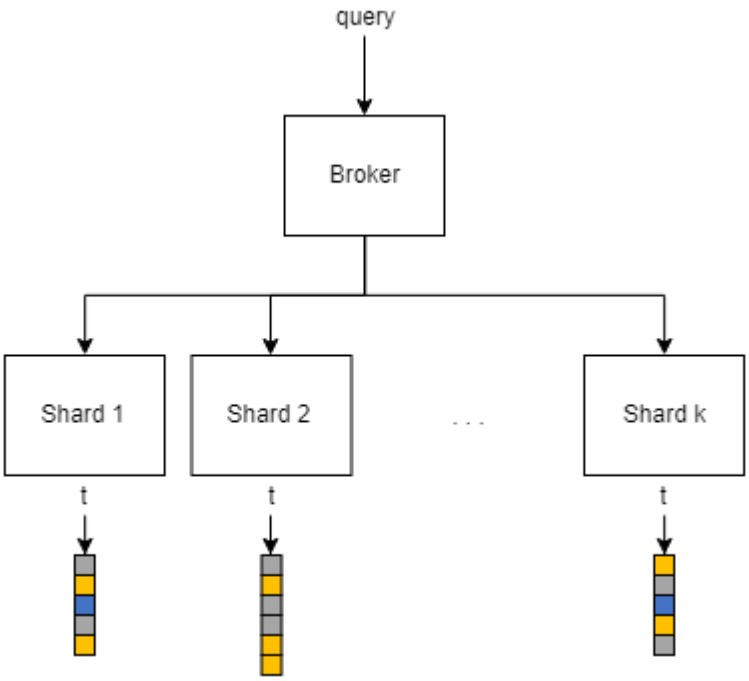



Figure 3.1: Search process in Random partitioned setup.

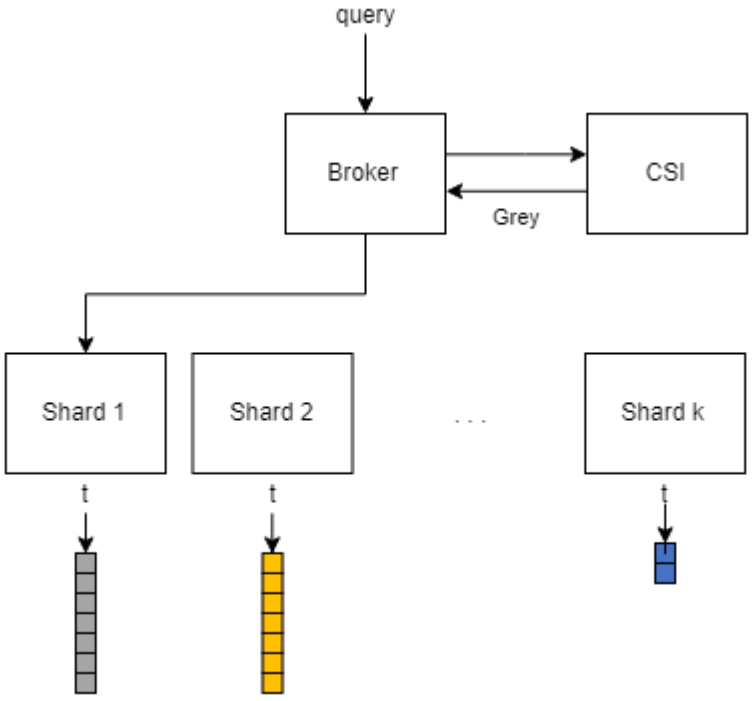



Figure 3.2: Search process in Topic partitioned setup.

using a single index on the entire collection. Notably, as search process is distributed to multiple shards, it reduces the query latency.

Another method used in the distributed search is Topic partitioning. In this method, documents are distributed to shards according to their topics and hence, topically similar documents are kept together. A sample index, namely, CSI, is created after the shard creation stage [4] [24]. The query processing steps can be seen in Figure 3.2. When a query reaches to the broker, it is first processed over CSI. From the top-k results of CSI results, most relevant shards to the query are determined using resource selection algorithms [33]. Next, query is forwarded to only these relevant shards. Each shard produces its own top-k results and returns them to the broker. In the end, returning results are merged in the broker and final top-k results are produced. In Figure 3.2, CSI processing yields that only documents with "Grey" topic are relevant to the query and then query is forwarded to the shard which contains "Grey" documents.

The main motivation in selective search is reducing the costs in query processing while maintaining the result quality in comparison to exhaustive search. Previous studies [12], [4], [22] and [25] have shown that this is possible with various clustering and resource selection strategies. In terms of the total cost, i.e., total number of documents visited for a query, it is clear that cost can be reduced by selecting $N$ relevant shards (skipping $K - N$ shards) for a query. However, efficiency is still questionable and studied by others [24]. Although total cost is valuable for showing the total work done, longest execution path, i.e., query latency is a more important metric than the total cost since it directly affects user experience. The latency is determined by the slowest shard because broker needs to collect results from all shards before merging them. In the environments with unbalanced cluster sizes, the longest execution path gaps between fastest and slowest shards become larger. To overcome this issue, Kulkarni and Callan proposes the size bounded clusters that helps to produce more balanced clusters. However, it still suffers in query latency compared to random partitioning. As explained in Section 4, we will use $C_{Total}(q)$ metric to study total cost and $C_{Latency}(q)$ metric to study query latency and compare the results.

Another issue which may show up in Topic partitioning is load imbalance. It is very likely that some of the shards are working for most of the time for popular queries

while the others may work occasionally for less popular queries, and otherwise, remain idle. As result of this, resources may be wasted, which is undesirable in a low-resource search setup. Kim et al. draw attention to this issue in their work [21] and propose better shard distributions as a solution.

In our work, we build a new solution that produces effective results with reducing total cost like Topic partitioning but fixing the query latency and load imbalance problems at the same time.

### 3.2 Selective Search with Hybrid Partitioning

In the previous section, we mentioned the two main problems of selective search: Query Latency and Load Imbalance. The first one has an impact on user experience while the second one causes troubles in resource utilization. As a solution, we propose the Hybrid partitioning approach that still uses the topical clusters of the documents but partition the documents randomly.

