The Quality of Probabilistic Search in Unstructured Distributed Information Retrieval Systems

Ruoxun Fu

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

Department of Computer Science

University College London

2012

I, Ruoxun Fu, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Searching the web is critical to the Web's success. However, the frequency of searches together with the size of the index prohibit a single computer being able to cope with the computational load. Consequently, a variety of distributed architectures have been proposed. Commercial search engines such as Google, usually use an architecture where the the index is distributed but centrally managed over a number of disjoint partitions. This centralized architecture has a high capital and operating cost that presents a significant barrier preventing any new competitor from entering the search market. The dominance of a few Web search giants brings concerns about the objectivity of search results and the privacy of the user. A promising solution to eliminate the high cost of entry is to conduct the search on a peer-to-peer (P2P) architecture. Peer-to-peer architectures offer a more geographically dispersed arrangement of machines that are not centrally managed. This has the benefit of not requiring an expensive centralized server facility. However, the lack of a centralized management can complicate the communication process. And the storage and computational capabilities of peers may be much less than for nodes in a commercial search engine. P2P architectures are commonly categorized into two broad classes, structured and unstructured. Structured architectures guarantee that the entire index is searched for a query, but suffer high communication cost during retrieval and maintenance. In comparison, unstructured architectures do not guarantee the entire index is searched, but require less maintenance cost and are more robust to attacks.

In this thesis we study the quality of the probabilistic search in an unstructured distributed network since such a network has potential for developing a low cost and robust large scale information retrieval system. Search in an unstructured distributed network is a challenge, since a single machine normally can only store a subset of documents, and a query is only sent to a subset of machines, due to limitations on computational and communication resources. Thus, IR systems built on such network do not guarantee that a query finds the required documents in the collection, and the search has to be probabilistic and non-deterministic. The search quality is measured by a new metric called accuracy, defined as the fraction of documents retrieved by a constrained, probabilistic search compared with those that would have been retrieved by an exhaustive search.

We propose a mathematical framework for modeling search in an unstructured distributed network, and present a non-deterministic distributed search architecture called Probably Approximately Correct

(PAC) search, We provide formulas to estimate the search quality based on different system parameters, and show that PAC can achieve good performance when using the same amount of resources of a centrally managed deterministic distributed information retrieval system.

We also study the effects of node selection in a centralized PAC architecture. We theoretically and empirically analyze the search performance across query iterations, and show that the search accuracy can be improved by caching good performing nodes in a centralized PAC architecture. Experiments on a real document collection and query log support our analysis.

We then investigate the effects of different document replication policies in a PAC IR system. We show that the traditional square-root replication policy is not optimum for maximizing accuracy, and give an optimality criterion for accuracy. A non-uniform distribution of documents improves the retrieval performance of popular documents at the expense of less popular documents. To compensate for this, we propose a hybrid replication policy consisting of a combination of uniform and non-uniform distributions. Theoretical and experimental results show that such an arrangement significantly improves the accuracy of less popular documents at the expense of only a small degradation in accuracy averaged over all queries.

We finally explore the effects of query caching in the PAC architecture. We empirically analyze the search performance of queries being issued from a query log, and show that the search accuracy can be improved by caching the top-k documents on each node. Simulations on a real document collection and query log support our analysis.

Acknowledgements

I first give my sincere thanks to Prof. Ingemar Cox, who supervised my PhD work. Sir, I have learnt a lot from you in the past four years, and will always remember your careful and earnest attitude to the research work. Dr. Jun Wang, who acted as my second supervisor, had many useful discussions with me on a variety of topics, and they were very inspiring. I thank you for all your help.

Thanks to Dr. Dell Zhang and Dr. Miguel Rio, who examined this PhD work. The discussion in the final viva was inspiring, and pointed out the new possibilities of the future work.

Thanks to all the people who have been members of the Media Futures group. The PhD life can get boring and frustrating but you guys made it fun.

Thanks to all staffs in the Computer Science department for your great help to deal with those administrative hassles so I can focus on my research work.

Doing a PhD has been one of my ambitions and the last four year have been really important to me. For those who helped me in one way or another, I wish to send out my thanks to each of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web has gone through a dramatic growth in the last two decades, and has transformed the social lives of individuals, and the operations of businesses and governments, in both developed and developing countries. Searching the web is critical to these successes. Search is now common - Americans alone are estimated to have performed over 13 billion searches in February 2009 [Sea09]. And the size of the indexed web is now estimated to be about 65 billion webpages, of which Google is estimated to index over 17 billion pages [web09].

The frequency of searches together with the size of the index prohibit a single computer being able to cope with the computational load. Consequently, a variety of distributed architectures have been proposed. Commercial search engines such as Google, usually use an architecture where the the index is distributed (and arguably "virtually centralized") over a number of disjoint partitions [BDH03b]. And within each partition, the partial index is replicated across a number of machines. A query must be sent to one machine in each partition and their partial responses are then consolidated before being returned to the user. The number of partitions and the number of machines per partition is a careful balance between throughput and latency [RHHR09b]. Changes to the collection or to the query distribution may necessitate that the index be repartitioned, a process than can be quite complex and time consuming [RHHR09b]. Note that while the index is distributed across machines, the machines themselves are typically housed within a central server facility. This centralized architecture has a high capital and operating cost that presents a significant barrier preventing any new competitor from entering the search market. It is reported that both Microsoft and Google spend more than $1 billion per year to support their search facilities. It is also reported that Google alone centrally manage over 80,000 servers in its data centers [goo10, goo06, Dea09, RHHR09a]. The dominance of Google and Microsoft's Bing reduces people's choice of search service providers, and many experts are concerned that this dominance may be abused by these Web search giants. Other concerns include the objectivity of search results, and the privacy of the user. Commercial search engines are more likely to adjust the results based on their commercial purpose, e.g. a less relevant sponsored link may be placed in a higher ranking position than

those non-sponsored links. A more competitive market might be needed to avoid abuses due to dominant market share.

A promising solution to eliminate the high cost of entry is to conduct the search on a less centrally managed distributed architecture. Many decentralized distributed architectures have also been proposed for web search, and are often referred to as peer-to-peer (P2P) architectures. Peer-to-peer architectures offer a more geographically dispersed arrangement of machines that are not centrally managed. This has the benefit of not requiring an expensive centralized server facility. Moreover, peer-to-peer networks could be used to share and search contents that are not published by online web sites, e.g. a user can distribute the information stored on his/her local computer into a peer-to-peer network easily. However, the lack of a centralized management can complicate the communication process. And the storage and computational capabilities of peers may be much less than for nodes in a commercial search engine. Li *et al.* [LLH+03] provide an overview of the feasibility of peer-to-peer web indexing and search. Their analysis assumes a deterministic system in which, if necessary, a query is sent to all peers in the network, for example. The authors do comment on the possibility of "compromising result quality" by streaming the results to the users based on incremental intersection. However such a "compromise" is quite different from the non-deterministic search proposed here.

P2P architectures are commonly categorized into two broad classes, structured and unstructured. Structured architectures usually rely on the distributed hash table (DHT) and partition the index disjointly based on terms, i.e. particular terms are indexed by particular nodes in the network. A query is only sent to those nodes responsible for the terms in the query. Ideally, such architectures also guarantee that the entire index is searched. However, such an architecture suffers two problems: 1) the transmission of term posting lists could impose high communication cost to the network, 2) maintenance of the distributed index can result in high communication overhead due to the dynamic nature, e.g. entry and exit, of peers in the network, and thereby the quality of the search could be degraded. In comparison, unstructured architectures do not disjointly partition the index and a query is usually broadcast to potentially all the nodes in the network in order to guarantee that all documents are compared. While the maintenance cost is significantly reduced, queries can generate high communication cost. In addition, unstructured architectures are usually more robust to adversarial attacks than their structured counterparts. In unstructured architectures, data is replicated onto different machines, and the failure of one machine seldom removes data from the whole network. Note P2P architectures can also be deployed in a central managed environment, e.g. data centers. In that case, since they need very little central management, their maintenance cost could be much lower than those centralized competitors.

This thesis particularly looks at search in an unstructured distributed network since such a network has potential for developing a low cost and robust large scale information retrieval system that encourages diversified information sharing and search. Note we focus on the search quality other than latency

or communication cost issues throughout this thesis, i.e. we do not compete the response time with other major search architectures. To achieve our goals, we propose a mathematical framework for modeling search in an unstructured distributed network, and present a non-deterministic information retrieval system based on such a network. A non-deterministic information retrieval system is defined to be one in which (a) the set of unique documents indexed may be selected randomly and/or (b) the response to a query may be a function of a random process. We assume that the non-deterministic system consists of a set of independent computers. By "independent computers" we mean that computers that do not communicate between one another for the purposes of either building the index, or responding to a query. The absence of communication/coordination between computers prevents the non-deterministic IR system from overloading the communication infrastructure, and provides a system architecture that is very scalable and reconfigurable.

1.1 Contributions and Structure

This thesis mathematically studies the quality of the probabilistic search in an unstructured distributed network. The search quality is measured by a new metric called accuracy, defined as the fraction of documents retrieved by a constrained, probabilistic search compared with those that would have been retrieved by an exhaustive search.

We propose a mathematical framework for modeling the full process of document acquisition and querying in an unstructured distributed network, and present a non-deterministic distributed search architecture called Probably Approximately Correct (PAC) search, We provide formulas to estimate the search quality based on different system parameters, and show that PAC can achieve good performance when using the same amount of resources of a centrally managed deterministic distributed information retrieval system.

We also study the effects of node selection in a centralized PAC architecture. We theoretically and empirically analyze the search performance across query iterations, and show that the search accuracy can be improved by caching good performing nodes in a centralized PAC architecture. Experiments on a real document collection and query log support our analysis.

We then investigate the effects of different document replication policies in a PAC IR system. We show that the traditional square-root replication policy is not optimum for maximizing accuracy, and give an optimality criterion for accuracy. A non-uniform distribution of documents improves the retrieval performance of popular documents at the expense of less popular documents. To compensate for this, we propose a hybrid replication policy consisting of a combination of uniform and non-uniform distributions. Theoretical and experimental results show that such an arrangement significantly improves the accuracy of less popular documents at the expense of only a small degradation in accuracy averaged over all queries.

Finally we explore the effects of query caching in PAC architecture. We empirically analyze the search performance of queries being issued from a query log, and show that the search accuracy can be improved by caching the top-k documents on each node. Simulations on a real document collection and query log support our analysis.

This thesis is organized as the structure below:

- **Chapter 1 - Introduction**
- **Chapter 2 - Background:** reviews important previous works in both information retrieval and distributed data sharing and search.
- **Chapter 3 - Probably Approximately Correct Search:** proposes a new distributed architecture for information retrieval, called *Probably Approximately Correct* (PAC) search [CFH09, CZFH10]. A full set of system parameters are defined in PAC, and mathematical formulas to estimate the search quality based on different system parameters are provided. PAC's cost and performance are also compareed with the traditional approaches, and it is shown that PAC can achieve good performance when using the same amount of resources of a traditional central-managed search system.
- **Chapter 4 - Adaptive Resource Selection**: examine the effects of resource selection approaches in PAC, and show the improvements achieved over the original PAC.
- **Chapter 5 - Document Replication Policies**: investigates the effects of different document replication policies in PAC . It is shown that the traditional square-root replication policy is not optimum for maximizing accuracy, and an optimality criterion for accuracy is derived. It is also shown that a non-uniform distribution of documents improves the retrieval performance of popular documents at the expense of less popular documents. To compensate for this, a hybrid replication policy consisting of a combination of uniform and non-uniform distributions is proposed. Theoretical and experimental results show that such an arrangement significantly improves the accuracy of less popular documents at the expense of only a small degradation in accuracy averaged over all queries.
- **Chapter 6 - Query Caching:** studys the effects of query caching in decentralized PAC architecture. The search performance across query iterations is theoretically and empirically analyzed, It is shown that the search accuracy can be improved by caching top-k documents on each node.
- **Chapter 7 - Conclusion:** concludes our work and proposes future research directions.

Chapter 2

Background

Information retrieval (IR) is a key research field in information science. However, its application was mostly implemented only in library service in early years. With the prevalence of web search, information retrieval has attracted more attention of the research community due to the close connection between web search and IR. The high query rate, together with an ocean of data on the web present a number of challenges to the researchers in IR. To cope with these challenges, the architectures of IR systems have evolved from single-server based to distributed multi-servers. In this chapter, we review the significant previous works in IR and web search. We first introduce the basic concepts in IR in Section 2.1, then go through the important architectures in distributed IR systems in Section 2.2. Next, we survey some widely known peer-to-peer networks, which are strongly connected to distributed IR, in Section 2.3. Finally, we focus on some important architectures that have been proposed in distributed IR field, in Section 2.4

2.1 Information Retrieval Basics

We first define some terminologies. *Documents* are the smallest units we build a retrieval system on. They might be text files of books, or chapters of a book, even individual memos, emails or web pages. A document comprises of a sequence of words. The size of a document may vary from a few words such as an email, to thousands of words such as in a book.

Collection (or called *corpus* sometimes) is the group of documents over which we perform retrieval.

An *information need* is the topic which a user is interested in. However, it is different from a *query*, which is a representation of an information need.

A document is *relevant* if it satisfies the user's information need. *Relevance* is subjective as a document may be judged as relevant by one person but irrelevant by another person. Especially the information need may be defined in terms of particular words like "computer performance", but usually the user would like to find relevant documents regardless of whether they precisely use those words or other words like "computer benchmark".

Using the definitions above, information retrieval (IR) is simply finding documents that satisfy an information need from within large collections (usually stored on computers) [MRS09]. Figure 2.1 shows the standard process in information retrieval.

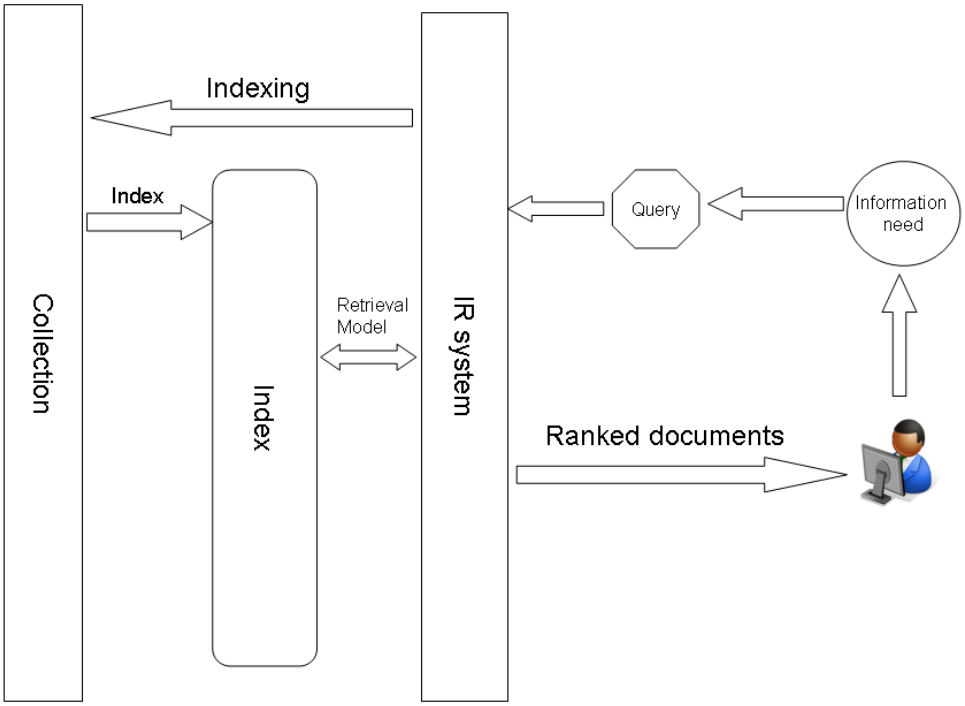

Figure 2.1: The standard process of IR system.

