Ranking and Retrieval of Information from Distributed Databases

Jay Mehta
Computer Science and Engineering
jmehta1@ucsc.edu

***Abstract*—This paper focuses on various information retrieval models and common problems faced during the retrieval of documents from heterogeneous distributed databases. The search process mainly focuses on optimizing for resources within the architecture and discusses several search algorithms and query scheduling techniques and compares them with the monolithic systems in-use. We then discuss the efficiency of these systems and how well can it support data-intensive applications in practice.**

I. Introduction

Today, efficient management of large text-based collections is a huge problem to deal with. Most of the search engines for the web and large corporate networks query on these collections. And therefore today, to support these large data-intensive applications and perform retrieval of information with as low latency as possible, architectures are adapting from their traditional single database model to Peer-to-Peer (P2P) based multi database model.

In these architectures, often the central node is referred to as a *broker* [1]. This is very similar to the master-slave architecture, though some differences persist and therefore this naming convention. The *broker* acts as the central node and receives the query from the user and distributes it based on certain attributes (which will be discussed further in the paper) to its subservient nodes. These nodes compile their top k-results based on the data collection they hold and return these as an intermediate result back to the broker node, which in turn performs a union operation to collate all these results and ranks them based on their relevance.

The process of Information Retrieval in a distributed environment is come to known as the *Distributed Information Retrieval* [2]. This area has its own set of problems to deal with and the major problems, that we'll also discuss in this paper revolves around the management of resources. This paper sheds light on some researches made in the past and discusses their efficiency in solving them. A unique decentralized approach is also observed in the scope of this paper and some other solutions that involved the concept of weights and thesauri in a distributed environment.

To ensure high throughput and minimum latency during the process of information retrieval, we also talk about different query scheduling methods that are a requirement these days, for sustaining large distributed networks. Hence scheduling queries, managing resources within the framework and fetching the results from the subservient nodes is the main focus of the paper.

II. Resoruce Management

Since long, many architectures have been using the unigram language model where, to index a piece of collection of data (text-based), the general schema was to have a count of the entire vocabulary and index the elements based on their frequency of occurrences in the document.

According to Zipf's law, which indicates that for large databases, the overall vocabulary is only a small fraction of the database size, as compared to small databases where the vocabulary is a very big fraction of the entire dataset. Though the unigram model may sound like a solution for large databases, it is not an efficient means of fetching data as compared to these retrieval models. Retrieval of information in its most modern form, includes ranking and efficient query processing, which is a lot more efficient as compared to the monolithic architectures.

The problem of resource management can be mainly divided into 3 sub-parts as follows [1].

A. *Resource Description*

This is a very crucial initial step that needs to be implemented to make sure the systems are aware about what each database contains in the distributed environment. The central broker node possesses a short profile, which basically describes certain attributes of a database or collection in each node. The attributes can vary depending on the schema followed. This is a way of representing data, and not related to acquiring it. Resource descriptions are usually stored on large broker servers where space constrain is not a major problem. [1]

Indexing is an important aspect of retrieval and needs to be addressed while allocating profile descriptions for each node. Indexing refers to a collection of terms with pointers to places where the required information about documents can be found [3]. An index term is a sematic which helps in remembering the main theme of the document. A common problem today regarding indexing is that due to the complex heterogenity nature of networks, indexers have to be take in account several formats of data including but not limited to, text, multimedia, JSON, etc. A very good in practice approach

is taken by ArangoDB where they have proposed a multidatabase model which has a core, one query language, but multiple data models. This is a hierarchial model that can be easily scaled horizontally and can support a large number of high-performance applications. The fact that documents can be indexed using a key/value pair, it possess the flexibility to support multiple formats of data like graphs, docuemnts, multimedia, etc. without the need to write domain specific constraints for each. [4]