To achieve this goal, we inspired from the cluster-skipping index method [1] [12]. This method has been shown to improve both efficiency and effectiveness in single shard setups. To enable cluster-skipping indexes in our setup, we first clustered the documents and assigned each document to a cluster, i.e., a topic. Next, we distributed documents randomly to the shards like Random partitioning. Inside shards, topics of the documents are used to group them and shard indexes are created accordingly. In other words, we distribute documents randomly to the physical shards but exploit their logical clusters to create cluster-skipping indexes. While processing a query, this method helps us to obtain a better load balance while reducing the number of postings processed at each random shard.

The summary of the query processing steps could be seen in Figure 3.3. When a query arrives to the broker, it is first conducted to CSI to determine the relevant clusters. This resource selection step is exactly same as the one in Topic partitioning setup. Next, the query is directed to all shards. It can be seen from the figure that postings in a list are grouped by their logical clusters and every list start with cluster-skipping nodes that are pointing the starts of documents belongs to a particular cluster. In this

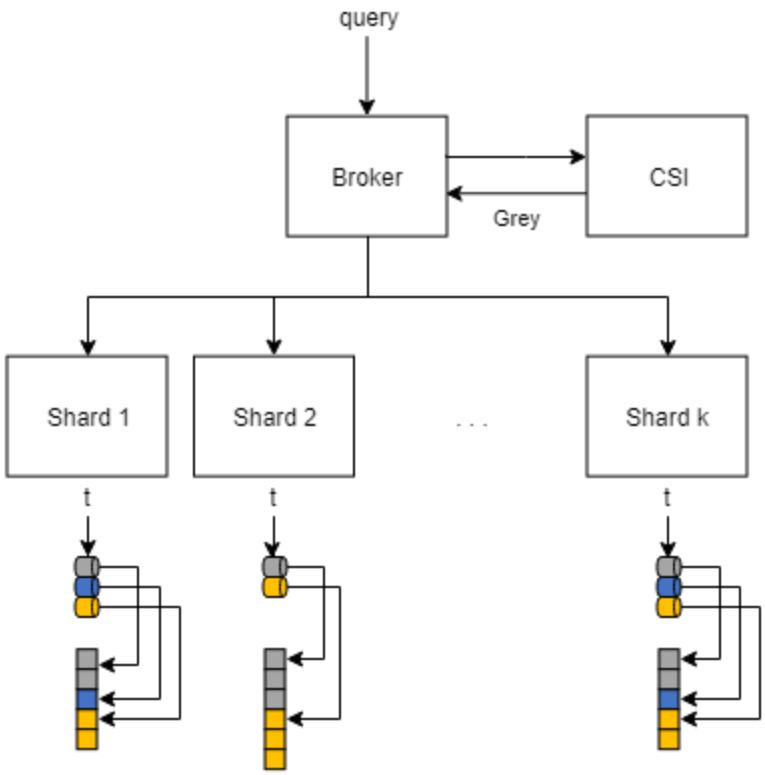



Figure 3.3: Search process in Hybrid partitioned setup.

way, it is possible to skip the documents that do not belong to the clusters identified in the resource selection stage. Since all shards work on the query at the same time, loads become more balanced compared to Topic partitioned setup. In addition to this, not all postings processed inside shards which helps to keep the total cost low.

To understand the differences of these three setups in terms of query processing steps, an example is illustrated in Figures 3.1, 3.2 and 3.3. Assuming a single word query $q$ with term $t$ executed in all of these setups, we compute the query latency and total cost. In the first setup, Random partitioned, the query is forwarded to all shards and all posting lists inside shards for term $t$ is processed. This means total cost, i.e., the total number of postings processed, is 16 for this method, while query latency is 6 and determined by slowest shard - $Shard2$. In Topic partitioned setup, the query is only forwarded to $Shard1$ because CSI results indicate that "Grey" cluster is relevant to the query. This time the total cost and query latency are determined by $Shard1$ and that is both 7. When it comes to Hybrid partitioned setup, query is forwarded to all shards but before that, relevant clusters are again determined with CSI processing.

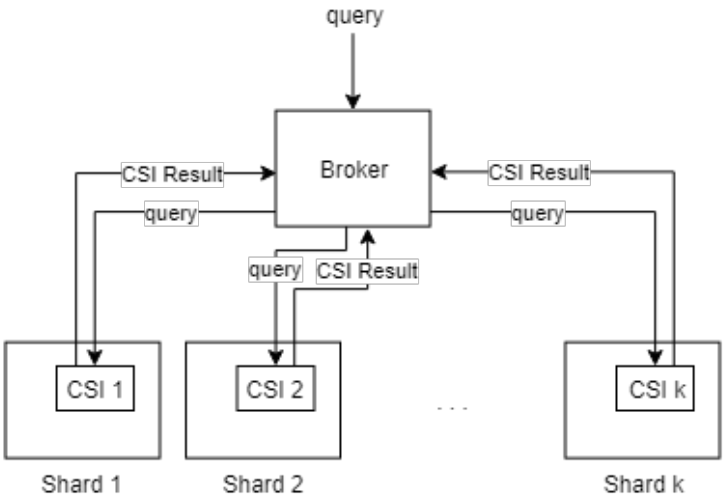



Figure 3.4: First phase of query processing in the distributed CSI setup.

In each shard, cluster skipping nodes and postings that are pointed by "Grey" cluster-skip node is processed. This makes the total cost of the method as 15 and query latency as 5 (3 skip nodes + 2 "Grey" postings on $Shard1$). In this toy example, the latency of Hybrid method could be worse if the query topic was "Blue" but we emphasize that the number of skipping nodes would be much less compared to posting list size in practice (as number of clusters would be much smaller than the number of documents) and this additional cost would not adversely affect the performance gains obtained via Hybrid partitioning. In contrary, since all the documents related to a particular topic are stored in same shard in Topic partitioned setup, query latency could be much worse in certain cases. We remove the possibility of such cases by distributing topically related documents to all shards.

### 3.3 Selective Search with Distributed CSI

While creating CSI, we mentioned that 1% of documents are randomly sampled from all clusters. Since the first step of query processing in Topic and Hybrid partitioned setups is sending the query to CSI, we added the processing cost of this stage to query latency (Eq. 4.2) and total cost (Eq. 4.1) formulas, as typical in the literature [24]. Thus, for each query, the system has to do calculations in CSI whose size is 1% of the index for the entire collection. This implies that while trying to reduce the query

latency, CSI processing cost is dominating the overall cost, because our improvements affect only the costs inside the shards, but not CSI.