To compare different IR systems, we also need to assess the *effectiveness* of an IR system, which is usually measured by two key statistics about the system's returned results for a query.

Precision is the fraction of the returned results that are relevant to the information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} \tag{2.1}$$

Recall is the fraction of the relevant documents that are retrieved by the system.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|} \tag{2.2}$$

Of course there are some other measurements like query throughput, query latency and data storage size which measure the efficiency of the IR system. These measurements might be paid more attention in the engineering field of IR systems.

The simplest way to conduct search is to linearly scan each document in a row and retrieve documents that satisfy the query. Apparently, the complexity of this approach is proportional to the collection

	The fundamental knowledge of Computer	Modern Computing	...
computer	1	1	...
performance	0	1	...
CPU	1	0	...
...	...	...	...

Table 2.1: An example of incidence matrix. Matrix element (t,d) is 1 if the document in column d contains the term in row t, and is 0 otherwise.

size, hence, searching could be very slow when the collection size becomes large. To address this problem, another good way for search is to *index* all the documents in advance. We refer to the indexed units as *terms*. They are usually words but IR literature normally uses terms because some of them may not be thought of as words such as "A-4" or "George Bush". Thus, each document can be viewed as a "set of terms". The simplest indexing process is to record for each document whether it contains each term out of all the terms in the collection. Here we usually refer to all the terms in the collection as the *vocabulary* of the collection. The result is a binary term-document incidence matrix.

Consider the simplest Boolean Model, using the example in Table 2.1. If we search "computer" AND "performance", it is equivalent to a bitwise AND

$$11\ AND\ 01 = 01 \tag{2.3}$$

Then as a result the book "Modern Computing" will be returned.

The problem of this incidence matrix is its extremely large size. Assuming the collection has 1 million documents with a 500K vocabulary, the resulted $500,000 \times 10^6$ matrix would have half-a-trillion 0's and 1's. However, the matrix is extremely sparse. Suppose each document contains an average of 1000 terms, the matrix has roughly 1 billion 1's. Thus 99.8% of the cells are 0. So a much better representation is to record terms that do occur, i.e. only the 1 positions.

This idea is behind the *inverted index*. The basic idea of an inverted index is to keep a dictionary of terms. For each term, we have a list that records which documents the term occurs in. Each item in the list, which records that a term appears in a document, is called a *posting*, and the list of postings is called a *postings list*. And all the postings lists together are referred to as the *postings*. See Figure 2.2.

The major steps to build an inverted index are:

1. Collect the documents to be indexed;

2. Tokenize the text, turning each document into a list of tokens;

3. Process the tokens to produce normalized tokens, which are indexing terms;

4. Index the documents that each term occurs in, build an inverted index which consists of a vocabulary and postings.

Here *tokenization* is the task of chopping a character sequence in a document up into pieces, called

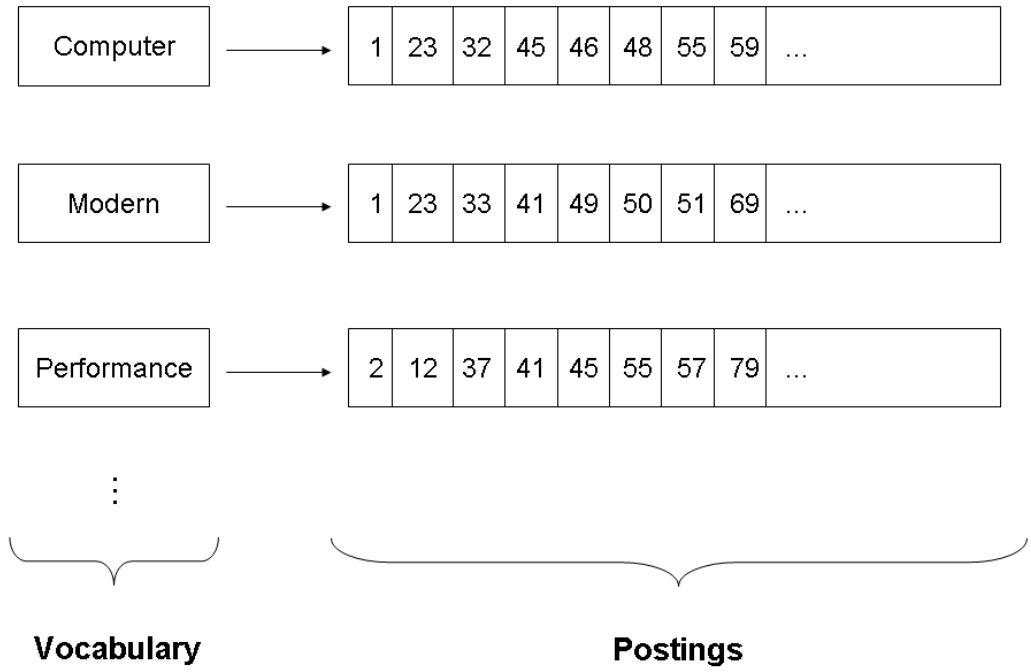

Figure 2.2: A example of the structure of the standard dictionary for a IR system.

tokens. A *token* is an instance of a sequence of characters in a document. A *type* is the class of the same tokens, e.g. a word can have multiple instances(tokens) in the collection but only one entry(type) in the vocabulary. A term is a type (usually normalized) that is included in the IR system's vocabulary. Therefore, a term j has *term frequency*(tf), t_{ij}, for a document i, which indicates how many times it occurs in that document. A term j also has *document frequency*(df), n_j, which indicates how many documents it occurs in. To discriminate between documents in a collection, we usually use *inverse document frequency*(idf), D_j, to scale a term j's weight for scoring. It is defined as:

$$D_j = \log \frac{N}{n_j} \tag{2.4}$$

where N is the size of the collection.

Note there may be some extremely common words like "and", "or" and "of", which would be useless in selecting documents matching a user information need. These words are called *stop words*. They are usually excluded from the index in the tokenization process.

Retrieval is to find the relevant documents that meet the information needs of the user. In practice, the relevance between documents and queries are usually calculated based on the information provided by inverted indices directly, without referring to original texts.

A simple idea is to use term frequency and inverse document frequency to assign a compound weight to each term in vocabulary. Thus every query and document can be viewed as a weighted vector of terms [SWY75]. We denote by $\overrightarrow{V}(d)$ the vector derived from document d; $\overrightarrow{V}(q)$ the vector derived from query q. We can score relevant documents for a query by measuring the cosine similarity of their vector representations.

$$Sim(q, d_i) = \frac{\overrightarrow{V}(q) \cdot \overrightarrow{V}(d_i)}{|\overrightarrow{V}(q)||\overrightarrow{V}(d_i)|} \tag{2.5}$$

The documents then can be ranked by their similarity scores in the result list.

People started to use probabilistic modeling approaches in IR from 1970s. The simple intuition behind this is that given a query and a document, what is the probability that they are relevant ($P(Relevance = True|Query, Document)$).

To rank the documents in probabilistic retrieval models, the *Probability Ranking Principle* (PRP) was proposed [Rob93]. According to the PRP, in a probabilistic model, in order to maximize the overall effectiveness of the retrieval, the optimal ranking order is to rank documents in decreasing order of their estimated probability of relevance to the query.

Okapi BM25 is one of the most widely known probabilistic models in IR [RWJ+96]. It was specifically designed for modern full-text collections by emphasizing the important statistic characteristics like term frequency and document length. The BM25 weighting scheme was developed to build a probabilistic model sensitive to these quantities without introducing too many parameters into the model. Given a query q, containing terms $q_1, ..., q_n$, the BM25 score of a document i is:

$$score(d_i, q) = \sum_{j \in q} D_j \cdot \frac{t_{ij} \cdot (k_1 + 1)}{t_{ij} + k_1 \cdot (1 - b + b \cdot \frac{l_i}{L})} \tag{2.6}$$

where l_i is the length of document i, and L is the average document length of the collection.

D_j is the *inverse document frequency* weight of the query term j. It is usually computed as:

$$D_j = \log \frac{N - n_j + 0.5}{n_j + 0.5}, \tag{2.7}$$

If the query is long, like a paragraph, we might also take into account weights of query terms. Then a document is scored as:

$$score(d_i, q) = \sum_{j \in q} D_j \cdot \frac{t_{ij} \cdot (k_1 + 1)}{t_{ij} + k_1 \cdot (1 - b + b \cdot \frac{l_i}{L})} \cdot \frac{(k_3 + 1)t_{qj}}{k_3 + t_{qj}} \tag{2.8}$$

where t_{qj} is the term j's frequency in query, and k_3 is another positive parameter that scales term frequency in query. k_1, k_3 and b are free parameters, which are usually set to $k_1, k_3 \in [1.2, 2.0]$ and $b = 0.75$, based on empirical results.

Another important probabilistic model for IR is the *language modeling* [PC98]. The intuition behind the language modeling approach is that a document is relevant to a query if the document's language model is likely to generate the query. Thus documents are ranked based on the probability of a document's language model M_i for document i generating the query q ($P(q|M_i)$).

The language modeling approach can be explained by the Bayes rule. Suppose our goal is to rank documents by $P(d_i|q)$, i.e. given a query, what is the probability that a document i is relevant to this query. Using the Bayes rule, we have:

$$P(d_i|q) = P(q|d_i)P(d_i)/P(q) \tag{2.9}$$

$P(q)$ is fixed for a specific query. The prior $P(d_i)$ is usually treated as uniform across all documents, so it can be ignored. However, in some cases we also can implement it with information like authority, number of times read and category. To simplify the problem, here we only rank documents by $P(q|d_i)$, which can be interpreted as the probability of generating query q from the document model derived from document i.

The most widely used document model is the multinomial unigram language model, which assumes there is no dependence among terms. Using maximum likelihood estimation (MLE), we have:

$$\hat{P}(q|M_i) = \prod_{j \in q} \frac{t_{ij}}{l_i} \tag{2.10}$$

At the first sight, the use of language models appears fundamentally different from previous models with tf-idf weighting schemes, since the unigram language model only explicitly encodes term frequency and there appears to be no use of inverse document frequency weighting in the model. However, there is an interesting connection between the language model approach and the heuristics used in the traditional models. This connection has much to do with the *smoothing*.

The classic problem in language modeling is the inaccurate estimation of term probabilities due to the sparsity of terms occurrences in one document. Smoothing are then proposed to solve the sparsity problem. Most smoothing methods utilize two distributions, a model $P_s(t|d)$ used for "seen" terms that occur in a document, and a model $P_u(t|d)$ for "unseen" terms that do not occur in a document. The probability of a query q can be written in terms of these models as following, where $c(t;d)$ denotes the

count of term t in d:

$$\begin{aligned}\log P(q|d) &= \sum_{t \in q} \log P(t|d) \\ &= \sum_{c(t;d)>0} \log P_s(t|d) + \sum_{c(t;d)=0} \log P_u(t|d) \\ &= \sum_{c(t;d)>0} \log \frac{P_s(t|d)}{P_u(t|d)} + \sum_{t \in q} \log P_u(t|d)\end{aligned}$$

The probability of an unseen term is typically taken as being proportional to the general frequency of the term, for example, as computed using the document collection. So, assume that $P_u(t|d) = \alpha_d P(t|C)$, where α_d is a document dependent constant and $p(t|C)$ is the collection language model. Now we have

$$\log P(q|d) = \sum_{t \in q; c(t;d)>0} \log \frac{P_s(t|d)}{\alpha_d P(t|C)} + l_q \log \alpha_d + \sum_{t \in q} \log P(t|C) \tag{2.11}$$

where l_q is the length of the query.

Note that the last term on the righthand side is independent of the document d, and thus can be ignored in ranking. Now we can see that the retrieval function can actually be decomposed into two parts. The first part involves a weight for each term common between the query and document (i.e., matched terms) and the second part only involves a document-independent constant that is related to how much probability mass will be allocated to unseen terms according to the particular smoothing method used. The weight of a matched term t can be identified as the logarithm of $\frac{P_s(t|d)}{\alpha_d P(t|C)}$, which is directly proportional to the document term frequency, but inversely proportional to the collection term frequency. The computation of this general retrieval formula can be carried out very efficiently, since it only involves a sum over the matched terms. Thus, the use of $p(t|C)$ as a reference smoothing distribution has turned out to play a role very similar to the well-known idf. The other component in the formula is just the product of a document-independent constant and the query length. We can think of it as playing the role of document length normalization, which is another important technique to improve performance in traditional models. Indeed, α_d should be closely related to the document length, since one would expect that a longer document needs less smoothing and as a consequence has a smaller α_d; thus, a long document incurs a greater penalty than a short one due to this parameter. Clearly, smoothing implicitly implements tf-idf weighting heuristics and document length normalization, and plays a key role in the language modeling approach.

Two popular smoothing methods are Jelinek-Mercer smoothing and Dirichlet smoothing [CG96, ZL04].

The Jelinek-Mercer method involves a linear interpolation of the maximum likelihood model with

the collection model, using a coefficient λ to control the influence of each:

$$P_\lambda(t|d) = (1-\lambda)P(t|d) + \lambda P(t|C) \tag{2.12}$$

The Dirichlet smoothing treats a language model as a multinomial distribution, for which the conjugate prior for Bayesian analysis is the Dirichlet distribution. Thus, the model is given as

$$P_\mu(t|d) = \frac{c(t;d) + \mu P(t|C)}{\sum_{t' \in V} c(t';d) + \mu} \tag{2.13}$$

Another important field of IR is system evaluation, which studies how to reliably judge the quality of a IR system. The $precision$ and $recall$ introduced above are commonly used for unranked retrieval results. Another metric, $Fmeasure$, calculates the harmonic mean of precision and recall.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R})} = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 \cdot P + R} \tag{2.14}$$

where P is precision and R means recall; $\alpha \in [0,1]$ and $\beta \in [0,\infty]$.

The default *balanced F measure* equally weights precision and recall, which makes $\alpha = 0.5$ or $\beta = 1$. It is commonly written as F_1:

$$F_1 = \frac{2PR}{P+R} \tag{2.15}$$

The harmonic mean prevents a system from simply returning all documents for a good evaluation score. It is always less than or equal to arithmetic mean or geometric mean, and is close to the minimum of two numbers when they differ greatly.

For ranked results, it is easy to draw a precision-recall curve along rank positions. Although examining the entire precision-recall curve is very informative, we often would like to transform it into a few numbers for comparing different IR systems. This desire leads to the appearance of the widely used measurement "*mean average precision*"(MAP), which provides a single-figure measure of quality across different recall levels. Average precision is the average of the precision values, each of which is obtained for the set of top k documents after a relevant document is retrieved at rank position k. Average precisions of different queries are then averaged to generate mean average precision [Cle93, YA06].

Suppose the set of all queries is Q; the set of relevant documents for a query $q_j \in Q$ is $\{d_1, d_2, ..., d_{m_j}\}$; R_{jk} is the set of top k ranked results for query q_j until the d_k ($d_k \in \{d_1, d_2, ..., d_{m_j}\}$) is obtained, then

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j} Precision(R_{jk}) \tag{2.16}$$

where $|Q|$ is the size of query set Q.

2.2 Centralized Distributed Information Retrieval

Large collections such as the scale of web may make the inverted index extremely large; thus, it is impossible to store the index on one machine. Although substantial research works have been done in the index compression, it is common to distributed the index across many machines. Furthermore, modern IR systems such as web search engines usually need to answer queries at a very high rate, a single machine could be easily overloaded by the heavy computation and communication burden, and this may cause unacceptable query latency. Consequently, a variety of distributed architectures have been proposed to cope with these challenges for IR systems. These distributed architectures can be roughly categorized into two broad classes: (i) centralized distributed architectures, usually used by commercial search engines such as Google and Yahoo!, and (ii) decentralized distributed architectures, often seen in a variety of peer-to-peer networks. We focus on the centralized distributed architecture in this section.