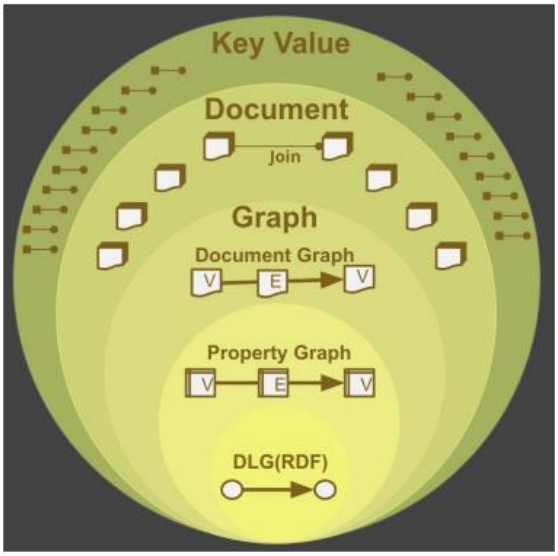

Fig. 1. ArangoDB Architetcure

As we talked earlier about the broker node maintaining a short description about each node, or in other words known as profiles, we can further expand that topic to refine these profiles which aims at characterizing an information resource not only by the content it offers but by the queries for which it provides relevant documents. In a P2P network, many systems use a collective discover approach where each peer stores information associated with other peers based on a specific query. This information is stored in routing tables together with the address of the peers that stores this information. Depending upon the schema and storage constraints, this approach either stores a certain number of keywords from the queries or the entirety of the query.

B. *Resource Selection*

Given an information need and the descriptors of the peers present in the network, a decision must be made for which peer to search to obtain the most relevant documents. Here a crucial step is ranking the documents based on their relevance to the document. Ranking is the process where a query is resolved using a statistical measure to compute the similarity of each document to the query, then retrieving the documents with the highest similarity.

One of the ranking methods observed is the cosine measure with logarithmic in-document frequency, which is one of the most effective measures to estimate relevance in a document. As stated by Owen k., the similarity *C(q,d)* of query *q* and document *d* in a collection of *N* documents is given by [5],

$$C(q,d) = \frac{\sum_{t \in q \cap d} (w_{q,t} \cdot w_{d,t})}{\sqrt{\left(\sum_{t \in q} w_{q,t}^2 \cdot \sum_{t \in d} w_{d,t}^2\right)}}$$

Fig. 2. Cosine Measure

where,

$$\begin{aligned} w_{d,t} &= \log(f_{d,t} + 1) \qquad \text{and} \\ w_{q,t} &= \log(f_{q,t} + 1) \cdot \log(N/f_t + 1)\,. \end{aligned}$$

Fig. 3. Cosine Measure (2)

Another way of ranking queries is illustrated by the Bayesian Network Model [2]. Here each resource R_i is reoresentated by a set of representation nodes r_j. An information need is representated by one or more queires (q), which are composed of query concepts (c_k) and query operators. The probability P(q| R_i) is calculated, which states the information need as requested by the query (q) is satisfied by the given/selected resource (R_i). The probability is calculated as below,

$$\begin{aligned} T &= \frac{df}{df + 50 + 150 \cdot cw/avg\_cw} \\ I &= \frac{\log\left(\frac{C+0.5}{cf}\right)}{\log\left(C + 1.0\right)} \\ p(r_k|c_i) &= b + (1-b) \cdot T \cdot I \end{aligned}$$

Fig. 4. Bayesian Network Model

where,
df is the number of documents in R_i containing r_k,
c_w is the is the number of indexing terms in resource R_i,
avg_{cw} is the average number of indexing terms in each resource,
C is the number of resources,
c_f is the number of resoruces conatiing r_k,
b is the minimum belief component.

The above model is also referred to as the CORI algorithm for ranking databases.

A lot of ranking algorithms which are in-practice are not publicly availble to study, however most of them use term-weighting, or vector space models, or a variation of those. One of the most widely used latvector space model is the Latent Semantic Indexing (*LSI*) [3], which is

responsible to nromalize the ranking problem by reducing the dimensionality of the documents.

Another way of selecting resources comes from the process of query refinement. This is a very elaborate process and we will go in detail in a later section. But in general the idea is that whenever the query is entered by the user, the query is refined in it's initial stages itself. The user could introduce problems like typos, or maybe not able to translate into words what exactly is to be retrieved. These are a part of huma errors that can be somehow made a lot less worse by query refinement. Another aspect of this is query scheduling and query expansion which will be discussed further.

C. *Resource Merging*

Once the descriptions are assigned, the broker node is aware about which nodes to communicate and retrieve the results from, the final question that arises here is that how the results are retrieved from these selected nodes. This is where the resource merging comes into picture. It is the process where all the retrieved ranked lists are merged into a single ranked list and based on which the broker may select the top *k-results*. Originally, systems assumed that all the data was evenly distributed within the architecture and most of the models or statistical means of ranking were similarity through the system to maintain uniformity and avoid complexity in any form. This was highly inefficient as databases started becoming multi-model and inclusive to heterogenous data formats. Not only were the statistical methods vague, but also the results fetched were not accurate.