Putting all together, we decided to distribute CSI to the physical shards (again, in a random fashion) to reduce its cost and take advantage of parallel processing. In distributed CSI setups, every physical shard of the system stores a subset of the CSI. Query processing is done in two-phase. In the first phase, the query is forwarded to all shards and it is evaluated over the shards' local CSIs. Next, results are collected and merged in broker and clusters relevant to the query are determined accordingly (Figure 3.4). In the second phase, the query is only forwarded to certain shards in Topic partitioned setup and forwarded to all shards in Hybrid one (together with the information about "target" clusters).

# CHAPTER 4

# EXPERIMENTAL SETUP

We will introduce the experimental setup used for the experiments in the next chapter. To begin with, we will explain the query processing method. The next section will be about the query sets. After that, evaluation metrics and partitioning methods will be described one after another.

## 4.1 Query Processing

We used Zettair[1] to construct the cluster and CSI indexes. Before processing queries, we remove the stopwords. Okapi BM25 scoring scheme [30] is used to score documents with setting parameters $k1$ to 1.2 and $b$ to 0.5.

## 4.2 Query Sets

Two query sets used in the experiments. TREC set consists of 200 queries from TREC Web tracks between 2009 and 2012 [13, 14, 15, 16]. Query lengths in this set vary from 1 to 10 with the average length of 2.1. The second one, AOL set, consists of 100K queries that are randomly selected from the well-known AOL log [28]. Minimum query length is 1 and maximum query length is 85 for this set with the average length of 3.1. Relevance judgments are available for TREC set and used for evaluations in this thesis. In order to evaluate the results for AOL set in terms of effectiveness, we used top 20 exhaustive results as relevant documents for a query

[1] http://www.seg.rmit.edu.au/zettair/

and evaluate accordingly.

### 4.3 Evaluation Metrics

**Effectiveness.** The effectiveness results in this thesis are evaluated using Precision (P@5, P@10 and P@20), Normalized Discounted Cumulative Gain (NDCG@20) and Expected Reciprocal Rank (ERR@20) metrics. As mentioned in the previous section, relevance judgments that are published for TREC Sets are used to evaluate them. On the other hand, we created custom relevance judgments from exhaustive search results for AOL Set. All metrics are calculated with Trec Eval[2] or nDeval[3].

**Efficiency.** Efficiency results are calculated in terms of total query cost ($C_{Total}(q)$) and query latency ($C_{Latency}(q)$) as in [24].

$$C_{Total}(q) = |I^q_{CSI}| + \sum_{i=1}^{M} \sum_{c \in T} |I^q_{M[i],c}| \tag{4.1}$$

$$C_{Latency}(q) = \max_{1 \leq i \leq M} (|I^q_{CSI}| + \sum_{c \in T} |I^q_{M[i],c}|) \tag{4.2}$$

where

- $|I^q|$ is the total number of posting lists for each term in q,
- $|I^q_{CSI}|$ is the query processing cost in CSI,
- $|I^q_{M[i],c}|$ is the query processing cost for a cluster $c$ located at a physical machine $M[i]$,
- $T$ is the set of selected clusters for a query.

$$M_{[i],Total} = \sum_{c \in T} |I^q_{M[i],c}| \tag{4.3}$$

[2] https://github.com/usnistgov/trec_eval/
[3] https://github.com/trec-web/trec-web-2014/blob/master/src/eval/ndeval.c

Table 4.1: Experiments on clusters sizes

| K | P@5 | P@10 | P@20 | NDCG@20 | ERR@20 |
|---|---|---|---|---|---|
| 50 | 0.3263 | 0.3121 | 0.2793 | 0.17972 | 0.12166 |
| 100 | **0.3343** | **0.3152** | **0.2811** | **0.18406** | **0.12316** |
| 250 | 0.3071 | 0.2965 | 0.2694 | 0.17507 | 0.11709 |

In addition to this, we also keep the physical machine statistics in terms of the total work done to calculate the loads per machine $M[i]$.

### 4.4 Collection and Partitioning

We used ClueWeb09 Cat-B collection in our experiments. This dataset consists of 50 million English web pages and it is widely used in distributed search setups[4].

To create *Topical* clusters, we used the sample-based K-means algorithm described in Chapter 2. Since applying the algorithm to the whole set would be costly, we have randomly sampled a subset of documents, namely 1% of the collection, as in the work of Kulkarni and Callan [22]. $K$ documents are selected to initialize the cluster centroids from these sampled documents randomly. Next, multiple K-Means runs are executed to produce the final clustering of sampled documents. For each run after the first one, previous run results used to determine the initial centroids of the clusters and the same process applied again. Then the remaining documents are projected to these clusters based on their similarities to cluster centroids. Kullback-Liebler divergence method (Equation (2.2) [22]) is used to calculate the similarity between the cluster centroids and documents.

In order to choose parameter *K*, we tried 3 different values as 50, 100 and 250. 200 queries from the TREC set between 2009 and 2012 are processed in these clusters with Redde as resource selection algorithm and 10% of the resources are selected in each run. The result of the experiments can be seen on Table 4.1. As $K = 100$ produces the best results in our experiments and to obtain comparable results with the previous studies, the parameter *K* is set to 100.