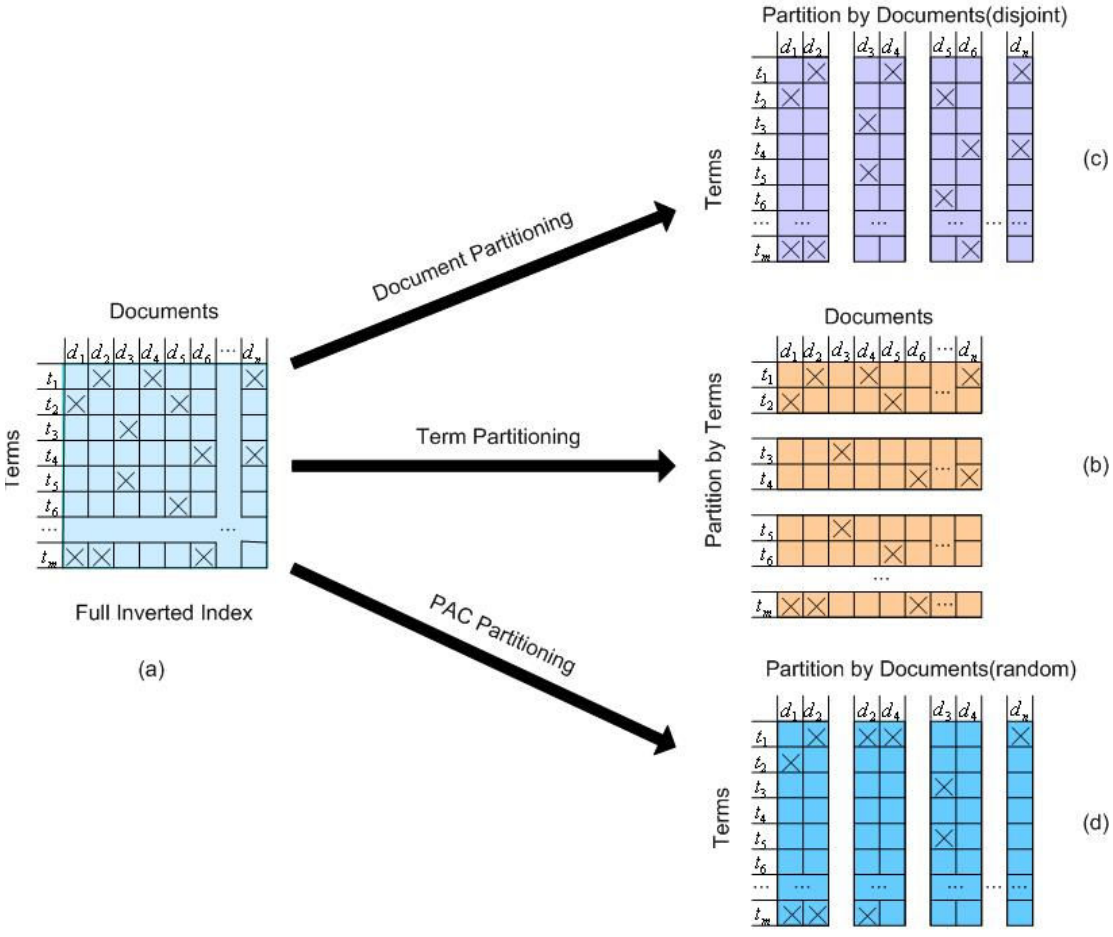

Figure 2.3: A comparison of different index partitioning schemes: (a) full index, (b) term-based partioning, (c) document-based partitioning, and (d) random partitioning

The centralized distributed architecture partitions the full index over many machines based on documents, as shown in Figure 2.3 (c). A response to a query requires each partition to be independently searched. Each partition contains a "randomly chosen subset of documents from the full index"

[BDH03a]. However, despite the documents may be chosen at random, each partition is *disjoint*. In addition, each partition is also replicated many times, in order to improve resilience to system failures, and to increase the query throughput. This architecture is referred to as distributed rendezvous [RHHR09a]. For example, it is reported that the well-known Google search engine partitions its index into around 1000 disjoint subsets, The search operations are supported by 40 data centers that are distributed globally, The replication level is about 1-3 in each center; hence, the global replication level can be approximated to be 80. Figure 2.4 shows the structure of a normal centralized distributed search engine.

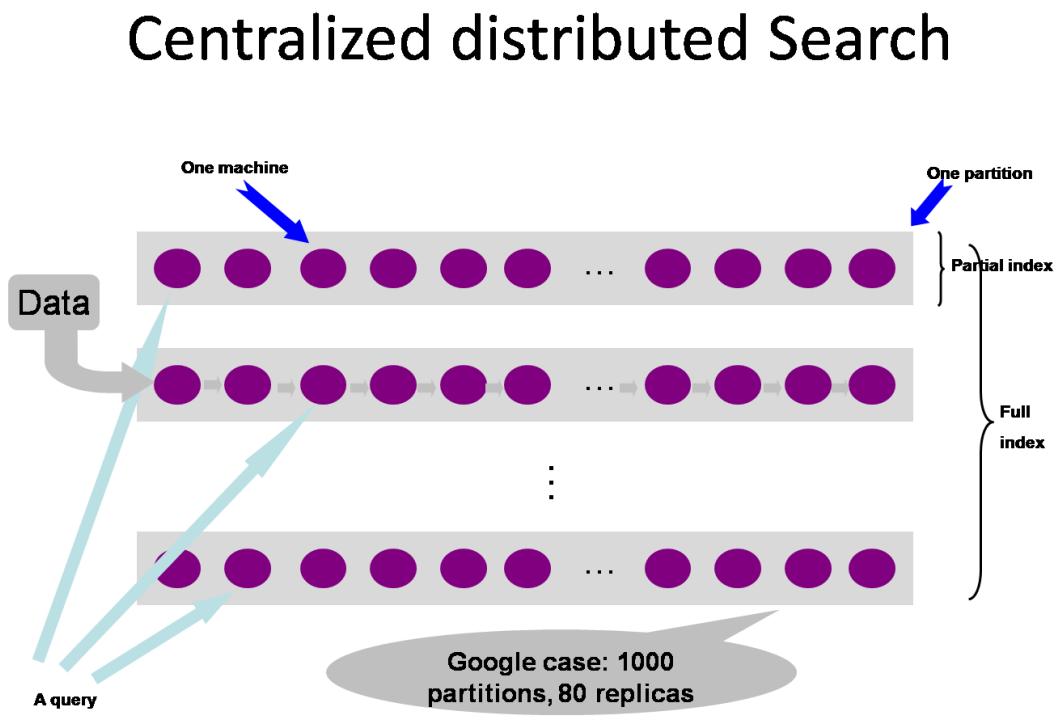

Figure 2.4: Centralized distributed architecture.

An advantage of such an architecture is the ability to tradeoff between the replication and partitioning, given a fixed number of machines. On one hand, when we increase the partitioning level, which reduces the replication level, the query completion time is reduced since more machines process the same query simultaneously. On the other hand, the reduced replication level decreases the number of queries that can be processed at the same time, i.e. decreases the query throughput. Thus, the query rate, which is the number of queries that can be processed in one second, is not guaranteed to be improved if we focus on improving only one of the two parameters. Consequently, it is crucial to find a best compromise between partitioning and replication. However, in the scenario of web search, where the document collection and query load may change continuously, this crucial work can become expensive since it has to be repeated periodically to cope with the new data and requirements. For example, Google usually

chooses periods when there are only a small number of users using the system [RHHR09a] to iteratively take offline only one data center at a time, move traffic away from that center using DNS load balancing, and then repartition that center. The whole process is time consuming, and usually terabytes of data need to be transferred between machines during the process.

2.3 Peer-to-Peer Networks

Peer-to-peer (P2P) network architectures may be the most inexpensive approach for various resource-intensive applications. Obviously, since P2P architectures normally distribute computation, communication and storage over a large number of individual machines (peers), the requirements for each single peer can be considerably reduced. Moreover, P2P architectures usually depend less on central management; hence, they potentially could decrease the maintenance cost that are usually high in those centralized architectures. It is commonly viewed that the development of three influential systems in 1999, namely the Napster music sharing system [SGG03], the Freenet anonymous data store [CSWH01] and the SETI@home volunteer based scientific computing projects [ACK+02], were the catalyst for modern peer-to-peer computing [RD10]. P2P architectures have attracted significant attention in recent years, after receiving substantial success in certain domains. The most successful applications of P2P architectures includes: (1) file sharing and distribution, such as Napster, eMule [KB+] and BitTorrent [GCX+05], (2) Media streaming, such as PPLive [VGLN06] and CoolStreaming [ZLLY05], (3) Telephony, such as Skype [BS04], and (4) Volunteer computing, such as SETI@home [ACK+02] and the BOINC platform [And04]. P2P web content distribution networks such as CoDeeN [WPP+04] and CoralCDN [FFM04], which were originally developed as research prototypes, also have attracted significant interest and gained a large user base in recent years. Users of these P2P web content distribution systems store a local copy of web content when accessing it, in order to build up a network of web caches. Thus, the loads of the servers hosting popular content are expected to be reduced by redirecting requests to those peers caching the same content. Many more P2P architectures have been proposed and prototyped, but may not be deployed publicly. Though the earliest and the strongest drive of P2P systems came from the file sharing applications, present P2P technologies are used in much more diverse fields, even including some commercial services. Based on the degree of centralization, the architecture of P2P systems can be commonly categorized into two main classes: (i) partly centralized P2P architecture, which has some form of central nodes that control the cooperations among different peers, (ii) decentralized P2P architecture, which has no central node at all. Depending on the organization of the network, the decentralized P2P architecture can be further categorized as: (a) decentralized structured P2P architecture, where peers connect to each other following certain rules and form a structured network, (b) decentralized unstructured P2P architecture, where peers connect to each other randomly.

2.3.1 Partly Centralized P2P

Partly centralized P2P architecture has some special central nodes that manage the participating nodes and facilitate the cooperations among different nodes. To reduce the burden of central nodes, usually only a small amount of data, like user profile, the shared file names and nodes' addresses, are stored on central nodes. Resource intensive operations such as file transmission do not need central nodes.

Napster [SGG03] used a central server to provide a Web site to maintain the membership and a content index. When a user logs in to the system, he/she needs to send an description of his/her shared files to the central server. Later, other users can send queries to the central server to search in all of shared files from different users. When a user find a desired file to download, he/she can obtain from the central server the IP addresses of those users who share the file, and directly download the file from them.

eDonkey/eMule has a similar architecture as Napster, while providing some new functions like segmenting the file and downloading from multiple users. Its utilization of resources is higher than Napster.

In comparison, BitTorrent [Coh03, GCX+05] reduces the functionality of the central server. It is mainly designed for transmitting a single large file. Each node first obtains a meta-data description of the file, then connects to a central server (called a tracker) and uploads the description. The central server acts as a broker which groups the users who request the same file together , and facilitates the cooperative downloading among users. BitTorrent focuses on full use of spare bandwidth of concurrent downloaders and has higher downloading speed than other competitors in most cases. However, it does not index file names, and thereby does not support search functionality.

Partly centralized P2P architectures can support fast growth of network scale, and provide abundant resources. However, its scalability and resilience to attacks/failures may not be as high as the decentralized P2P architecture, since the central node introduces a potential bottleneck and a single point of failure.

2.3.2 Decentralized Structured P2P

A decentralized structured P2P architecture does not have dedicated central nodes that operates critically in the network; thus, in principal be exempt from the scalability and single-point-of-failure problem existing in partly centralized P2P architecture. However, decentralized structured P2P architecture organizes the network based on certain rules and presents a specific order for the nodes in the network. This order must be maintained in the operation, and the maintenance is usually associated with an expensive cost.

The most widely known decentralized structured P2P is the *Distributed Hash Table* (DHT) [SMk+01, RFH+01, RD01, ZKJ01]. In a DHT, each node is allocated uniformly a unique identifier

in a numeric key space. The key space is then divided by the identifiers of the participating nodes to guarantee that each key is mapped to only one node through a mapping function. Thus, *Key Based Routing* (KBR) can be efficiently implemented such that given a starting node and a key, KBR produces a short path which is a sequence of a few nodes that ends at the node responsible for the key. Similar to the conventional hash table, the DHT also provides "put" and "get" interfaces to insert and retrieve key/value pairs into and out of a distributed system. When churn (nodes joining/leaving/failing) appears, the key/value pairs have to be moved between nodes to keep the node-to-key mapping correct. Depending on the implementation details, this maintenance communication cost can be very high, e.g. move the whole file between nodes. However, to minimize the communication cost, large data are usually not inserted as values into the DHT directly, instead, an indirection pointer which points to the node storing the data is inserted.

According to the node-to-key mapping function, there are several protocols of DHT proposed in previous works.

Chord [SMk$^+$01] maintains both keys and node identifiers in a one dimensional key space. The nodes connect each other to form a ring, so that the node with the lowest ID follows the node with the highest ID. The node whose ID most closely follows a key is responsible for that key , and is called as the successor of the key. Each node keeps a routing table, called finger table, which contains a relatively small number of special links. These links point to the nodes that are $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and so forth distance in the ID space from the current node. To improve the reliability of routing, each node additionally keeps a list of links to the next several successors, which is called successor list. The KBR is done by using the finger tables to iteratively approach the node that is responsible for a given key . When a key is queried on a node, the node checks its finger table and forwards the message to the node whose ID is highest but not greater than the key. The power-of-two structure of the finger table guarantees $O(logN)$ routing cost, which is the standard DHT performance. To join the network, a new node chooses an arbitrary ID (different from other nodes), and finds the node responsible for this ID. Then the new node is positioned into the ring immediately following the responsible node. Part of the keys on the responsible node must be redistributed to the new node, and all existing nodes have to update their finger tables. If a node leaves, its responsible keys are moved to its successor, and its predecessor connects its successor to form the new ring.

CAN [RFH$^+$01] uses a d-dimensional Cartesian coordinate space for keys and node IDs. The coordinate space is partitioned into hyper-rectangles, also called zones. Each node is responsible for one zone, and is identified by the boundaries of that zone. Two nodes are neighbors if their zones share a $d-1$ dimensional hyper-plane, and the routing table on each node contains links to its neighbors. Routing is done by forwarding a key along the path that approximates the straight line in the coordinate space from the issuer of the key to the node responsible for the key. The size of routing table is $O(d)$.

Let us assume that d is $O(\log N)$, then the routing cost would be $O(dN^{1/d}) = O(\log N)$, which also presents standard DHT performance. To join the network, a node chooses an arbitrary point in the key space, and asks any node in the network to find the node responsible for this point. Then the new node requests the responsible node to split its zone into two equal parts, and one of them is assigned to the new node. If a node leaves, its zone is taken over by a neighbor.

Pastry [RD01] also uses a ring to represent its key space. Each node randomly choose 128-bit ID in the base 2^b, where b is usually 4. The node with the ID numerically closest to a key is responsible for that key. Routing is implemented under a prefix-based forwarding scheme. A node X stores a leaf set L, which consists of: (1) the set of $\frac{|L|}{2}$ nodes whose IDs are the closest to and smaller than X's ID, and (2) the set of $\frac{|L|}{2}$ nodes whose IDs are the closest to and larger then X's ID. Similar to Chord's successor list, this leaf set ensures correct routing. The routing table contains $\log_{2^b} N$ rows, each of which has $2b - 1$ links that point to the other nodes in the ID space. A node uses its leaf set and the routing table to forward a key to the node responsible for that key. If the key is covered by the leaf set, the message is directly forwarded to the node responsible for the key. If the key is not covered by the leaf set, the node checks its routing table to find a node that has a longer shared prefix with the key. If the node is found, the message is forwarded to that node. If such a node can not be found, the message is forwarded to the node with the same shared prefix as current node, but numerically closer to the key. The routing cost is upper bounded to $\log_{2^b} N$.