Raw Score Merging (*RSM*) [6], is a technique which assigns a score to the docuents and based on those scores, the highest scoring document in a collection is retrieved. A very similar concept was proposed by Mazur Z. as well [7] where he used the concepts of weights and thesauri. This technique does have some prior assumptions such as collections are indexed by the same search model and the scores are of comparable nature in the network.

Using result Length to calculate Merging Score (*LMS*), is a technique which is developed on top of RSM. This technique combines the idea of weights and RSM and portrays a resulting margining methodology. The general idea is that it is capable of providing dynamic feedback to the system based on the current average score of the system and keeps updating the scores of nodes in real-time. It introduces the concept of negative weights and gives one to all those nodes whose scores are lower than the current average score of the system. At the end it is efficient in merging lists, based on the length of the lists retrieved. It does so by using the following formula,

$$s_i = \log\left(1 + \frac{l_i \cdot K}{|C| \sum_{j=1} l_j}\right)$$

Fig. 5. LMS score evaluator

where,

s_i is the score of the i^{th} collection,

K is a constant,

l_i is the number of documents retrieved in the i^{th} collection,

$|C|$ is the total collection count.

One of the most common problems faced during this step is that scores across the systems cannot be compared. To overcome this Jamie C. et al, have proposed to normalize the document scores, to an extent that it is almost equivalent to writing the scores in one homogenous collection. This does have some downsides such as it involves additional communication and computational costs when dealing with nodes which are distributed in a wide-area network. [7]

III. MANAGING QUERIES

As we talked about query refinement and expansion earlier, this is step usually taken as soon as the query is received by the database. This is taken care of in addition to the resource managing task, inorder for a server to answer queries in sub-second respond times and is an equally important improvement to make for efficient models. In this paper, we will talk about scheduling the queries and expansion of query profiles to explore its semantics in a more statistical manner.

A. *Scheduling Queries*

When queries start arriving at a very high rate and the system needs to manage the queries in real-time, several scheduling techniques can be implemented to make sure firstly that every query is responded as early as possible, each waiting query is scheduled to a node in a way that the overall load-balance structure is maintained in the system, and most importantly, if throughput is very high, it does not slow down the retrieval process or affect the system in any way.

The general approach to deal with this is replicating nodes within a specific collection. The concept of shards and replication was very well stated by Ana F. et al. [8]. It introduced the idea of a hybrid scheduling for queries, for a system to handle both low and high overheads,

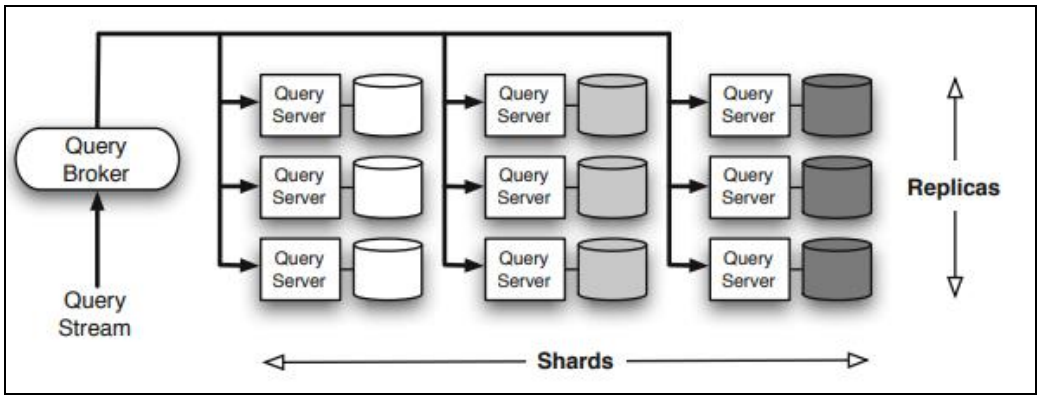

Fig. 6. Query Scheduling Using Shards

depending upon the need. This falls in line with dynamic management of queries and is highly efficient. The concept of *shard* is introduced. A *shard* is a cluster of nodes working togather in a particular collection. The idea is to maintain a lot of replicas of a specific node inside a shard, to enhance the workflow. The architecture is displayed in figure 6 for reference.