[4] http://lemurproject.org/clueweb09.php/

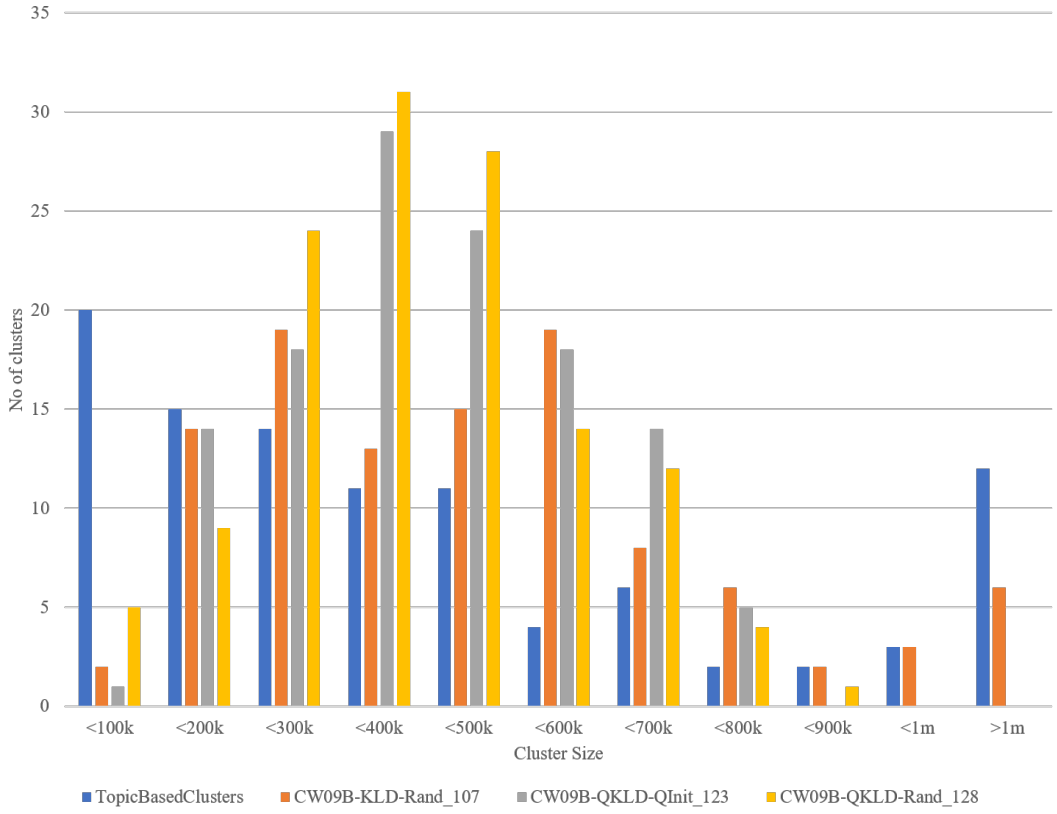



Figure 4.1: Size distribution of the clusters.

We have also used the *CW09B-KLD-Rand*, *CW09B-QKLD-Rand* and *CW09B-QKLD-QInit* clusters created by Dai et al. using the same dataset with different clustering techniques [18]. *CW09B-KLD-Rand* is created with random seeds and Kullback-Lieber divergence as similarity function, just like our clustering. In *CW09B-QKLD-Rand*, they add term importance which is generated from query logs and word embeddings of the terms to Equation (2.2) [22]. For the final one, *CW09B-QKLD-QInit*, they also changed the seed selection stage of the algorithm. Instead of randomly choosing documents to create initial centroids of the clusters, they again used query logs and word embeddings of the terms to create better centroids. The resulting clusterings consist of 107, 128 and 123 clusters, respectively.

Figure 4.1 shows the size distributions of the clusters. While *Topical* clusters (our setup) and CW09B-KLD-Rand datasets have large number of small-sized and large-sized clusters, CW09B-QKLD-QInit and CW09B-QKLD-Rand datasets have more mid-sized clusters. In addition to this, the distribution of the clusters in *Topical* clus-

Table 4.2: Percentage of relevant documents covered in top *N* clusters

| Dataset | N | | | | |
|---|---|---|---|---|---|
| | 1 | 3 | 5 | 10 | 10% |
| Our | 55 | 84 | 93 | 99 | **99** |
| CW09B-KLD-Rand | 58 | 84 | 91 | 98 | **99** |
| CW09B-QKLD-Rand | 65 | 89 | 97 | 99 | **99** |
| CW09B-QKLD-QInit | 67 | 90 | 97 | 99 | **100** |

ters are very similar to the one in [20].

We have also calculated the cluster coverage percentages as another analysis on the clusters in order to make sure that the topics are well distributed across the clusters. To calculate the coverage percentages, we used 200 queries from TREC 2009-2012 sets with relevance judgments. For each query, we looked for the clusters of the relevant documents and sorted them with respect to the number of relevant documents covered. Next, for each cluster in order, we calculated the percentage of documents covered incrementally. Table 4.2 summarizes the coverage percentages with respect to the top *N* clusters. The last column of the table shows that top 10% percent of clusters covers at least 99% of the relevant documents. This analysis also helps us to determine the number of clusters to be selected in resource selection algorithms which is the topic of the next sub-section.

In addition to Topic partitioned setups we also created Hybrid partitioned ones. For each clustering structure used in Topic partitioned setups, we partitioned documents using the proposed Hybrid approach. For this method, documents are distributed randomly to the clusters, but we are keeping their topics as logical clusters that documents belong. With this way, we are able to construct indexes with cluster-skipping elements which are described in the previous chapter.

Finally, we have Random partitioned setup that documents assigned randomly to the shards. Here we used the same distribution used in Hybrid partitioned setup in order to have comparable results in Random and Hybrid partitioned setups.

In a low-resource setup, we are aware of the fact that we can't use a physical node for each cluster. So, we assume that there are *M* physical machines where $M < K$ as

Table 4.3: Experiments on the number of selected resources $N$ with different $K$ values

| K | N | p@5 | p@10 | p@20 | NDCG@20 | ERR@20 |
|---|---|---|---|---|---|---|
| - | - | 0.3434 | 0.3157 | 0.2833 | 0.18585 | 0.12555 |
| 50 | 1 | 0.2505 | 0.2283 | 0.1813 | 0.12219 | 0.10259 |
| | 3 | 0.3162 | 0.2980 | 0.2566 | 0.16540 | 0.11702 |
| | 5 | **0.3263** | **0.3121** | **0.2793** | **0.17972** | **0.12166** |
| 100 | 1 | 0.2758 | 0.2490 | 0.1992 | 0.14478 | 0.10898 |
| | 3 | 0.3242 | 0.3030 | 0.2551 | 0.17067 | 0.12265 |
| | 5 | 0.3303 | 0.3182 | 0.2722 | 0.17696 | 0.12088 |
| | 10 | **0.3343** | **0.3152** | **0.2811** | **0.18406** | **0.12316** |
| 250 | 1 | 0.2616 | 0.2429 | 0.1876 | 0.12645 | 0.09908 |
| | 3 | 0.2879 | 0.2768 | 0.2341 | 0.15203 | 0.10964 |
| | 5 | 0.2980 | 0.2833 | 0.2465 | 0.16438 | 0.11406 |
| | 10 | 0.2970 | 0.2919 | 0.2609 | 0.16834 | 0.11220 |
| | 25 | **0.3071** | **0.2965** | **0.2694** | **0.17507** | **0.11709** |

in [21]. For partitioning, we randomly assigned the clusters to the physical nodes. In the end, each physical node hosts $K/M$ clusters [24]. In this work, we choose M as 10.

### 4.5 Resource Selection

We used Redde [34] as the resource selection method in our experiments. To begin with, we create a Central Sample Index (CSI) by randomly sampling the documents from shards. Aly et al. tried sampling 1%, 2% and 4% of the documents in the same dataset and observed very similar results with different sample rates [4]. Here, we also decide to sample 1% of the documents based on the previous studies [4] and [24].