A common problem in these decentralized structured P2P architectures is the high cost associated with maintaining the structure of the network, especially in a network with high degree of dynamics, i.e. high churn rate. Some specific replication algorithms have been proposed to improve the reliability of the decentralized structured P2P systems; however, they have not promoted the resilience of the structured ones to the level of unstructured ones.

2.3.3 Decentralized Unstructured P2P

In decentralized unstructured P2P architecture, nodes randomly connect each other and replicate their shared content or indirection pointers to the content to other nodes. To find desired content, a node typically floods a query to a large number of nodes, each of which then checks its local storage and response to the requesting node if the desired content is found.

Gnutella [gnu, SGG03] is the first example of a decentralized unstructured P2P system which does not depend on any central server to operate and uses query flooding techniques to search content.

Freenet [CSWH01] builds on the same concept of total decentralization, but segments each shared file into pieces and actively replicates these small pieces onto other nodes in the network. A query is also flooded to many nodes but one node normally can only return some pieces of a file. The originator of the query takes responsibility to reconstruct the whole file after receiving all the pieces of a file. Such

a implementation emphasizes the anonymity of the content issuer, but the distribution of file pieces pose extra communication load in the network.

The common problem of the decentralized unstructured P2P architectures is the inefficiency in query routing. Flooding techniques usually impose heavy communication burden in the network, and a shared content is not guaranteed to be found even without churn.

2.4 Decentralized Distributed Information Retrieval

A variety of P2P network organizations for information retrieval have been investigated in the past works. The unstructured networks usually replicate documents (or index) and broadcast queries to a large number of participating nodes. In case of full text retrieval, these operations can easily consume an unacceptable high level of bandwidth. Thus, some widely known approaches, e.g. random walks, smart query routing and hierarchical structured network, have been proposed to reduce the traffic in various P2P networks. A commonly used technique is to maintain peer-level document collection descriptions, in order to identify nodes that are more likely to answer a query well in the retrieval stage. A query is then only sent to those identified nodes, so as to avoid the high cost of broadcasting. Such technique works well for the clustered P2P networks, i.e. a small subset of nodes holding the majority of documents relevant to a query. However, the retrieval quality usually suffers when the size of the network increases, if the traffic is expected to be kept low.

It is reputed [YDRC06] that structured networks can more naturally support full text retrieval, using term partitioning scheme as shown in Figure 2.3 (b). For example, a DHT can simply hash terms into keys to maintain a global inverted index in the distributed network, and thus provide a straightforward solution to single term look-ups. A multi-term query can be resolved by first retrieving each single term's posting list and then intersecting them at the originator of the query. However, this approach has two problems: (i) high maintenance costs to keep the index distribution correct, (ii) high communication costs for transferring large posting lists. Again, several solutions have been proposed to address these issues.

CORI [Cal00] and the Kullback-Leibler divergence-based algorithm [XC99] try to select only a small set of nodes that contain a large number of documents relevant to a query, Nodes are ranked by their likelihood of storing relevant documents, and only top-ranked nodes are selected.

The federated search system proposed in [LC03, LC05] uses a hierarchical P2P network structure. There are hubs and leaf nodes in the network. Leaf nodes are clustered and connected to hub nodes. A query is initially sent to one or more hub nodes. The hub node then selects some leaf nodes as well as other neighboring hub nodes based on their descriptions, and forwards the query to them. A Time-To-Live (TTL) counter is used to limit the resources used by each query. Leaf nodes match the query against their local document collections, and the results are returned back to the hub nodes. Hub nodes

merges answers from different leaf nodes into a single set before displaying back to query issuer. In [LC06, Lu07], resource selection is further improved by modeling past users' behavior to forward a query into the light loaded part of the network.

SemreX [JC08] uses an unstructured P2P network to share desktop documents. Semantically similar nodes are clustered and connected with each other. In addition, long-range links are also established to enable efficient content-based query routing.

PlanetP [CAPMN03] provides content addressable publish/subscribe service in an unstructured P2P network. It summarizes a node's document collection based on a *bloom filter* [Blo70], then uses these summaries to estimate the inverted term-to-node statistics and forward a given query to only those nodes that are more likely to process the query well.

Minerva [BMT$^+$05a, BMT$^+$05b] also summarizes peer-level document collections and keeps a distributed global index with peer statistics in a structured P2P network. The global index only holds compact meta-information about the nodes' local documents. A query originator first retrieves term-to-peer information from the global index, and then forwards the query to a few most promising nodes based on their published per-term peer statistics. The results from different nodes are then consolidated and displayed back to the user.

All unstructured distributed architectures discussed above do not provide an integrate model that mathematically describe each stage in the retrieval process. Hence, it is difficult to precisely control and optimize the retrieval performance. In next chapter, we propose a new distributed search architecture called Probably Approximately Correct (PAC) search, which mathematically models the full process of document acquisition and querying in an unstructured distributed network. We define important system parameters and provide formulas to estimate the search quality based on these parameters. We then compare PAC with a fully managed deterministic DIR system, assuming the same amount of resources to be used.

Chapter 3

Probably Approximately Correct Search

In this chapter, we investigate the expected performance of a non-deterministic information retrieval system consisting of a set of independent computers. We define a non-deterministic information retrieval system to be one in which (a) the set of unique documents indexed may be selected (in part) randomly and/or (b) the response to a query may (in part) be a function of a random process. By "independent computers" we mean computers that do not communicate between one another for the purposes of either building the index, or responding to a query. The absence of communication/coordination between computers prevents the non-deterministic IR system from overloading the communication infrastructure, and provides a system architecture that is very scalable and reconfigurable.

Our system assumes two capabilities. First, the ability to randomly sample documents from a collection. And second, the ability to randomly sample/query computers within the network. The random sampling of documents within a collection, is, of course, trivial if the collection is available as a static document set with limited number of documents. However, if the collection is considered to be the Web, then it is necessary to randomly sample pages from the Web. This is more difficult. A comparison of several techniques can be found in [BHK+09, RPLG01]. These sampling techniques usually manipulates a random walk and consists of two phases: (1) a walk phase, where a memoryless random walk is performed on an aperiodic and irreducible graph, and converges to a unique stationary distribution. Under such a distribution the probability of being in a node is proportional to either its degree or its PageRank. (2) a subsampling phase, where nodes are subsampled with values inversely proportional to their degrees or PageRanks, in order to get a uniform random sample. The random sampling of computers within a network is straightforward when the computers are a part of a "centralized" cluster. And recent works [KS04, TLB+07, TKLB07], based on distributed hash tables, provides algorithms for choosing a random peer within a peer-to-peer network. It has also been proven in [BGK+09] that uniformly sampling random nodes in a large unstructured peer-to-peer network is feasible.

We are interested in comparing the performance of non-deterministic and deterministic IR systems. In this regard, we consider the results of the deterministic system to be correct, i.e. we are not judging

the performance of our system based on standard IR metrics such a mean average precision (MAP). Rather, given a deterministic implementation of a specific IR system, how close will the outputs of a non-deterministic system be to the deterministic system? Given this measure, we refer to our system as a PAC (probably approximately correct) IR system, in (broad) analogy with PAC learning [Val84].

In Section 3.1 we first define a number of terms and concepts before deriving analytic expressions describing the expected coverage of a PAC IR system and its expected level of correctness. Section 3.2 then discusses the performance of a PAC for two specific configurations, the first of which models the architecture used by search engine services, and the second models a hypothetical peer-to-peer network configuration. Section 3.3 provides simulation results that support the previous theoretical analysis. Finally, Section 4.4 summarizes our results and suggests avenues for future work.

3.1 Framework

Our model of an IR system assumes a set of computers, and that each independently samples a fraction of available documents to construct a corresponding inverted index. We refer to this as the acquisition stage. Next, a user query is issued to a (small) subset of these computers and each computer independently responds with a corresponding result set. These result sets are then merged by the query issuer to produce the overal result set. We refer to this as the retrieval stage.

In the next Section, Section 3.1.1, we first define a variety of terms and concepts. Section 3.1.2 then considers the acquisition stage, and derives an analytic model for the expected coverage of our PAC IR system. This model is then used in Section 3.1.3 to derive an analytic model for the correctness of a PAC IR system.

3.1.1 Definitions

The entire set of unique documents is referred to as the *collection*. The size of the collection is denoted by m. For the Web, m ranges from 17 to 65 billion webpages, as noted earlier.

Let n represent the total number of computers available to perform searches. Note that in the case of peer-to-peer networks, n is not constant. However, in such a case, let us assume n represents the average number of available computers. For simplicity, we assume that the computers are homogeneous. However, this is not needed in practice.

Each computer is assumed to be capable of indexing ρ unique documents, which form an individual sample from the collection. We assume that $\rho \leq m$, and, in practice, normally $\rho \ll m$. We define the "collection sample" as the union of individual samples. As such, the collection sample may well contain duplicate documents. We define coverage as the ratio of the number of unique documents in a collection sample to the size of the collection. Finally, during retrieval, we query a subset, z, of computers, and the union of their indices is known as the "retrieval index".

3.1.2 Sampling the Collection

In order to index the m distinct documents, the n computers must sample the collection (Web). In our analysis we assume that each computer operates independently, with no cooperation between computers. In such a scenario, there is no guarantee that the samples on each computer will be disjoint. In fact, it is almost certain that documents will be sampled more than once, i.e. they will be indexed by more than one computer. This redundancy is, in fact, helpful. First, it provides tolerance to node failures, and to the dynamic entry and exit of nodes in a peer-to-peer network. Second, the redundancy allows only a subset of nodes to answer a query (see Section 3.1.3), which both reduces the communication overhead and increases the throughput, i.e. the number of queries that can be answered per second.

Independent sampling of the m documents in the collection by each of the n computers is analogous to having an urn with m labeled balls. Each of n people, then randomly choose ρ balls each. An individual chooses his/her ρ balls without replacement, thereby guaranteeing that there is no repetition on a single computer. After indexing the ρ balls, they are returned to the urn. Thus, the next person may also randomly select balls that have been previously chosen by other people (i.e. indexed by other computers).

The key question to answer is, how many different balls have been drawn from the urn after all n people have each randomly picked ρ balls? The answer to this question determines the coverage obtained after all n computers have sampled ρ documents.

Obviously, the coverage ranges from $\frac{\rho}{m}$ in the worse case, where all computers sample the same set of n documents, to $\frac{\min(m,n\rho)}{m}$ in the best case, where each computer's sample is disjoint from all other computers' samples. Treating the coverage as a random variable, we need to understand its probability distribution. Of course the complete probability distribution may be quite complicated. However, from a practical point of view, we believe that an analysis on the expected coverage would be sufficient to explain the underlining rules of our algorithm.

The probability of a ball being picked by a single individual is $\frac{\rho}{m}$, and the probability of *not* being picked is therefore $1 - \frac{\rho}{m}$. Thus the probability, $P(\bar{d_i})$, that document d_i will *not* be picked by *any* of the n people is

$$P(\bar{d_i}) = \left(1 - \frac{\rho}{m}\right)^n \tag{3.1}$$

Thus, the probability, $P(d_i)$, of being chosen one or more times in the total sample is

$$P(d_i) = 1 - P(\bar{d_i}) = 1 - \left(1 - \frac{\rho}{m}\right)^n \tag{3.2}$$

and the expected number of distinct documents in our total sample, $\hat{m}$, is

$$\hat{m} = P(d_i)m = \left(1 - \left(1 - \frac{\rho}{m}\right)^n\right) m \tag{3.3}$$

To simplify our further analysis, let us now set

$$\epsilon = P(\bar{d_i}) = \left(1 - \frac{\rho}{m}\right)^n \tag{3.4}$$

Then

$$\hat{m} = m(1 - \epsilon). \tag{3.5}$$

And the expected coverage can be defined as

$$E(Coverage) = \frac{\hat{m}}{m} = 1 - \epsilon. \tag{3.6}$$

Thus, as ϵ approaches zero, our coverage approaches unity, i.e. we approach a complete sampling of the document collection.

We can use Equation (3.4) - Equation (3.6) to determine the coverage. We are interested in the relationship between coverage, the size of the individual sample, ρ, and the number of computers, n, given a certain collection size, m. In particular, given a collection, how many machines do we need, and what capacity should each of them have, to meet a desired level of performance for our system.

To help our analysis, let us first denote c as the size of the collection sample, where $c = n\rho$. The collection sample, c, is treated as a constant in the following analysis. Also to simplify our analysis, let us first assume that $c \leq m$. From Equation (3.4), we have

$$\epsilon = \left(1 - \frac{\rho}{m}\right)^n = \left(1 + \frac{-\frac{c}{m}}{n}\right)^n \tag{3.7}$$

Thus, if the collection sample, $c = n\rho$ is a constant, then ϵ is a monotonically increasing function with respect to n. The smallest value of $\epsilon = (1 - \frac{\rho}{m})$ occurs when $n = 1$. In this case, ρ is at its largest, and the coverage is maximized since there are no duplicates in our collection sample. Conversely, as the number of computers, n, increases, ϵ increases, approaching the limit of $e^{-\frac{c}{m}}$ as n approaches infinity. The proof is shown as below.

From the property of exponential functions, we know that

$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n \tag{3.8}$$

From Equations (3.7) and (3.8), we have that

$$\lim_{n\to\infty} \epsilon = \lim_{n\to\infty} \left(1 + \frac{-\frac{c}{m}}{n}\right)^n = e^{-\frac{c}{m}} \tag{3.9}$$

Next, the derivative of ϵ with respect to n is

$$\frac{\partial \epsilon}{\partial n} = \epsilon \left(\ln\left(1 + \left(-\frac{c}{mn}\right)\right) - \left(\frac{-\frac{c}{mn}}{1 + (-\frac{c}{mn})}\right)\right)$$

From the property of natural logarithms, we also have

$$\ln(1+h) \geq (\frac{h}{1+h}), \text{for } h \geq -1 \tag{3.10}$$

Since

$$\rho \leq m \Rightarrow \rho n \leq mn \Rightarrow -\frac{c}{mn} = -\frac{\rho n}{mn} \geq -1$$

So $\ln\left(1 + \left(-\frac{c}{mn}\right)\right) - \left(\frac{-\frac{c}{mn}}{1+(-\frac{c}{mn})}\right) \geq 0$, because $\epsilon \geq 0$, we have

$$\frac{\partial \epsilon}{\partial n} = \epsilon \left(\ln\left(1 + \left(-\frac{c}{mn}\right)\right) - \left(\frac{-\frac{c}{mn}}{1 + (-\frac{c}{mn})}\right)\right) \geq 0 \tag{3.11}$$

Combining Equations (3.9) and (3.11), we show that the ϵ increases monotonically with an upper bound of $e^{-\frac{c}{m}}$, as n increases. Thus, the expected coverage ranges from $(1 - e^{-\frac{c}{m}}$ to $\frac{c}{m}]$. Figure 3.1 plots ϵ and coverage for the case where $m = 1000000$ and $c = m$.