The proposed workflow is as follows, when the query is entered by the user, the proposed hybrid scheduling architecture uses certain *query efficiecny prediction* algorithms for the broker to allocate a node in a *shard* to each query. The algorithm chooses such a node where the wait time for the query is minimum and the other above mentioned constrains are satisfied. This architecture is proposed to be hybrid as for heavy load, it uses these efficiency prediction algorithms combined with scheduling, and retains simplicity and speed under low load.

Algorithms surveyed for prediction are as follows.

- Round Robin (*RR*) – This is a simple modulo function that acts by allocating shards based on the total number of available shards and modulo by total number of replicas. In case of collision, this simply allocates the very next available replica. The downside for this prediction is that it can keep assigning multiple queries to a specific node while other nodes in the network remain unoccupied.
- Queue Length (*QL*) – This does not make predictions as such but rather simply allocates a resource available with the minimum number of queries already allocated to it. Clearly this would not be able to handle complexities of queries and waiting time would vary for each retrieval.
- Least Loaded (*LL*) – This is an adaptation over the previous two and uses the sum of all queued queries for a replica, and then assign a node based on the queue length.

Another common problem observed while scheduling queries is that often queries with very large response time are dropped, so that a system can support multiple queries at once and also that one particular resource is not over-worked. This might sound like a practical solution for maintaining large data-intensive applications or frameworks, but it does take a toll when the user gets dissatisfied with the output. A creative solution to this was proposed by Daniele B. et al., [13]

The idea proposed here is about query pruning, and the concept is introduced in a dynamic nature. A *term-at-a-time* (*TAAT*) strategy is discussed here, which aims at eliminating unwanted collections which have lists of ranking with a low score. Therefore, only the top relevant lists are retrieved. The only downside discussed by the author for this was that *TAAT* can sometimes prune relevant documents as their context may not directly align with the strategy, though *TAAT* on average increases overall efficiency.

B. *Query Exapnsion*

Lately, user-queries are mostly processed using indexes and ontologies which are direct matches to the query keywords and though this may be efficient for applications with low load, it is very laborious for large-scale networks and costly. A common problem occurring because of this is that some collections might have a synonym of the keyword present in the query and with the same intuition yet may not be selected due to ‘term-mismatch’. Query expansion is the process where the semantics of the queries are extended such that it makes the retrieval process more efficient. It involves several steps such as relevance feedback, interactive query filtration, corpus dependent knowledge models, corpus independent knowledge models, search result clustering, and word sense disambiguation, etc. [9]

Some observed query expansion techniques studied are as follows, [1]

- For the case of Yahoo!, the engine retrieves only the top 10 results. The search engine returns snippets as a summary of the result page, which are

the "passages" from which expansion terms are extracted.

- Local peer feedback – In this, each peer gives a feedback/score to the profiles of its neighbouring peers in the collection. This allows a lot of pruning beforehand and saves time to rank relevant documents only.
- Global pseudo feedback - the intuition here is that the broker node keeps a basic profile of the entire structure, just like a centralized system would return.

IV. TESTING

In this section we look at the performance analysis of the techniques mentioned in the previous sections and how well they contribute to improving the overall efficiency of the system.

First, we analyse the CORI algorithm that is a database ranking algorithm mentioned in Section II of the paper. This algorithm aimed at ranking resources for the databases so that they could be efficiently selected.

The CORI algorithm was initially tested by Jamie C. [2], under several series of experiments on different testbeds ranging from *O*(100) to *O*(1000) databases, which was developed by several sources. The experiments yielded the results which are displayed in Figure 7.