To determine how many shards should be visited for a query, i.e., the parameter *N*, we try different values and expand the experiment in Table 4.1. The results can be seen on Table 4.3. The first line of the table shows the exhaustive search results. We can say from these results that selecting 10% percentage of the shards is enough to

produce effective results as much as the exhaustive search does.

For Topic and Hybrid partitioned setups queries are first executed over CSI and then selected resources are determined with Redde. After that, the query is forwarded to the selected shards in Topic partitioned setup. On the other hand, in the Hybrid partitioned setup, the query is forwarded to all shards along with the information of selected resources to allow skipping. On Random partitioned setup, CSI processing skipped and query forwarded to all shards.

# CHAPTER 5

# EXPERIMENTAL RESULTS

Several experiments are conducted to evaluate the performance of the Hybrid partitioning. In what follows, we present the results in terms of effectiveness and efficiency metrics that are described in Chapter 4.

## 5.1 Effectiveness

We have used two different query sets to measure the effectiveness of selective search and compare it with exhaustive search.

Table 5.1 shows the results of TREC query set. This set consists of 200 queries from 2009 to 2012 query sets and their relevance judgments are available. As it can be seen from the results, the selective search produces comparable results with the exhaustive search. Our results are also similar to [24], but the scores are better than theirs because of the differences in the pre-processing methods used. In particular, we use spam filtering while they apply stemming.

In addition to TREC query set, we also used 100K queries from AOL log to test the partitioning methods in large scale. Since we didn't have the relevance judgments of AOL query set, we used exhaustive search results to evaluate them. For doing this, first 20 results of the exhaustive search are assumed as relevant and selective search results are evaluated based on them. The results can be seen in Table 5.2. At least 98% of the documents from top-5 results of the exhaustive search also occurs in top-20 of selective search results for TREC query set. The precision score is 94% for

top-10 results and 78% for top-20 results respectively. AOL query set results have slightly worse results than TREC query set results as it can be seen from the last 4 rows of Table 5.2.

Table 5.1: Effectiveness results for TREC query set with selective search (Topic & Hybrid partitioning) and exhaustive search (Random partitioning).

| Partitioning | Dataset | K | N | p @5 | p @10 | p @20 | ndcg @20 | err @20 |
|---|---|---|---|---|---|---|---|---|
| Random | - | 100 | NA | 0.34 | 0.32 | 0.28 | 0.19 | 0.13 |
| Topic & Hybrid | Our | 100 | 10 | 0.34 | 0.32 | 0.28 | 0.18 | 0.12 |
| | KLD-Rand | 107 | 11 | 0.33 | 0.31 | 0.29 | 0.18 | 0.13 |
| | QKLD-Rand | 128 | 13 | 0.34 | 0.32 | 0.28 | 0.18 | 0.13 |
| | QKLD-QInit | 123 | 12 | 0.33 | 0.32 | 0.29 | 0.19 | 0.13 |

In conclusion, we found similar results as previous works like [24] and [18] revealing that selective search can be as effective as exhaustive search. The differences in topical clustering stage have a minor impact in the effectiveness results but more balanced shards are likely to yield better efficiency and load balancing results, as will be shown in the next sections.

Table 5.2: Recall analysis for selective search results with Topic & Hybrid partitioning.

| Query Set | Dataset | K | N | p @5 | p @10 | p @20 | ndcg @20 | err @20 |
|---|---|---|---|---|---|---|---|---|
| TREC 200 Query | Our | 100 | 10 | 0.98 | 0.94 | 0.78 | 0.84 | 0.17 |
| | KLD-Rand | 107 | 11 | 0.99 | 0.96 | 0.80 | 0.86 | 0.17 |
| | QKLD-Rand | 128 | 13 | 0.98 | 0.96 | 0.84 | 0.88 | 0.17 |
| | QKLD-QInit | 123 | 12 | 0.99 | 0.95 | 0.84 | 0.88 | 0.17 |
| AOL 100K Query | Our | 100 | 10 | 0.93 | 0.86 | 0.63 | 0.73 | 0.16 |
| | KLD-Rand | 107 | 11 | 0.92 | 0.85 | 0.65 | 0.73 | 0.16 |
| | QKLD-Rand | 128 | 13 | 0.93 | 0.87 | 0.70 | 0.77 | 0.16 |
| | QKLD-QInit | 123 | 12 | 0.93 | 0.87 | 0.71 | 0.77 | 0.16 |

Table 5.3: Efficiency results for TREC query set with selective search (Topic & Hybrid partitioning) and exhaustive search (Random partitioning). $C_{Total}$ and $C_{Latency}$ are in million documents.

| Partitioning | Dataset | K | N | $C_{Total}$ | $C_{Latency}$ |
|---|---|---|---|---|---|
| Random | - | 100 | NA | 4.579 | 0.46 |
| Topic | Our | 100 | 10 | 1.831 | 0.69 |
| | KLD-Rand | 107 | 11 | 1.268 | 0.47 |
| | QKLD-Rand | 128 | 13 | 1.177 | 0.40 |
| | QKLD-QInit | 123 | 12 | 1.137 | 0.38 |
| Hybrid | Our | 100 | 10 | 1.833 | 0.23 |
| | KLD-Rand | 107 | 11 | 1.270 | 0.17 |
| | QKLD-Rand | 128 | 13 | 1.180 | 0.16 |
| | QKLD-QInit | 123 | 12 | 1.140 | 0.16 |

Table 5.4: Efficiency results for AOL query set with selective search (Topic & Hybrid partitioning) and exhaustive search (Random partitioning). $C_{Total}$ and $C_{Latency}$ are in million documents.

| Partitioning | Dataset | K | N | $C_{Total}$ | $C_{Latency}$ |
|---|---|---|---|---|---|
| Random | - | 100 | NA | 6.506 | 0.65 |
| Topic | Our | 100 | 10 | 2.345 | 0.92 |
| | KLD-Rand | 107 | 11 | 1.548 | 0.55 |
| | QKLD-Rand | 128 | 13 | 1.442 | 0.47 |
| | QKLD-QInit | 123 | 12 | 1.350 | 0.44 |
| Hybrid | Our | 100 | 10 | 2.348 | 0.30 |
| | KLD-Rand | 107 | 11 | 1.551 | 0.21 |
| | QKLD-Rand | 128 | 13 | 1.446 | 0.20 |
| | QKLD-QInit | 123 | 12 | 1.353 | 0.17 |

## 5.2 Efficiency

Another important concern we take into account is the efficiency of the partitioning methods. To evaluate this, we use $C_{Total}$ and $C_{Latency}$ metrics. $C_{Total}$ shows the total cost in terms of the number of postings processed for a query. This metric used in

earlier works [22] to compare Random partitioning with Topic partitioning. Later, it is noticed that latency of the query, $C_{Total}$, is also important to evaluate methods in terms of efficiency [24]. Indeed, latency directly effects the user experience and should be taken into consideration.