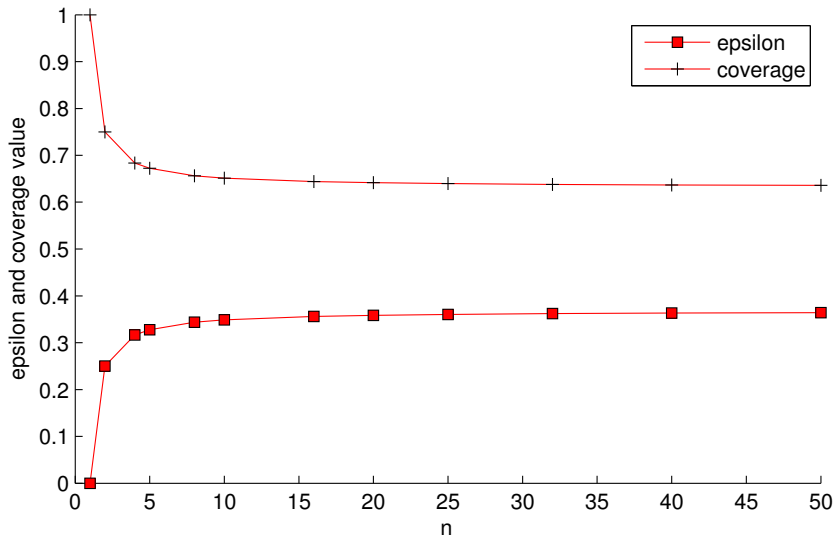

Figure 3.1: Simulation calculating ϵ and coverage as a function of the number of computers, n, when the number of documents, $m = 1000000$, and the collection sample is $c = m$.

This monotonic property remains true when we relax the assumption that $c \leq m$, and allow $c > m$,

provided $\rho \leq m$. In this case, n cannot start from 1 since it would imply that $\rho > m$. Let us define n_{min} as the smallest value of n such that the property $\rho \leq m$ is maintained. Then, a more general form of coverage can be written as $(1 - e^{-\frac{c}{m}}, 1 - (1 - \frac{c}{mn_{min}})^{n_{min}}]$

In summary, for any given c, we have a lower bound, $1 - e^{-\frac{c}{m}}$, for the expected coverage. The smallest coverage occurs when $n = c$ and $\rho = 1$, and approaches $1 - e^{-\frac{c}{m}}$ if n is large enough. Conversely, coverage is maximized when $n = n_{min}$, and is given by $1 - (1 - \frac{c}{mn_{min}})^{n_{min}}$.

Unfortunately, coverage is not our only concern. We must also consider the throughput of the system, as well as the system's latency. However though smaller n promises a larger coverage, it results in a larger individual sample, ρ. Let us define z as the number of machines that process a query simultaneously, and p as the number of documents that each machine can process in a unit time. Then, the query rate, T, can be defined as

$$T = \frac{n}{z} \times \frac{p}{\rho} \tag{3.12}$$

The first factor represents the query throughput of the system, and the second factor is the inverse of the latency. Suppose z and the collection sample size, c, are fixed, then

$$T = \frac{n^2 \times p}{z \times c} \propto n^2 \tag{3.13}$$

Obviously larger n increases the query throughput, but, as we discussed earlier, a larger n decreases coverage, when our sample collection size, c, is fixed. Thus, for a given c, choosing appropriate values of n and z is a tradeoff between coverage and query rate, and will depend on the application.

For example, consider the case where the size of collection sample, c, is equal to the size of the collection, m. Using Equation 3.9, we can easily infer that ϵ tends to be $e^{-1} = 0.367$ with a large n. Inserting this value in Equation (3.6), we see that when we sample m documents, our expected coverage is at least $1 - \epsilon = 0.63$.

Next, let us assume we want complete coverage. Of course, this cannot be guaranteed, but we can set ϵ to a small value such that the probability of missing a document is low. For example, consider the case when $\epsilon = 10^{-2}$, say. That is, 99% coverage. In this case, from Equation (3.9) we have

$$\epsilon = e^{-\frac{c}{m}} = 10^{-2} \Rightarrow \frac{c}{m} \approx 4.6$$

Thus, if the size of the collection sample is 4.6 times the size of collection, we can expect 99% coverage of the collection.

3.1.3 Retrieval

The previous theoretical analysis elucidated the connection between (i) the number of computers, n, (ii) the size of each computer's sample, ρ and (iii) the fraction of the collection that is not indexed, ϵ. By increasing n and/or ρ, we can make ϵ as small as desired. Of course, in practice, economic considerations can limit the values of both n and ρ.

When performing retrieval within such an architecture, we wish to send the query to z randomly chosen nodes, where $z \leq n$, and normally $z \ll n$. This is because it is necessary to (i) limit the amount of communication generated by a query, (ii) limit the computational resources expended in responding to a query, and (iii) limit the latency between query issue and response.

Clearly, if we only interrogate z machines, we cannot guarantee the coverage provided by all n machines. However, Equation (3.3) can be used to determine the expected size of the index used during retrieval, i.e. the expected number of distinct documents in the retrieval index. For this, we simply have to replace n with z.

$$\hat{m'} = P(d'_i)m = \left(1 - \left(1 - \frac{\rho}{m}\right)^z\right) m \tag{3.14}$$

The probability of any document being present in the retrieval index is then

$$P(d'_i) = 1 - \left(1 - \frac{\rho}{m}\right)^z \tag{3.15}$$

In practice, information retrieval systems are seldom evaluated based on a single target document. Instead, performance metrics such as precision and recall are often used. In our case, we assume that the retrieval model is identical, irrespective of whether we are using a deterministic or non-deterministic search architecture. Thus, if we want to compare our PAC strategy to a deterministic implementation of the IR system, we need to consider what the expected overlap in the two result sets is. Thus, given the top-k documents from the deterministic system, what is the probability that our PAC IR system will retrieve k' documents from k, where $k' \leq k$.

Let us define that the correctness of a PAC search is measured by *retrieval accuracy*, a, given by

$$a = \frac{k'}{k}. \tag{3.16}$$

We know from the Equation (3.15) that the probability of a specific document being present in the result set is $P(d'_i)$. Thus, the probability of exactly k' documents from k being present in the result set is

$$P(k') = \begin{pmatrix} k \\ k' \end{pmatrix} P(d'_i)^{k'} (1 - P(d'_i))^{k-k'} \tag{3.17}$$

This is a standard binomial distribution, and the expectation of k' is therefore

$$E(k') = kP(d'_i) \tag{3.18}$$

Equation (3.18) indicates that acceptable performance using PAC search can be achieved provided the probability, $P(d'_i)$, is sufficiently high. We will discuss this problem in more detail in the next section, where we consider several practical applications. And from Equation (3.16) we can obtain the expected retrieval accuracy as

$$E(a) = \frac{E(k')}{k} = 1 - \left(1 - \frac{\rho}{m}\right)^z \tag{3.19}$$

3.2 Discussion

Information retrieval systems can be broadly categorized into one of three architectures, namely (i) single server search, (ii) distributed and, arguably, "virtually centralized" search, and (iii) peer-to-peer decentralized search. We do not consider the first case, as we assume that our collection and/or query rate is too large to be handled by a single machine. The second case represents the architecture used by commercial search engines such as Google [BDH03b]. Finally a variety of peer-to-peer decentralized architectures have been proposed and deployed [SMk+01, TXM02, HHH+02, RV03, YDRC06, YH06, SLZ+07] with a variety of search capabilities. In the following two subsections we examine the second case and the third case, respectively.

3.2.1 Distributed Search

Due to the rate of queries and the huge size of the Web, modern commercial search engines partition the Web index over many machines. A response to a query requires each partition to be independently searched. Each partition (Google refers to them as index shards [BDH03b]) contains a "randomly chosen subset of documents from the full index" [BDH03b]. Note, however, that while the documents may be chosen at random, each partition is disjoint. In addition, replicas are added to each partition to increase the query throughput. This architecture is referred to as a distributed cluster architecture.

The key parameter of such an architecture is the tradeoff between the replication and partitioning. While increasing the partitioning level, which reduce the replication level, improves the query completion time since more machines process the same query simultaneously, the reduced replication level decreases the number of different queries that can be answered at the same time. A crucial problem faced by these engines is to find a best compromise between partitioning and replication, especially as the data set and the query rate change continuously. Clearly this compromise changes over time, as the database and query loads change. However, changing the partitions and replications can be expensive in both time and bandwidth, as reported in [RHHR09b].

In [RHHR09b] it is claimed that Google partitions its index into 1000 disjoint sets. Thus, the number of documents indexed by a single machine is $\frac{\rho}{m} = \frac{1}{1000}$. It is further claimed that the data in any partition is replicated over 300 machines, so the total number of machines is $n = 300,000$, and the total number of samples is $n\rho = 300m$. Let us now examine the performance of such a system, when configured for PAC IR.

First, let us consider the expected coverage when each of the n machines, independently samples 0.1% of the Web. Solving for ϵ in Equations (3.7) and (3.9), we have

$$\epsilon = \left(1 + \left(\frac{-300}{300000}\right)\right)^{300000} \approx e^{-300}$$

This is a very small number and indicates that if all 300,000 machines were to each, independently randomly sample and index 0.1% of the Web, then it is almost certain the every document on the Web would be contained in the combined index.

For the query part, let us consider the configuration ascribed to Google, in which 1000 machines, one per partition, are used to service each query. In this case, $z = 1000$ and $\frac{\rho}{m} = \frac{1}{1000}$, as before. Substituting in Equation (3.15) we get

$$P(d_i') = 1 - \left(1 - \frac{1}{1000}\right)^{1000} \approx 0.63$$

Thus, if a user is looking for a particular document, there is a 63% chance that it is present in a subset of 1000 randomly chosen nodes. That is, approximately two thirds of the time, the user will find the specific target document.

Assuming we are primarily interested in the top-10 results, i.e. $k = 10$, and given $P(d_i') = 0.63$, substituting in Equation (3.18), gives

$$E(k') = 10 \times 0.63 = 6.3$$

This shows that we can, on average, expect 6 documents from the top-10 retrieved by a deterministic search algorithm to be present in our PAC IR top-10.

We can also use Equation (3.17) to calculate the probabilities for all possible k'. These probabilities are enumerated in Table 3.1 for $k = 10$ and $0 \leq k' \leq 10$.

Table 3.1 indicates that there is over an 88% chance of retrieving 5 or more documents in common with the deterministic solution. And the most likely situation, occurring about 25% of the time, is that 6 out of the 10 documents will be common. There is approximately a 1% chance that the PAC search result set will be identical to the deterministic case. In contrast, the likelihood that the PAC search results do not contain any of the documents from the deterministic case, occurs less than 0.01% of the time.

k'	$P(d_1 \cdots d_{k'})$
0	0.000045173
1	0.00077682
2	0.0060113
3	0.027566
4	0.082957
5	0.17119
6	0.24532
7	0.24106
8	0.15545
9	0.059405
10	0.010216

Table 3.1: The probability of exactly k' documents being present in the top-10

In summary, the performance of our PAC IR system is approximately 63% of the deterministic system, when utilizing equivalent resources. Of course, we can improve performance by simply increasing the number of machines the query is sent to. For example, if we send the query to 2000 servers, then the query correctness increases to 86%. Unfortunately, this is at the expense of halving the query throughput. However, this example serves to highlight the flexibility of PAC search, which allows accuracy to be traded for throughput. That is, a PAC IR system could choose to tradeoff accuracy for query throughput during peak load periods.

Due to the unstructured nature of the PAC IR system, it is also straighforward to add and remove machines as well as adjust the data present on a machine.

3.2.2 User Satisfaction Optimization

The user satisfaction can be simply improved by re-sending a failed query to another random subset of nodes. Thus, together with the search results obtained in the first round of the search, it is likely that the chance of an unsatisfied search can be reduced to an ignorable level, while the average communication cost imposed on the network is not increased significantly. Certainly, this depends on how we define the user satisfaction.

Let us consider a simple example. Suppose the user is only interested in top-2 documents of a query. Let us assume that the user can be satisfied when the system returns at least the top 1st document from those top-2 documents. Given a general PAC configuration, where $m = 2 \times 10^10$, $\rho = 2 \times 10^7$ and $n = 10^6$, we can list all cases the user may encounter, with the corresponding probabilities under different search size, in Table 3.2

Suppose the bandwidth limit restricts the maximum search size to 1000 at each time. If we only allow the user to search once for a query, the user satisfaction is only 63%. However, if we take another policy, which simply allow the user to re-send the query if he/she is not satisfied. The user satisfaction can be improved to $0.4 + 0.2325 + (0.2325 + 0.135) \times (0.748 + 0.117) = 95\%$. And the average search size is $1000 * (0.4 + 0.2325) + 2000 * (0.2325 + 0.135) = 1368$, only 36.8% increase in the

Table 3.2: All cases a user may encounter if only top-2 documents are of interest

Top-2 documents	Retrieval state			
Rank 1st	$\surd$	$\surd$	$\times$	$\times$
Rank 2nd	$\surd$	$\times$	$\surd$	$\times$
Probability ($z = 1000$)	0.4	0.2325	0.2325	0.135
Probability ($z = 2000$)	0.748	0.117	0.117	0.018

average search size. In comparison, if the maximum search size is allowed to be 2000, it incurs a 100% increase in communication cost, with the user satisfaction only improves to 86.5%. This suggests that by allowing the user to re-send a query when he is not satisfied, a higher level of the user satisfaction can be achieved, at the cost of a small increase in communication overhead. Of course the user satisfaction criterion can be defined more complicated, but the effect of the query re-sending policy could be similar.

3.2.3 Peer-to-Peer Search

Another possible implementation of PAC IR is in peer-to-peer search. We assume a probabilistic search architecture based on the PAC model. The topology of the peer-to-peer network could be as simple as a random graph with average degree equal to 4. A query could be issued from any node and forwarded to a set of nodes by the simple flooding technique. That is, a query has a Time-To-Live (TTL) parameter to control its forwarding process. A node forwards a query to all its neighbors and reduces the TTL by 1 for each forwarding. The query forwarding process halts when the TTL reaches 0. We further make the following assumptions with regard to the system:

- a network size of $n = 1,000,000$ nodes.

  Several P2P services already exceed this number, e.g. Gnutella and BitTorrent, and the commercial P2P information retrieval system Faroo[1] currently claims 1 million users.

- a query rate of 1,000 queries a second.

  The estimated query rate of Google is 38,000 queries per second.[2] However, Google's query rate is based on a user community of about 150M unique users.[3] A query rate of 38,000 queries per second is equivalent to each of 1 million nodes issuing over 2 queries a minute! A rate of 1000 queries per second corresponds to each node issuing almost 4 queries an hour, 24 hours per day, which would seem like an upper bound on any realistic query rate.

- a collection size of 10 billion documents to be indexed.

  Currently, it is estimated that Google indexes approximately 20 billion documents, while Bing and Yahoo index approximately 12 billion documents.[4]

[1] http://www.faroo.com/hp/p2p/p2p.html
[2] http://searchengineland.com/by-the-numbers-twitter-vs-facebook-vs-googlebuzz-36709
[3] http://siteanalytics.compete.com/google.com+facebook.com+yahoo.com/
[4] http://www.worldwidewebsize.com

- a required expected retrieval accuracy of 90%.

  For each query, the expected accuracy is given by Equation (3.19). Thus, we must choose a combination of the number of nodes the query is sent to, z, and the local storage capacity, ρ. Let $f = \frac{\rho}{m}$, denote the fraction of the global collection indexed at a node.

- a minimum storage of 5 GB, available at each node and a maximum of 10 GB.

  This allows a node to index $f = \frac{1}{1000}$ of the global document collection as discussed shortly.

Figure 3.2 illustrates the expected accuracy as a function of the number of nodes queried when each node randomly samples $f = \frac{1}{1000}$ of the global document collection. For 90% accuracy we need to query approximately 2,300 nodes. Figure 3.3 shows the number of nodes that need to be queried to obtain 90% accuracy as function of f.