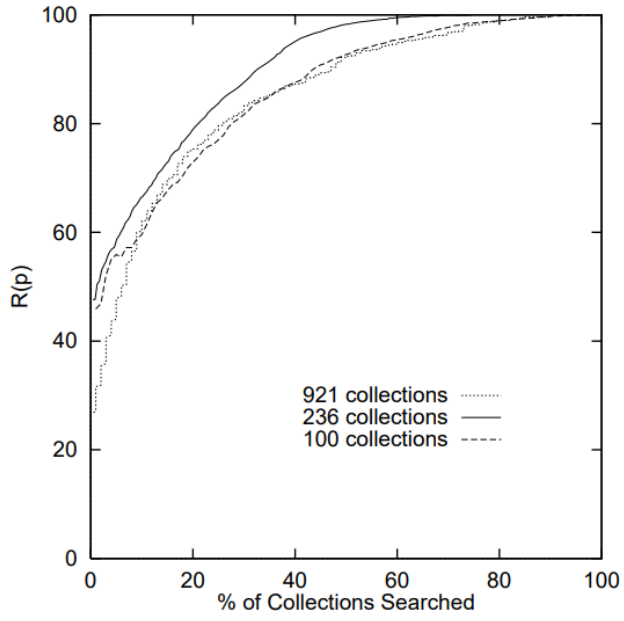

Fig. 7. Testing CORI with different resoruces

In Graph (Figure 7), the X-axis represents the percentage of the database searched in the testbed mentioned above, while the Y-axis represents the relevance of the documents retrieved at that iteration of the testbed. From the above results we can see that, algorithm retains efficiency when the dataset size is varied. The efficiency marginally reduces for small datasets as compared to large datasets, yet the difference is not very much. These were extensive searches made on the TREC testbeds.

Following this, authods Yves R. et al., performed several experiments on the resource merging/selection algorithms and compared it with CORI. Lets look at the perofrmances of RSM and LSM.

A brief overvew of the results is shown in Fig. 8. A little about the databases used, the 3 algorithms are compared on two sets of testbeds here, the TREC8 and TREC9 namely. The TREC8 testbed conatins around 528,155 documents with fields ranging from Finance, Federal Register, Foreign Broadcast Service and News from the LA Times. The size of this dataset is 1,904 MB. On the other hand, TREC9 is a relatively larger database with 1,692,096 documents form the web, using a memory of 11,033 MB. It is roughly 6 times greater than TREC8 and therefore tests the algorithms vigoroulsy.

From the given Fig 8, the three merging and selecting algorithms are compared (namely RSM, CORI, LMS) and their precisions are written down with a difference from the centralized approach (the column labelled Single). The row of 'TREC8-OPT' and 'TREC9-OPT' can be viewed as the optimal selection procedure. And from the table the author infers that the the RSM algorithm performs worse than either the CORI or LMS. And further the LMS retrieves atleast as precise queries as the CORI, or does better.

Next we consider comparing the three query prediction algorithms that are stated above in Section III. Namely, LL, RR, QL. In Fig 9, the Average Waiting times (AWT) and the Average Completion tijmes (ACT) of the three algorithms are compared with a prelimary of window size of 8 minutes. These three algorithms are compared for different configurations of shards and replicas. In addition to the elemental comparison, each hybrid combo is also compared namely RR/LL and QL/LL both. The least time taken is highlighted and that also indicates the best hybrid architecture for that specific configuration.

Upon spectating further, it is evident that the best times are achieved by the LL technique over the other two, though for the configuration of 5 shards nad 15 or more replicas, with low contention, RR and QL have very low waiting times. And upon observing the hybrid architectures, we can confirm the fact the once the load increases, the behaviour of the system closes to the LL technique and for low

Merging / Selection	Single (baseline) Av. Prec.	RSM Av. Prec.	RSM Diff.	CORI Av. Prec.	CORI Diff	LMS Av. Prec.	LMS Diff
TREC8-NS	0.2566	0.2397	-6.59 %	0.2416	-5.85 %	0.2462	-4.05 %
TREC8-CORI	0.2566	0.2005	-21.86 %	0.2033	-20.77 %	0.1997	-22.17 %
TREC8-TRD-CS	0.2566	0.2440	-4.91 %	0.2453	-4.40 %	0.2462	-4.05 %
TREC8-OPT	0.2566	0.2543	-0.90 %	0.2533	-1.29 %	0.2480	-3.35 %
TREC9-NS	0.1986	0.1832	-7.75 %	0.1847	-7.00 %	0.1932	-2.72 %
TREC9-CORI	0.1986	0.1862	-6.24 %	0.1893	-4.68 %	0.1922	-3.22 %
TREC9-TRD-CS	0.1986	0.1828	-7.96 %	0.1867	-5.99 %	0.1944	-2.11 %
TREC9-OPT	0.1986	0.2097	5.59 %	0.2142	7.85 %	0.2144	7.96 %