We report the efficiency results of our experiments for two different query sets namely TREC and AOL query sets in Table 5.3 and 5.4. In addition to these, we also report the efficiency results obtained with distributed CSI setup in Table 5.5. Note that, distributed CSI has no effect on $C_{Total}$ since the it is the sum of all postings processed and does not change. For this reason, we only report $C_{Latency}$ results in Table 5.5 for both TREC and AOL query sets.

Table 5.5: Efficiency results in terms of query latency for both TREC and AOL query sets with selective search (Topic & Hybrid partitioning) in distributed CSI setup. $C_{Latency}$ is in million documents.

| Partitioning | Dataset | K | N | $C_{Latency}$ | |
|---|---|---|---|---|---|
| | | | | TREC | AOL |
| Topic | Our | 100 | 10 | 0.65 | 0.86 |
| | KLD-Rand | 107 | 11 | 0.43 | 0.49 |
| | QKLD-Rand | 128 | 13 | 0.36 | 0.41 |
| | QKLD-QInit | 123 | 12 | 0.34 | 0.40 |
| Hybrid | Our | 100 | 10 | 0.18 | 0.23 |
| | KLD-Rand | 107 | 11 | 0.12 | 0.15 |
| | QKLD-Rand | 128 | 13 | 0.11 | 0.14 |
| | QKLD-QInit | 123 | 12 | 0.11 | 0.13 |

When we look at the results in Table 5.3, it is obvious that Topic partitioning has better total cost results than Random partitioning for all four datasets. The improvements vary from 60% to 75% depending on the dataset. Even if there are cluster-skipping nodes in Hybrid partitioning, their effect on cost is very low and similar gains are also valid for this method. However, in terms of the query latency, Topic partitioning has not clear improvements over Random partitioning. In fact, there are datasets where it works worse than Random partitioning. On the other hand, Hybrid partitioning provides significant savings over both Random and Topic partitioning up to 65% and 66%, respectively. Results in Table 5.4 for AOL query set with 100K queries also

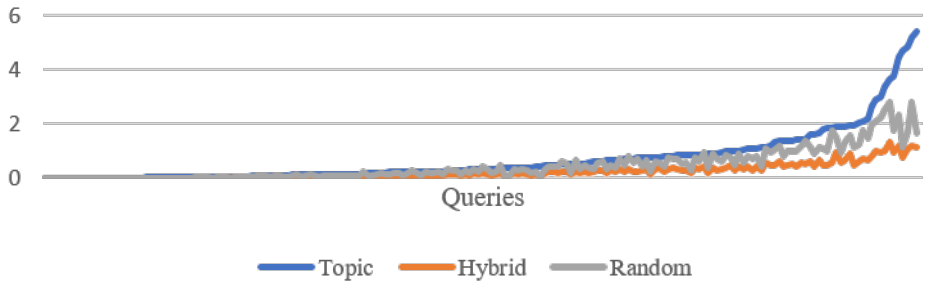
reveal the same trends for a much larger query volume. In distributed CSI setups, savings become even larger (i.e., %83 and 72%) since the cost of the CSI has more effect on Hybrid partitioning as it can be seen from Table 5.5.

We also studied the tail latencies of these setups. The main purpose here is that instead of looking at the averages, we also want to look at the 1% of the slowest queries to understand the efficiency results better. For this experiment, latencies for each query is determined for three methods and queries are resorted by the costs of Topic partitioning setup. Figures in 5.1 plotted with those values obtained for four different datasets. We can say from these figures that, Hybrid partitioning consistently produces lower latencies compared to other two methods for the tail queries. However, there is no clear winner between Random and Topic partitioning methods because as their performance vary for different queries.

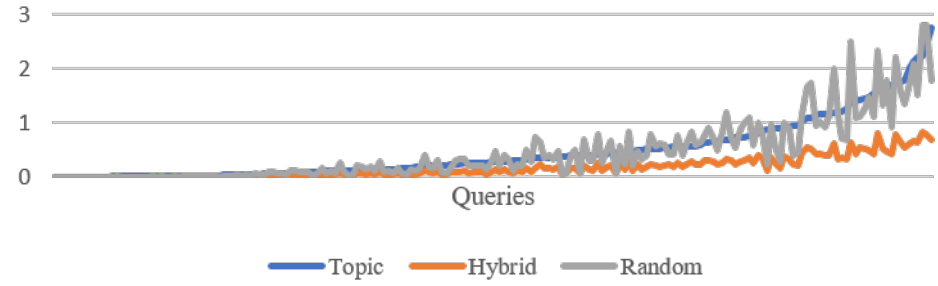
### 5.3 Load Balancing

We present the relative loads of physical nodes and plotted them in Figure 5.2. As it is stated earlier, we apply all partitioning methods to four different datasets and report the results for each in Figures 5.2(a) to 5.2(d). In addition to this, since the load distributions of Hybrid and Random partitioning methods are almost identical, we only plot the former one for better visuals.

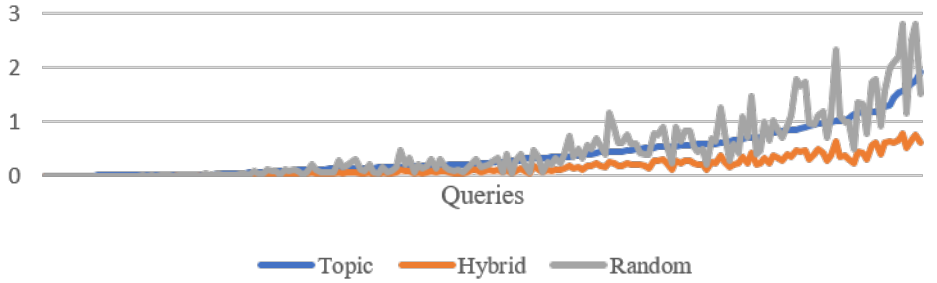
If we look at the results, Hybrid partitioning always yields balanced results for all four datasets with each shard has 10% load relatively and thus, shares the total load evenly. On the other hand, there is a load imbalance for Topic partitioning at various levels. Nevertheless, even for the best case obtained with $QKLD - QInit$ dataset (Figure 5.2(d)), there is a 10% load difference between the busiest shard and the most idle shard. This means that, due to load imbalance, some of the shards are waiting in the idle state while others are overloaded. This is a huge obstacle for optimizing resources which is one of the most important concerns in a low-resource environment and justifies the importance of our Hybrid partitioning approach.