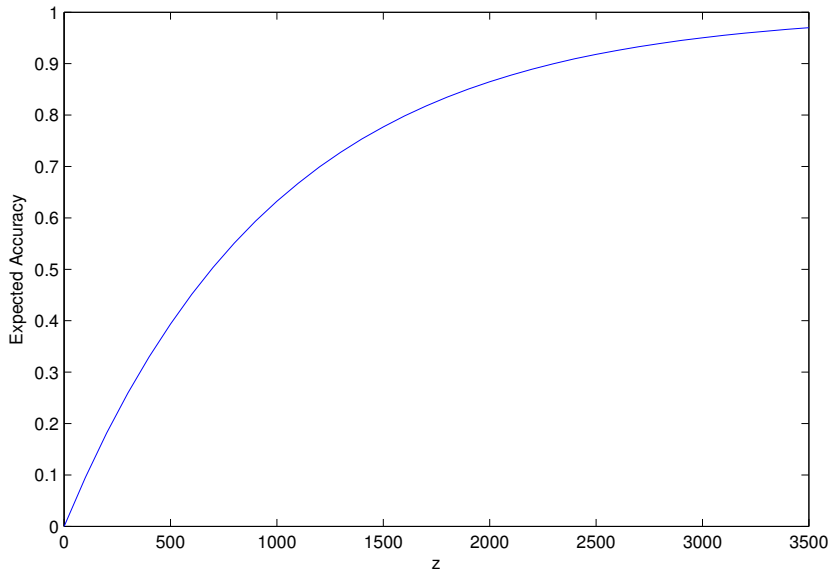

Figure 3.2: Expected accuracy as a function of the number of nodes queried when $f = 0.001$.

In order to estimate the communications load we further assume:

- an average of 2 bytes per character.

  This is based on UTF-8 encoding, where each character takes between 1 - 4 bytes depending on the language used.[5]

- a query message size of 300 bytes.

  Analysis of query logs [BYGJ+07] has shown that the average query size is 2.5 terms or 30 characters. This corresponds to about 60 bytes per query message. However, we must also assume some overhead associated with the underlying TCP/IPv6 protocol. We therefore conservatively

[5] http://tools.ietf.org/html/rfc3629

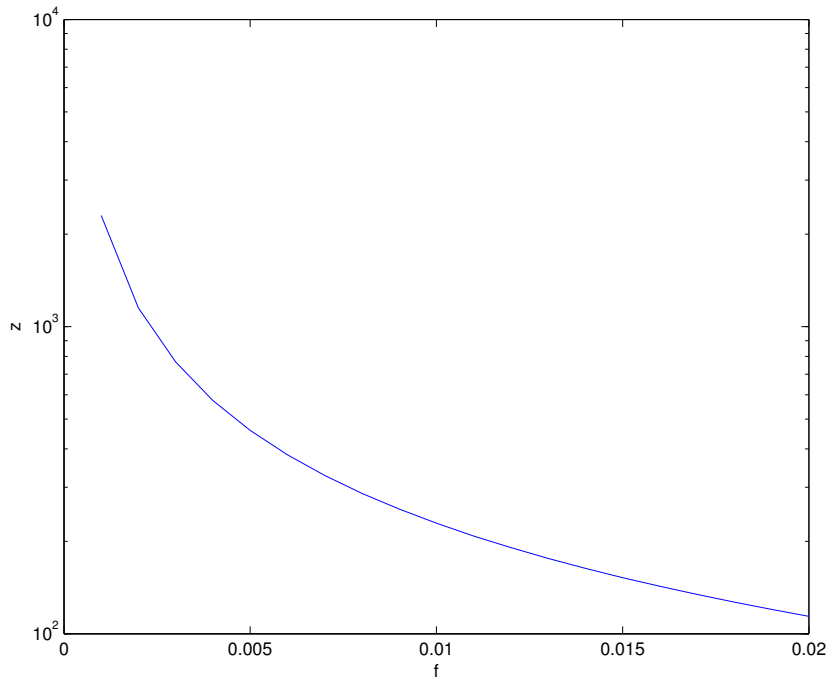

Figure 3.3: The number of nodes queried for 90% accuracy as a function of f. Note that this is a log-linear plot.

assume a query message size of 300 bytes. Therefore if this message must be sent to $z = 1,000$ peers, the communication cost associated with *sending* a query is 300 KB.

Finally, in order to estimate the communication bandwidth needed to respond to a query, we assume the following:

- we are only interested in the top-10 documents.

  Analysis of commercial search engine query logs show that users rarely look beyond the top-10 documents. Thus, when a user issues a query, each node only needs to return its top-10 URLs. If, however, the user requests to see results 11-20, we could ask the same nodes to return their top 11-20, which would again be merged and re-ranked at the node originating the query.

- a query response size of 1KB.

  We estimate that each result (result name, hyper-link, snippet, minimal surrounding XML etc) requires no more than 400 characters or 800 bytes. Since the query result is entirely alphanumeric, it can usually be compressed to 10% of its original size. This is common practice with modern web servers.[6]

  Thus the total bandwidth required to answer a query is simply 800 bytes per result, multiplied

[6] http://httpd.apache.org/docs/2.0/mod/mod_deflate.html

by 10 results per query, times 0.1 compression factor, i.e. 800 bytes. We round this to 1 KB to account for TCP/IPv6 overheads.

3.2.3.1 Communication Load

Based on the previous assumptions, we consider the communication load imposed on the peer-to-peer network. From Equation (3.19), the expected accuracy of 90% can be obtained by sending the query to 2,300 nodes, assuming each node indexes 0.1% of the global collection.

We are now in a position to calculate the total communication load of such a system. For broadcasting the query and receiving the response from 2,300 peers, the total cost per query is approximately 3 MB. For 1,000 queries per second, the total traffic generated is 3 GB/s.

Note that this traffic is spread throughout the internet. The total internet traffic in 2009 was approximately 4,630 GB/s [LIJM+10, min] and is forecast to grow by 50% each year, primarily due to video traffic. Using the 2009 figures, the traffic generated by a PAC web IR service would only constitute 0.065% of the global internet traffic. Thus, web search using an unstructured P2P network will not impose a significant global communication cost.

As well as the global communication cost, it is useful to consider the requirements placed on each node, both in terms of storage and bandwidth.

We now estimate the resource requirements on each peer participating in the search. We have assumed that each node randomly samples 1/1000 of the 10 billion documents which need to indexed. This implies that each peer must index 10 million documents, which must first be crawled. The Akamai internet report[7] states that the global average internet connection speed is approximately 200 KB/s. In the developed nations it is considerably higher, but we do not account for this here. If we assume that 25% of this bandwidth can be utilized (say, during the peer's idle time), it will take approximately 58 days to complete the crawl, assuming that the average size of a document on the Web is 25KB[8].

The crawled documents, representing 250GB of data, can be indexed using approximately 10 GB of disk space which would record term frequencies and positions as well as other statistical measures. This is typical of popular information retrieval packages such as Lucene[9]. We are aware that some machines may not have 10GB of disk storage available for this service. However, lossless compression [WMB99] can reduce the size of the index by utilizing efficient data structures, and lossy index compression techniques [dMdSF+05] have been shown to reduce the size of the index by 50 to 70% with minimal loss in precision.

Using efficient Trie structures, only small percentages of the index need to be read and loaded into memory, and the system can answer queries using no more than 500 MB of the peer's memory, as has

[7] "Akamai report: The state of the internet, 3rd quarter, 2010", http://www.akamai.com/stateoftheinternet/
[8] http://www.optimizationweek.com/reviews/average-web-page/
[9] http://lucene.apache.org/

Simulation	Expectation	Average	Std dev.
1	0.6323	0.6322	0.0003
2	0.8648	0.8648	0.0004
3	0.8347	0.8346	0.0004

Table 3.3: Comparison of expected coverage, average coverage and standard deviation across 20 trials.

been demonstrated by systems such as Lucene.

For a PAC web IR system of 1 million nodes answering 1,000 queries per second, each peer on average would have to answer 2.3 queries per second. The corresponding bandwidth needed is 0.69 KB/s in the download direction and 2.3 KB/s in the upload.

To summarize, each peer would need to contribute 5-10 GB of disk space, 500 MB of memory, and approximately 0.69 KB/s download as well as 2.3 KB/s upload from the peer's bandwidth for query answering as well as 50KB/s during idle time for crawling. Both the local communication, and disk and memory requirements appear reasonable.

3.3 Simulation

The previous theoretical analysis examined the *expected* coverage and corresponding query performance, which is the result of averaging over many trials. In practice, any configuration for a PAC IR system represents a single instance or trail. Thus, it is interesting to investigate the standard deviation form the expected value, across trails. Clearly, we would like this to be small.

We investigated this issue using a simulation with different settings of machine capacity (ρ), number of machines (n) and collection size (m). In the first simulation, we manually generated a collection with $1e+6$ documents ($m = 1e+6$), and set $\rho = 1000, n = 1000$. This synthetic collection was simply a set of document identifiers. In each trial, each of the n machines samples ρ documents to form a collection sample, which is then stored in disk. Then we repeat this process to generate 20 trails and a corresponding 20 collection samples. Next, we randomly generated 100 test queries and computed the top-10 ranking results from the original full collection. Then, for each trial, the queries are replicated to all 1000 nodes and an averaged query performance on each collection sample is calculated. The results for each trial were then avaerged to provide an estimate of the expected values for coverage and query performance.

In the second simulation, we change n to be 2000 with all other parameters being the same as the first one.

In the third simulation, we use TREC45 as our experiment environment to test the performance of PAC. TREC45 contains about 550,000 documents, i.e. $m = 556079$. All other settings are set the same as for the first simulation.

The results from Tables 3.3 and 3.4 show that the variation across trials is very small. This is very

Simulation	Expectation	Average	Std dev.
1	0.6323	0.6264	0.0135
2	0.8648	0.8636	0.0124
3	0.8347	0.8377	0.007

Table 3.4: Comparison of expected query performance, average query performance and standard deviation across 20 trials.

encouraging and supports our analysis in section 3.1, which indicates that in spite of the random nature of the PAC search, the most common outcomes for coverage and query performance concentrate in a short range centered around their expectations.

3.4 Summary

We examined the problem of non-deterministic search in which a set of computers (i) independently sample the collection/Web and (ii) queries are sent to a random subset of computers. Equations are derived for the expected coverage of the sample collection, and the accuracy of the retrieval results. The latter is measured with respect to the results provided by a deterministic IR system. Under the assumption that the deterministic system provides correct result, we consider the probability of being approximately correct. We therefore describe our approach as PAC search.

Our analysis of PAC IR in the context of commercial search engines suggest that a performance level of 63% can be achieved using the same amount of storage and computation. However, while the performance is lower, we believe that the PAC IR architecture may be simpler to manage. Moreover, more sophisticated implementations might close this performance gap.

PAC IR was also analyzed in the context of peer-to-peer decentralized web search. The key problem with such a configuration appears to be the much small storage available on any machine. Consequently, it would be necessary to send the query to many more computers, and the communication overhead may then be too high.

The fact that a query is sent to a random set of machines means that the same search, issued multiple time, is likely to produce different results. Users may find this disconcerting. However, if a pseudo-random set of machines is selected based on a function (hash) of the query, then the result set would remain the same each time the same query is issued. For common queries, additional random machines could be queried to determine if better results exist within the sample collection. If so, these documents could be indexed by the pseudo-random set of machines corresponding to the query. More generally, for common queries, it is interesting to consider how to optimally learn the best set of z machines to answer the query.

A further level of optimization is the caching of query results. First, it would be interesting to analyze the expected cache hit rate for a given distribution of queries when a query is sent to a random set of machines. And a similar analysis should be performed when the query is sent to a pseudo-random

(deterministic) set of nodes.

A key assumption in our analysis is the ability to randomly sample the collection. This is difficult, but certainly possible. Moreover, in the case of a centrally managed system, common to commercial search engines, it would not be necessary to for each machine to independently sample the Web. Rather, a centralized crawler could still be used, and the documents from this crawl could be randomly (and non-disjointly) partitioned across the computers.

We have also implicitly assumed that the deterministic and non-deterministic IR systems both implement the same underlying retrieval model. Usually, most retrieval models have parameter values that are based on the statistics of the collection. However, for the PAC IR system, each computer only has access to its local sample. Future work is needed to determine if, and under what conditions, the statistics of the local samples will be sufficiently close to the statistics of the overall collection.

Chapter 4

Adaptive Node Selection

In Chapter 3 theoretical bounds on the expected performance of PAC search were derived and verified by simulation. In particular, the performance of PAC search was compared to the centralized architecture used by commercial search engines such as Google. Using the same number of computers to store partial copies of the index, and querying a random subset of these computers (equivalent to one computer in each disjoint partition), it was shown that the retrieval accuracy of PAC search is 63%. And if we are interested in the top-10 documents, then there is over an 88% chance of finding 5 or more documents in common with the deterministic solution.

These percentages seem surprisingly high given the random nature of PAC search. Moreover, comparing to the traditional centralized architecture, PAC is advantageous to cope with the updating of the document collection. The traditional centralized architecture relies on repartitioning in response to the new coming documents, and consumes a large amount of bandwidth and computation power in the repartitioning process. In comparison, PAC assumes independent random samples of document on each machine, and the new coming documents can be easily indexed by a random set of machines without complex coordination, thus saves both communication cost and computation time. However, to be practical, the retrieval accuracy is expected to be much closer to 100%. Of course, this can be achieved by querying a larger number of computers. In the example above, 1,000 computers are randomly chosen from a set of 300,000 available computers. If instead, the query is sent to 5,000 computers then the retrieval accuracy is 99%. Unfortunately, this accuracy requires fives times the communication cost expended for an equivalent deterministic search, and is therefore not economically feasible.

It is observed that queries on the web follow a power law distribution, where a small proportion of popular queries have a large number of occurrences, while a large proportion of less popular queries only have a small number of occurrences [BJC$^+$04, FPSO06, SHMM99]. In this chapter, we examine a number of ways in which a centralized PAC architecture, i.e. a PAC architecture deployed in a central retrieval environment such as a Web search engine, can be modified to improve the accuracy of a frequent query, while utilizing computational resources comparable to a deterministic search.

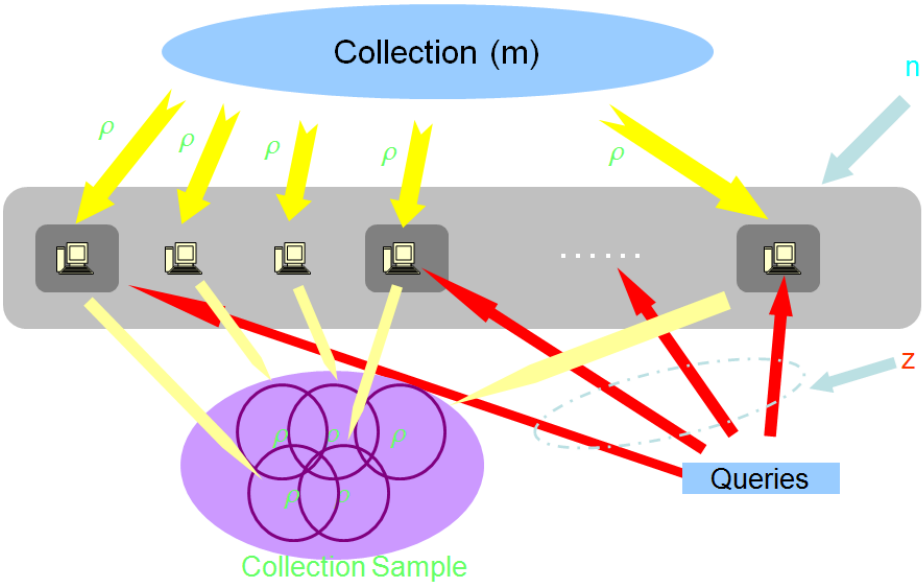

Figure 4.1: Basic elements of the PAC architecture.