Table 3. Average precision achieved by various selection and merging strategies

Fig. 8. Comparison of CORI vs LMS vs RSM

	ACT					AWT				
# Rep	RR	QL	LL	RR/LL	QL/LL	RR	QL	LL	RR/LL	QL/LL
	2 Shards									
5	-	-	**908**	**908**	**908**	-	-	**388**	**388**	**388**
10	683	650	**550**	555	554	162	130	**30**	34	34
15	564	559	**535**	542	541	44	39	**14**	21	21
20	537	536	**534**	**534**	**534**	16	15	14	14	**13**
	5 Shards									
5	716	636	**417**	**417**	**417**	346	303	**47**	**47**	**47**
10	424	421	**383**	384	392	54	51	**14**	**14**	22
15	380	**379**	383	379	**379**	10	10	14	**9**	**9**
20	**372**	**372**	537	**372**	**372**	3	**2**	14	3	**2**

Fig. 9. Comparison of waiting times and completion times of queries

loads, it maintains simplicity and the speed of the basic algorithms QL and RR.

V. Conclusion and Future Scope

In the end, there is a lot of vigorous testing that still has to be done on these testbeds (and hopefully even larger testbeds). The domain of Distributed Information Retrieval (*DIR*) has only scratched its surface yet. In this paper, we've tried to cover the major topics in *DIR* and potential works/researches that have been conducted over the years. The paper also tries to summarize potential solutions have been adapted over the period of time and explore similar solutions through different approaches.

We studied about resoruce management and several techniques that come to play when we want to manage these resoruces within a distributed environment, then we talked a pre-processing phase for scheduling queries that only make the system more efficient and finally laid some insights on the findings about all the algorithms/architectures discussed in the paper.

This does lay a general path for a lot of future scope to be followed. Further, the study would want to cover some decentralized approach and how well they fare with the mostly centralized architectures that we covered here. Moreover the concept of ranking would be improved using several machine learning algorithms that can also improve upon the query efficiency prediction algorithms. And lastly, a cloud-based infrastrcuture would want to be covered and how well the factor of latency can be improved over the cloud.

VI. REFERENCES

[1] H. F. Witschel, "Ranking Information Resources in Peer-to-Peer Text Retrieval: An Experimental Study," in *Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*, Association for Computing Machinery, 2008.

[2] J. Callan, "Distributed Information Retrieval," in *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval*, 2000.

[3] M. a. T. K. Kobayashi, "Infomration Retreival on the Web," vol. 32, 2000.

[4] "ArangoDB, the database for graph and beyond," [Online]. Available: https://www.arangodb.com/wp-content/uploads/2020/03/ArangoDB-White-Paper_What-is-a-multi-model-database-and-why-use-it.pdf?hsCtaTracking=964a2732-53d1-477e-93ed-0e7430c8d1bf%7C7ff1d46f-2bc6-439e-8e69-98b650993860.

[5] O. a. M. A. a. S. T. a. Z. J. de Kretser, "Methodologies for distributed information retrieval," in *Proceedings. 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183)*, 1998, pp. 66-73.

[6] Y. a. A. F. a. S. J. Rasolofo, "Approaches to Collection Selection and Results Merging for Distributed Information Retrieval," Association for Computing Machinery, 2001.

[7] Z. Mazur, "Models of a distributed information retrieval system based on thesauri with weights," *Information Processing & Management*, 1994.

[8] J. P. a. L. Z. a. C. W. B. Callan, "Searching Distributed Collections with Inference Networks," 1995.

[9] A. Freire, C. Macdonald and N. Tonellotto, "Hybrid Query Scheduling for a Replicated Search Engine," Berlin, 2013.

[10] D. Broccolo, "Load-Sensitive Selective Pruning for Distributed Search," San Francisco.

[11] H. K. Azad, "Query Expansion Techniques for Information Retrieval: a Survey," *CoRR*, vol. abs/1708.00247, 2017.

[12] J. P. a. L. Z. a. C. W. B. Callan, "Searching Distributed Collections with Inference Networks," 1995.

[13] D. a. M. C. a. O. S. a. O. I. a. P. R. a. S. F. a. T. N. Broccolo, "Load-Sensitive Selective Pruning for Distributed Search," New York, 2013.

[14] F. a. C. V. a. P. V. a. O. I. Cacheda, "Performance Analysis of Distributed Information Retrieval Architectures Using an Improved Network Simulation Model," 2007.

[15] P. a. L. T. a. R. M. a. R. E. Felber, "Managing Collaborative Feedback Information for Distributed Retrieval," New York, 2008.

[16] P. a. C. J. Ogilvie, "The Effectiveness of Query Expansion for Distributed Information Retrieval," New York.