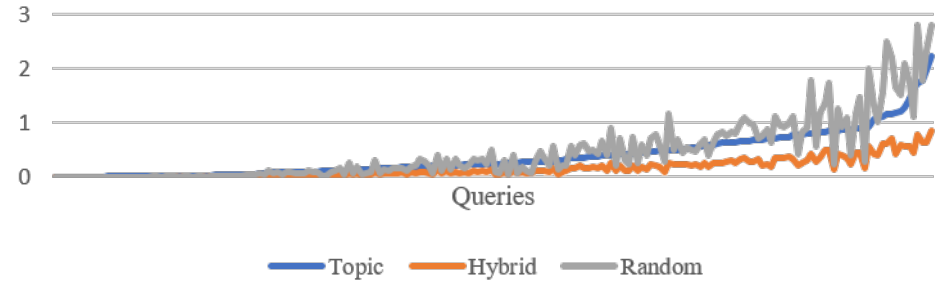

(a)




(b)

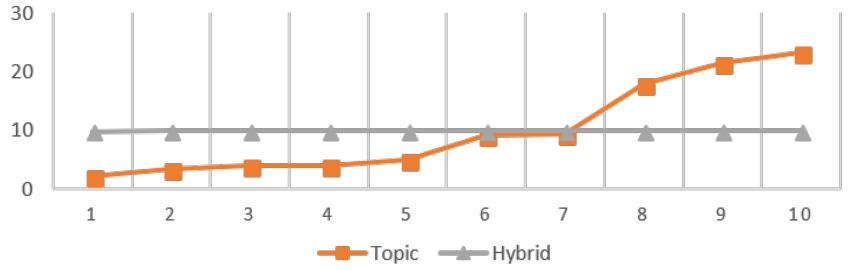



(c)

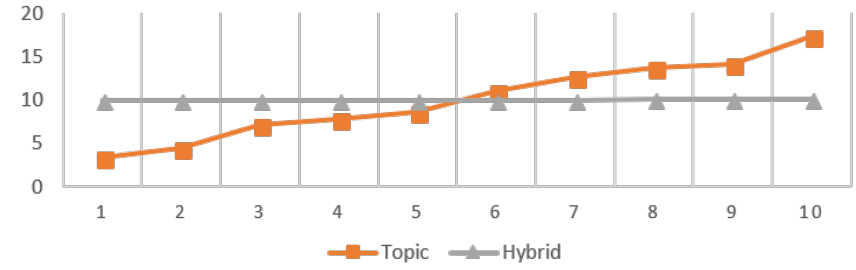



(d)

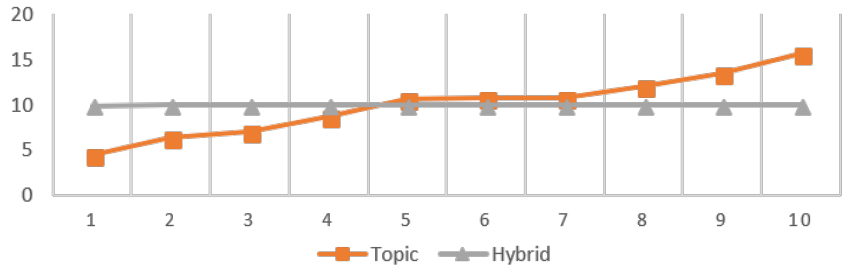
Figure 5.1: Tail analysis of 4 datasets in terms of number of postings processed on TREC query set: (a) OUR; (b) KLD-Rand; (c) QKLD-Rand; and, (d) QKLD-QInit.



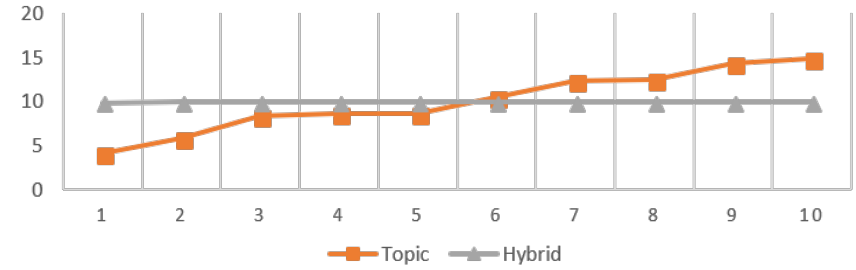

(a)




(b)




(c)




(d)

Figure 5.2: Load balancing analysis of 4 datasets in terms of relative load (%) of the nodes on TREC query set: (a) OUR; (b) KLD-Rand; (c) QKLD-Rand; and, (d) QKLD-QInit.

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

In this thesis, a new approach to distributed IR problem is presented, namely, selective search with Hybrid partitioning. We showed that traditional selective search with Topic partitioning may suffer from high query latency and load imbalance although it reduces the total query processing cost. To remedy these problems, our proposed approach distributes all documents to physical shards randomly but uses cluster-skipping indexes inside shards during query processing in order to make calculations only for the documents that are in the clusters relevant to the query.

In addition to Hybrid partitioning, we also presented a new approach for CSI processing, namely, selective search with distributed CSI. In this approach, documents inside the CSI are distributed to physical shards to reduce the effect of CSI processing cost in the query latency.

To evaluate our approaches and compare with others, we used two different query sets. The first one is TREC query set which consists of 200 queries from TREC 2009 to 2012 and second one is AOL query set with 100K queries collected from well-known AOL log. In addition, four different clustering strategies are used. Three of them are taken from [18]. We also created one by using K-Means with sampling. With varying query set-dataset combinations, we demonstrated the robustness of our results.

Our experimental evaluation showed that Hybrid partitioning method reduces the total cost like Topic partitioning while solving latency and load imbalance problems. For every experiment, Hybrid partitioning produces better query latency results in com-

parison to both Topic and Random partitioning. While doing this, it also achieves perfect load balance and thus, allows the optimized use of system resources. Moreover, we also showed that parallelizing CSI processing with distributed CSI enables further improvements in terms of query latency.