In Section 4.1 we briefly describe some basics of a centralized PAC search system. In Section 4.2 we then propose a novel approach which utilizes pseudo-random query node selection to improve PAC search performance. In particular, we consider those queries that frequently occur, and propose an iterative algorithm whereby PAC retrieval accuracy quickly increases to close to 100%. Section 4.3 presents both simulation results, as well as results based on a TREC data set. Finally, we conclude and discuss future works in Section 4.4.

4.1 Centralized PAC Search

The PAC architecture has been proposed in Chapter 3, and its basic elements are illustrated in Figure 4.1. For the analysis in this chapter, we assume that a PAC architecture is deployed in a central information retrieval environment, e.g. Web search engines. It is assumed that there are m unique documents in the collection to be indexed. For Web search, m is the unique number of web pages, currently estimated to be of the order of 65 billion documents [www]. It is further assumed that n computers are available and each computer indexes a random sample of ρ documents. It is reported in chapter 3 that Google utilizes 300,000 computers and we therefore set $n = 300,000$. The ρ documents on each computer are assumed to be unique, i.e. no document appears more than once. However, each computer's sample of ρ documents is *not* disjoint with other computers, so a document may, and very likely will, be indexed by more than one computer. In [RHHR09a] it was reported that the fraction of the collection indexed by each machine in the Google search architecture, $\frac{\rho}{m}$ is 0.001, and we therefore use this ratio in our subsequent analysis.

The union of the n samples is referred to as the collection sample, C_s, and is the union of all the individual samples. The size of the collection sample, $|C_s| = n\rho$. Note that in a deterministic

centralized distributed architecture the same storage capacity is needed. However, in this case, each document is replicated on each machine within a partition.

It was shown in Chapter 3 that if each computer randomly samples the web to acquire its ρ documents, then the expected coverage, $E(\text{coverage})$, i.e. the ratio of the expected number of unique documents in the collection sample to the number of unique documents in the collection, m, is

$$E(\text{Coverage}) = \hat{m}/m = 1 - \epsilon, \tag{4.1}$$

where ϵ is given by

$$\epsilon = (1 - \rho/m)^n \tag{4.2}$$

Since the value inside the parentheses is less than one and $n = 300,000$, then ϵ is effectively zero and our expected coverage $E(\text{coverage})$ is effectively one. Thus, it is (nearly) certain that all documents in the collection will exist within the collection sample.

During retrieval, only a subset, z, of computers are queried, where $z = 1,000$ based on the number of disjoint partitions attributed to Google [RHHR09a]. The union of ρ documents on each of the z computers is referred to as the sample index. Since z is much smaller than the total number of computers, n, the ratio of the expected number of unique documents in the sample index to the number of unique documents in the collection, (equivalent to the probability of any document being present in the retrieval index), given by

$$P(d) = 1 - (1 - \rho/m)^z \tag{4.3}$$

is 0.63 when $z = 1,000$ and $\frac{\rho}{m} = 0.001$. Thus, the coverage during retrieval is much smaller than the coverage during acquisition, and there is therefore a finite chance that the retrieval set provided by PAC will not be the same as for a deterministic search.

If k represents the number of documents retrieved by a deterministic system, and k' represents the number of documents retrieved by PAC search that are contained in the k documents, i.e. $k' \leq k$, then in Chapter 3 it was shown that the probability of retrieving k' documents, $P(k')$ is

$$P(k') = \binom{k}{k'} P(d)^{k'} (1 - P(d))^{k-k'}, \tag{4.4}$$

For the specific case where $\frac{\rho}{m} = 0.001$, $z = 1000$, and $k = 10$ (top-10), the probability of retrieving 5, 6, 7, 8, 9, and 10 documents in top-10 is 17.1%, 24.5%, 24.1%, 15.5%, 5.9%, and 1%, respectively, and the probability of retrieving 5 or more documents in the top-10 is therefore over 88%.

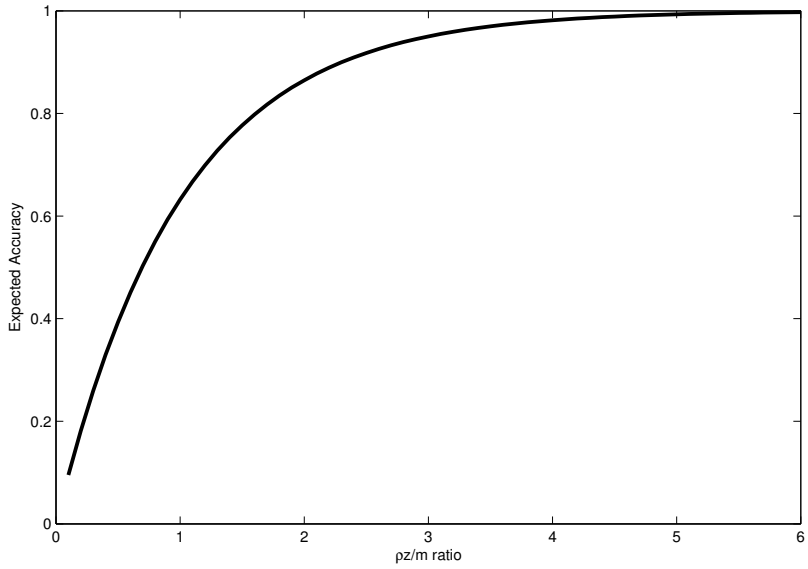

Figure 4.2: The probability of retrieving any given relevant document of PAC search for different ratios of the sample index to collection size, $\frac{\rho z}{m}$.

The expectation of k' is $E(k') = kP(d)$. Therefore, Equation (4.3) indicates that acceptable performance using PAC search can be achieved provided the probability, $P(d)$, is sufficiently high.

The performance of PAC search is determined by the size of the sample index, $z\rho$ in comparison with the collection size, m. Following the same setting as Google where ρ/m = 0.001, Figure 4.2 shows the expected accuracy of PAC search for various ratios of $\frac{\rho z}{m}$. Here we see that the accuracy rapidly increases from 63% when the sample index is equal to collection size to 86% when the sample index is twice as big as the collection, and to 95% when the sample index is three times as big as the collection. If the sample index is 5 times the size of the collection, then the accuracy is 99%.

The previous discussion of PAC search does not specify the presence of a central node that can coordinate the search, and a client device only randomly selects z computers to issue a query to. As discussed in Chapter 3 a non-deterministic search may be disconcerting to users, since the same query, issued more than once, is likely to retrieve different result sets, albeit with significant overlap. In Chapter 3, it was proposed to ameliorate this problem, by issuing the query to a set of z pseudo-randomly selected computers, where the pseudo-random function was seeded with the query. As a result, a user issuing the same query more than once is likely to see the same result set, since the set of pseudo-random computers most probably be the same for a given query.

If this pseudo-random selection is performed independently by each client/user, then different users will still see different results when issuing the same query, again albeit with significant overlap. To resolve this, we can construct a centralized non-deterministic PAC search architecture in which a centralized node, which may consists of one computer or a group of computers, receives queries from users, performs the pseudo-random selection centrally, and then forwards the query to the chosen set of z

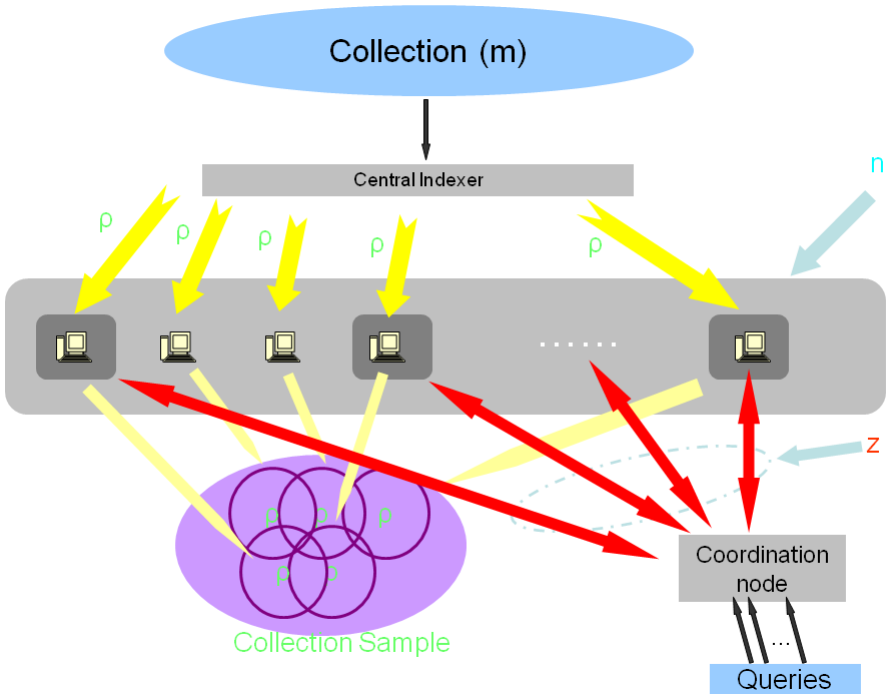

Figure 4.3: A centralized PAC search architecture.

machines. This is illustrated in Figure 4.3.

Of course, such a configuration reduces the fault tolerance of the system. Reliability is now a function of single point of failure, the coordination node. However, this is no worse than for deterministic centralized distributed architectures.

Given a centralized PAC search architecture, we now investigate how such an arrangement can be used to improve the correctness of PAC search for frequently issued queries.

4.2 Adaptive Node Selection

For frequently issued queries, we consider whether it is possible to improve upon our initial selection of z randomly selected computers in order to improve the correctness of the PAC search. The fundamental idea is the following. Given the first instance of a frequently occurring query, q, we select z pseudo-random computers. These computers form the sample index for this query. After receiving the results from the z machines, consolidating the results, and transmitting the retrieval results back to the user, we record the subset of computers that performed well, i.e. those that provided the largest number of relevant documents or the highest ranked documents. The choice and size of this subset will be discussed shortly. When we receive the same query a second time, the query is issued to this subset together with a new set of randomly chosen machines. Of course, the total size of these two sets of machines remains unchanged, i.e. z, to meet the limitation of computational resources. Once again, after receiving the results from the z machines, consolidating the results, and transmitting the retrieval results back to the user, we record the subset of computers that performed well, and the process repeats. At each query

iteration, only the identifiers of the retained computers, rather than the documents retrieved from these nodes, are cached. We refer to this as node caching, i.e. we are not caching queries in the traditional sense [FPSO06].

The purpose of query caching is to reduce the computational requirements of the information retrieval system by saving and caching the results of common searches. In contrast, the purpose of node caching is to iteratively improve the accuracy of the PAC search. There is no saving in computation. Of course, node and query caching could be combined, but this is outside the scope of this chapter.

After responding to the initial query, we have examined z computers, and the expected accuracy is given by Equation (4.3). After responding to the query a second time, we have examined more than z computers. From Equation (4.3) we know that if we had looked at all these computers simultaneously, then our expected accuracy would increase. This provides an upper limit on our performance. In practice, we can *not* examine all these computers simultaneously. Rather, at each iteration we can only examine z computers, this being the computational resource available to each query. Based on the adaptive pseudo-random selection of the sample index outlined above, can we design an algorithm that closely follows the upper limit provided by Equation (4.3)?

Before answering this, we consider a more fundamental question. What is the probability that there exists z computers from the set of all computers, n, such that, for a particular query, q, the accuracy of the z computers is 100%? That is to say, given a set of k documents retrieved deterministically in response to the query, q, what is the probability of these k documents existing on at least one configuration of z computers?

For a specific set of z computers, the expected number of unique documents in the sample index, $\hat{m}$, is, from Equation (4.1)

$$\hat{m} = (1-\epsilon)m = \left(1-(1-\rho/m)^{z}\right)m \tag{4.5}$$

The probability of finding a specific document within the sample index is simply $\frac{\hat{m}}{m}$. The probability, $P(k)$, of finding a specific set of k documents in the sample index is

$$P(k) = \prod_{\delta=0}^{k-1}\left(\frac{\hat{m}-\delta}{m-\delta}\right) \tag{4.6}$$

and the probability of all k documents not being in the sample index is

$$P(\bar{k}) = \left[1-\prod_{\delta=0}^{k-1}\left(\frac{\hat{m}-\delta}{m-\delta}\right)\right] \tag{4.7}$$

The probability of *not* finding the k documents in any sample of z computers is

$$P_{all}(\bar{k}) = \left[1 - \prod_{\delta=0}^{k-1} \left(\frac{\hat{m} - \delta}{m - \delta}\right)\right]^{\binom{n}{z}} \tag{4.8}$$

where $\binom{n}{z}$ is the number of ways of choosing z computers from n. Thus, the probability that all k documents will be present in at least one sample of z computers is

$$P = 1 - \left[1 - \prod_{\delta=0}^{k-1} \left(\frac{\hat{m} - \delta}{m - \delta}\right)\right]^{\binom{n}{z}} \tag{4.9}$$

Clearly the quantity in the square brackets is less than one, and this is raised to the power of $\binom{n}{z}$, which, for $n = 300,000$ and $z = 1,000$, is an extremely large number. Thus, there is near certainty that there exist configurations of z computers on which all k relevant documents (as defined by deterministic search) are present in the sample index.

4.2.1 Basic Model

To simplify the analysis of the problem, let us assume that the set of the relevant documents are known. We first study the number of relevant documents retrieved during the multiple occurrences of a single query. Each time we observe such a query, we wish to exploit knowledge learnt in previous searches to improve the retrieval accuracy of the query in the current iteration. To do so, we need to identify good performing nodes for this query and use these nodes in subsequent iterations. In general, there are three parameters that affect the accuracy. The first is the number of nodes, z, which this query is sent to. We assume that this is fixed based on computational and communication costs. In addition, at each interaction, the z_s best performing nodes across all previous iterations have been recorded, and z_q nodes chosen from z_s are queried together with $(z - z_q)$ random nodes. In the following analysis, we will consistently use the superscript i to denote some system parameters' values at ith query iteration, e.g. z_s^i indicates z_s's value at the ith iteration.

Let us first consider the distribution of relevant documents on a single machine. To simplify our discussion, we assume that for each query there are, on average, k ($k \leq \rho$) relevant documents. Since the probability of any document, including each relevant document, being present on a single computer is $P(d)$, assuming a uniform distribution, then the probability of k_s documents from k documents being

present on a single computer is

$$P(k_s) = \begin{pmatrix} k \\ k_s \end{pmatrix} P(d)^{k_s}(1 - P(d))^{k-k_s} \tag{4.10}$$

This is a standard binomial distribution,so the expected number of relevant documents on a single computer is

$$E(k_s) = rP(d) = \frac{\rho k}{m} \tag{4.11}$$

For a given PAC architectural configuration, Equation 4.11 allows us to compute the expected number of nodes that would index a certain number of relevant documents, k_s. For $m = 556000$ (close to the size of TREC disk 4/5), $\frac{\rho}{m} = 0.001$, i.e. each node index 0.1% of the index, and $k = 1000$, from Equation 4.10 we know that about 37% of queried computers contain no relevant documents, 37% are expected to contain 1 relevant document, 18% contain 2 relevant documents, 6% contain 3 relevant documents, and 2% of nodes contain 4 relevant documents.