As a future work, separate indexes for each cluster can be constructed inside each shard instead of using skip indexes. This may help the small overhead caused by skipping nodes but also has two disadvantages. First one is, it will be hard to update index files. Secondly, more random accesses will be needed inside disk during query processing. We believe analyzing these trade-offs is an exciting future work direction. As another interesting research direction, instead of distributing CSI documents randomly to the physical shards, a CSI could be created for each shard and selected resources could be determined inside that shard. With this way, two-phase processing won't be needed.

# REFERENCES


[1] Ismail Sengor Altingovde, Engin Demir, Fazli Can, and Özgür Ulusoy. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Transactions on Information Systems (TOIS)*, 26(3):15, 2008.

[2] Ismail Sengor Altingovde, Engin Demir, Fazli Can, and Özgür Ulusoy. Site-based dynamic pruning for query processing in search engines. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 861–862. ACM, 2008.

[3] Ismail Sengor Altingovde, Özlem Nurcan Subakan, and Özgür Ulusoy. Cluster searching strategies for collaborative recommendation systems. *Information Processing & Management*, 49(3):688–697, 2013.

[4] Robin Aly, Djoerd Hiemstra, and Thomas Demeester. Taily: shard selection using the tail of score distributions. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 673–682. ACM, 2013.

[5] Jaime Arguello, Jamie Callan, and Fernando Diaz. Classification-based resource selection. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1277–1286. ACM, 2009.

[6] Jaime Arguello, Fernando Diaz, Jamie Callan, and Jean-Francois Crespo. Sources of evidence for vertical selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322. ACM, 2009.

[7] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. Web search for a planet: The google cluster architecture. *IEEE micro*, 23(2):22–28, 2003.

[8] Fidel Cacheda, Vassilis Plachouras, and Iadh Ounis. Performance analysis of

distributed architectures to index one terabyte of text. In *European Conference on Information Retrieval*, pages 394–408. Springer, 2004.

[9] Brendon Cahoon, Kathryn S McKinley, and Zhihong Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems (TOIS)*, 18(1):1–43, 2000.

[10] James P Callan, Zhihong Lu, and W Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28. ACM, 1995.

[11] B Barla Cambazoglu and Ricardo Baeza-Yates. Scalability challenges in web search engines. *Synthesis Lectures on Information Concept, Retrieval, and Services*, 7(6):1–138, 2015.

[12] Fazli Can, Ismail Sengör Altingövde, and Engin Demir. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems*, 29 (8):697–717, 2004.

[13] Charles L. A. Clarke, Nick Craswell, and Ian Soboroff. Overview of the TREC 2009 web track. In *Proceedings of The Eighteenth Text REtrieval Conference, TREC 2009, Gaithersburg, Maryland, USA*, 2009.

[14] Charles L. A. Clarke, Nick Craswell, Ian Soboroff, and Gordon V. Cormack. Overview of the TREC 2010 web track. In *Proceedings of The Nineteenth Text REtrieval Conference, TREC 2010, Gaithersburg, Maryland, USA*, 2010.

[15] Charles L. A. Clarke, Nick Craswell, Ian Soboroff, and Ellen M. Voorhees. Overview of the TREC 2011 web track. In *Proceedings of The Twentieth Text REtrieval Conference, TREC 2011, Gaithersburg, Maryland, USA*, 2011.

[16] Charles L. A. Clarke, Nick Craswell, and Ellen M. Voorhees. Overview of the TREC 2012 web track. In *Proceedings of The Twenty-First Text REtrieval Conference, TREC 2012, Gaithersburg, Maryland, USA*, 2012.

[17] Zhuyun Dai, Yubin Kim, and Jamie Callan. How random decisions affect selective distributed search. In *Proceedings of the 38th International ACM SIGIR*

*Conference on Research and Development in Information Retrieval*, pages 771–774. ACM, 2015.

[18] Zhuyun Dai, Chenyan Xiong, and Jamie Callan. Query-biased partitioning for selective search. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1119–1128. ACM, 2016.

[19] Fatih Hafizoglu, Emre Can Kucukoglu, and Ismail Sengor Altingovde. On the efficiency of selective search. In *European Conference on Information Retrieval*, pages 705–712. Springer, 2017.

[20] Yubin Kim, Jamie Callan, J Shane Culpepper, and Alistair Moffat. Does selective search benefit from wand optimization? In *European Conference on Information Retrieval*, pages 145–158. Springer, 2016.

[21] Yubin Kim, Jamie Callan, J Shane Culpepper, and Alistair Moffat. Load-balancing in distributed selective search. In *Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval*, pages 905–908. ACM, 2016.

[22] Anagha Kulkarni and Jamie Callan. Document allocation policies for selective searching of distributed indexes. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 449–458. ACM, 2010.

[23] Anagha Kulkarni and Jamie Callan. Topic-based index partitions for efficient and effective selective search. In *SIGIR 2010 Workshop on Large-Scale Distributed Information Retrieval*, pages 19–24, 2010.

[24] Anagha Kulkarni and Jamie Callan. Selective search: Efficient and effective search of large textual collections. *ACM Transactions on Information Systems (TOIS)*, 33(4):17, 2015.

[25] Xiaoyong Liu and W Bruce Croft. Cluster-based retrieval using language models. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193. ACM, 2004.

[26] Andrew MacFarlane, Julie A McCann, and Stephen E Robertson. Parallel search using partitioned inverted files. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 209–220. IEEE, 2000.

[27] Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri. Design of a parallel and distributed web search engine. In *Parallel Computing: Advances and Current Issues*, pages 197–204. World Scientific, 2002.

[28] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *InfoScale*, volume 152, page 1, 2006.

[29] Diego Puppin, Fabrizio Silvestri, and Domenico Laforenza. Query-driven document partitioning and collection selection. In *Proceedings of the 1st international conference on Scalable information systems*, page 34. ACM, 2006.

[30] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[31] Jangwon Seo and W Bruce Croft. Blog site search using resource selection. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1053–1062. ACM, 2008.

[32] Milad Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. In *European Conference on Information Retrieval*, pages 160–172. Springer, 2007.

[33] Milad Shokouhi, Luo Si, et al. Federated search. *Foundations and Trends® in Information Retrieval*, 5(1):1–102, 2011.

[34] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 298–305. ACM, 2003.

[35] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, 1979.

[36] Jinxi Xu and W Bruce Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 254–261. ACM, 1999.