The expected total number of relevant documents returned by all z computers is simply $zE(k_s)$, ignoring duplicate documents. To increase the number of relevant documents, and thereby improve the accuracy of PAC, it is necessary to increase $E(k_s)$. This cannot be done using random sampling, if z is fixed. Note that the likelihood that a node returns two or more relevant documents is relatively small. Consequently, if a node does return two or more relevant documents this node can be considered a "good" node for this particular query. If we wish to increase the expected number of relevant documents for the given query we could record such good nodes and direct subsequent repetitions of this query to these nodes. This is the basis for the iterative learning algorithm.

The ith time we see a query, we assume that we have recorded the z_s^i best performing nodes from the first through $(i-1)$th iteration. Note that this requires z_s^i units of storage for each unique query. Thus it is desirable that z_s^i be small. Note that determining which nodes are best is discussed later. At iteration i, we then choose to send the query to the z_q^i best nodes identified in the previous $(i-1)$ iterations, and to $(z - z_q^i)$ randomly chosen nodes. The value of z_q^i may vary from 0 to z_s^i, and controls the relative degree of exploration, $(z - z_q^i)$, and exploitation, z_q^i, at ith iteration. In the simplest case, z_s and z_q can be fixed across all iterations.

Let us now consider the first three iterations of the general algorithm. We assume that z_s is fixed to z ($z_s = z = 1000$) across iterations, so we have only one free parameter, z_q, which can take any integer value from 0 to z. Let k_z denote the total number of relevant documents contained by the z queried computers. Note that k_z^i may include repetitions, i.e. the same relevant document may be retrieved by more than one machine.

The first time the query is seen it is sent to z ($z_q^1 = 0 \Rightarrow z - z_q^1 = z$) random nodes, since we

have no prior information. Assuming that there are k relevant documents in the collection associated with this query, then Equation 4.10 is the probability, $p(k_s)$, of finding k_s relevant documents on a single node. Thus, k_z^1 is simply $zE(k_s) = 1000$ based on Equation 4.11. Note that the fact that k_z^1 equals the assumed total number of relevant documents, k, for the query, does not imply that we have found all relevant documents, since k_z^1 may contain repetitions. We will return to this shortly. When we record the z_s^1 best performing nodes, we know that the expected number of nodes returning k_s relevant documents is simply $z_s p(k_s)$.

Let us now assume that the query is at iteration $i = 2$, Based on our earlier example using a configuration where $\rho/m = 0.001$, and $z_s = 1000$ nodes, we know from Equation (4.10) that there 2% of nodes (total 20) contain 4 relevant documents, 6% of nodes (total 60) containing 3 relevant documents, 18% of nodes (total 180) contain 2 relevant documents, etc. Therefore, if $z_q^2 = 1$, the expected number of relevant documents on the best node is 4. And the number of relevant documents we can expect to acquire from the remaining $z - z_q^2 = 1000 - 1 = 999$ randomly selected nodes is 999 from Equation 4.11. So k_z^2 is $4 + 999 = 1003$. Similarly, for $z_q^2 = 500$, the expected number of relevant documents on the 500 best nodes is $20 \times 4 + 60 \times 3 + 180 \times 2 + 240 \times 1 = 860$ and the number of relevant documents acquired from the $z - z_q^2 = 1000 - 500 = 500$ randomly chosen nodes is 500. Hence, $k_z^2 = 860 + 500 = 1360$. Using this approach, we can, of course, calculate k_z^2 for all possible values of z_q^2 (there are only z_s such values) and find the values of all possible k_z^2.

Let us now assume that the query is at iteration $i = 3$. A difference here is that we have actually visited $z + z - z_q^2$ random selected nodes in the first 2 iterations. Let us assume $z_q^2 = 500$ here. So we have actually visited 1500 random selected nodes in the first two iterations. For these 1500 random nodes, we know that 2% of nodes (30) contain 4 relevant documents, 6% of nodes (90) containing 3 relevant documents, etc., based on Equation 4.10. As a result, in the recorded $z_s = 1000$ best performing nodes chosen from these 1500 random nodes, there are 30 nodes containing 4 relevant documents, 90 nodes containing 3 relevant documents, 270 nodes containing 2 relevant documents, 555 nodes containing 1 relevant document, and 55 nodes containing no relevant documents. Thus, if we select $z_q^3 = 500$, in the third iteration, the expected number of relevant documents on the 500 best nodes is $30 \times 4 + 90 \times 3 + 270 \times 2 + 110 \times 1 = 1040$. And the number of relevant documents we can expect to acquire from the remaining $z - z_q^3 = 1000 - 500 = 500$ randomly selected nodes is 500 from Equation 4.11. Then, the number of relevant documents we expect to acquire at iteration 3 is $k_z^3 = 1040 + 500 = 1540$. And for arbitrary z_q^2 and z_q^3 we can calculate k_z^3 in the same way.

Next, we explain how the retrieval accuracy can be predicted given our prediction of k_z. The example above shows how to theoretically calculate the expected number of relevant documents, (k_z), at each iteration. As noted earlier, the expected number of relevant document, k_z may contain repetitions. What we really need to know is the expected accuracy. Given k_z, we can use Equation (4.11 to determine

the equivalent number of random machines, z_e, necessary to acquire k_z relevant documents.

$$k_z = E(k_s)z_e = \frac{\rho k z}{m} \Rightarrow z_e = \frac{m k_z}{\rho k} \tag{4.12}$$

Substituting z_e into Equation 5.4, we can obtain the corresponding expected retrieval accuracy.

$$E(\text{RETRIEVAL ACCURACY}) = 1 - (1 - \rho/\mathrm{m})^{\mathrm{z_e}}\,, \tag{4.13}$$

Thus, we are now able to calculate the expected accuracy of a PAC search for any combination of z_s and z_q. For a given combination of z_s and z_q values, the expected retrieval accuracy at each iteration i can be determined as follows:

1. For the $1st$ iteration, the query is sent to z randomly selected nodes, i.e. $z_q^1 = 0$. The retrieval accuracy can be directly obtained from Equation 5.4.

2. For the ith ($i \geq 2$) iteration:

   (a) Calculate the total number of random nodes visited up to iteration $i-1$, and then use Equation (4.10) to obtain the expected number of relevant documents for best performing z_q^i nodes.

   (b) use equation 4.11 to calculate the expected number of relevant documents to be found in the remaining $z - z_q^i$ random selected nodes

   (c) sum the results to obtain the total expected number of relevant documents k_z^i, acquired at iteration i. Use Equation (4.12) to obtain equivalent searched nodes z_e, and substitute z_e into Equation (4.13) to obtain the expected retrieval accuracy for this iteration.

The preceding analysis can be used to guide an heuristic search to find a z-configuration that includes as many relevant documents as possible for a specific query, and thereby significantly improves retrieval accuracy. That is, assuming $z_s = z_q$, at iteration i, each of our z computers has retrieved k_j relevant documents, where $1 \leq j \leq z$. We order the computers based on the number of relevant documents retrieved, k_j. For simplicity, and without loss of generality, we assume that $k_1 \geq k_2 \cdots \geq k_z$. Then, we only keep the best performing z_q computers, i.e. computers correspond to $\{k_1, k_2, ..., k_{z_q}\}$. These z_q computers will again be used in the subsequent iteration, and the process repeats.

To validate this model, we compared the predicted performance with a simulation. The parameters of the simulation were $m = 556000$, $\frac{\rho}{m} = 0.001$, $n = 300,000$, $k = 1000$, $z_s = z = 1000$ and $z_q = 500$, i.e. z_s and z_q are fixed throughout query iterations. In the simulation the relevant documents are known, and this is used to select the best 500 nodes at each iteration. The case where the relevant documents are unknown is addressed in Section 4.2.2. The simulation results are averaged over 10

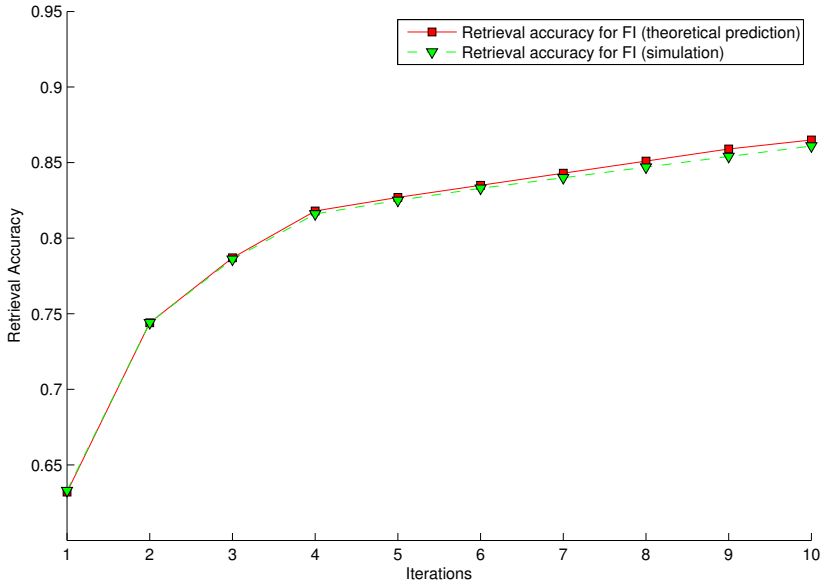

Figure 4.4: Comparison of retrieval accuracies obtained by theoretical prediction and simulation respectively, PAC settings, where $m = 556000$, $\frac{\rho}{m} = 0.001$, $n = 300,000$, $k = 1000$, $z_s = z = 1000$ and $z_q = 500$.

trials. Figure 4.4 shows the retrieval accuracy as a function of iteration, i. There is very close agreement between the theoretical and simulated results.

Of course, in practice the relevant documents may be unknown, and the algorithm is modified in Section 4.2.2 to account for this.

4.2.2 Relevant Documents Unknown

The assumption of the iterative method proposed in Section 4.2.1 is that the set of relevant documents is known beforehand. However, this may not be realistic in a real retrieval environment.

In PAC, we assume that the top ranked k documents of a deterministic system, based on a retrieval model such as the BM25 model [RWJ+96], are relevant to the query. Indeed, at each iteration, our z computers produce a merged ranked list of documents, where each of our z computers has retrieved k_j documents, $1 \leq j \leq z$, in the top k rank positions. Intuitively, we favor a computer with many documents ranked highly in the top k positions.

As before, we can simply rank the z computers based on the number of documents each computer has in the top k positions, k_j. However, this measure gives equal weight to documents at the top and bottom of the result set. In order to also consider the rank of documents, we propose an NDCG-like score [JK02] for judging how well each computer is responding to a query. For computer, j, its score, s_j is defined as

$$s_j = \sum_{r=1,\ldots,k} \frac{\delta_r}{log_2(1+r)} \tag{4.14}$$

where $\delta_r = 1$ is an indicator variable that is one if the document at rank position r is one of the documents on computer j, and 0 otherwise.

We set k=1,000 in our TREC data based experiments in Section 4.3.1, and the results show that our NDCG-like score performs better than simply counting the number of documents in the top rank positions.

Once again, we retain the top z_q of the ordered list of computers in each query iteration, and use the retained computers in subsequent query iterations.

4.3 Experimental Results

In order to investigate the effectiveness of our two methods we performed both simulations and experiments based on a TREC dataset.

In our experiments, we used the same computation and storage requirements as assumed of Google. There are n=300,000 computers which independently sample ρ documents. We kept the ratio between the number of documents indexed by each computer, ρ, to m as 0.001, and fix the number of computers queried at each step $z = 1,000$.

We fixed the initial $z_q = 200$ in the first iteration. This is incremented by 30 in each subsequent iteration, in order to gradually increase the weight of knowledge exploitation. A less heuristic pruning strategy is a topic of future work.

4.3.1 TREC Experimental Results

In a real retrieval environment, the number of relevant documents is unlikely to be known. Thus we need to evaluate our approach based on document retrieval models. Our proposed approach in Section 4.2.2 allows us to carry out real world retrieval.

We tested our approach using the TREC-8 50 topics, using only the title part. The dataset consists of approximately half a million documents. We used the same settings as previous, i.e., n=300,000, and $\frac{\rho}{m}$=0.001. We used the BM25 retrieval model [RWJ$^+$96] for ranking documents on each computer. In order to eliminate any unreliability from a single trial, we ran 10 trails, and in each trial, we performed 10 iterations for each query. In the first iteration, each query is issued to 1,000 randomly chosen computers. Then each computer's performance is evaluated based on the metric proposed in Section 4.2.2. At each iteration we query 1,000 computers, a portion retained from the previous iterations together with a randomly selected set of computers. We used standard IR metrics including MAP (mean average precision) and Recall at 1,000 to evaluate the performance at each iteration. The MAP and Recall values

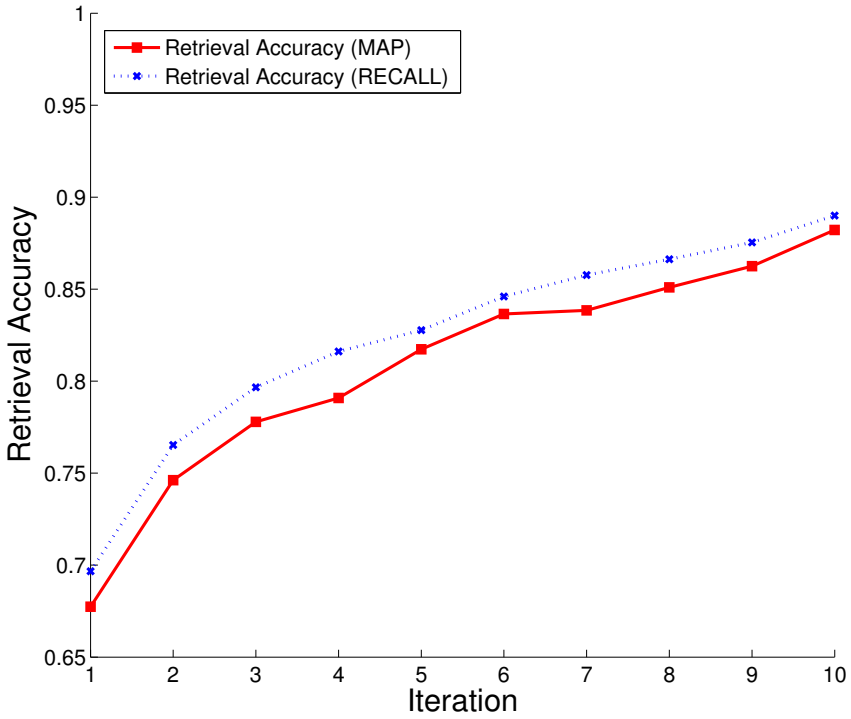

Figure 4.5: Retrieval accuracy of our iterative approach on MAP and Recall-1000. Results are based on the TREC8 50 queries, when nodes are pruned based on the number of documents they retrieve in the ranked list.

at each iteration are averaged over the 10 trials and the result is reported here. To compare our iterative approach with a deterministic system, following the definition of the correctness of the PAC search, we define the retrieval accuracy for MAP and recall as:

$$RetrievalAccuracy(Metric) = \frac{Metric_{PAC}}{Metric_{det}}, \tag{4.15}$$

where $Metric_{PAC}$ is the MAP or Recall-1000 of the PAC system, and $Metric_{det}$ is the MAP or Recall-1000 of the deterministic system.

Figure 4.5 and 4.6 shows the performance of the system when pruning is based on (a) simple counting and (b) an nDCG-based metric. It is clear that the nDCG-like metric outperforms the simple counting. The retrieval accuracy when pruning based on simple counting increases from 67% to 81% in the first 5 iterations. In comparison, the retrieval accuracy using the nDCG-like metric increases from 67% to 92% in the first 5 iterations. The retrieval accuracy when pruning based on the simple counting method only reaches 88% at iteration 10, while the retrieval accuracy of the nDCG-like metric reaches 96% at iteration 10.

Figure 4.6, also shows that both MAP and Recall-1000 follow a similar trend, i.e., performance

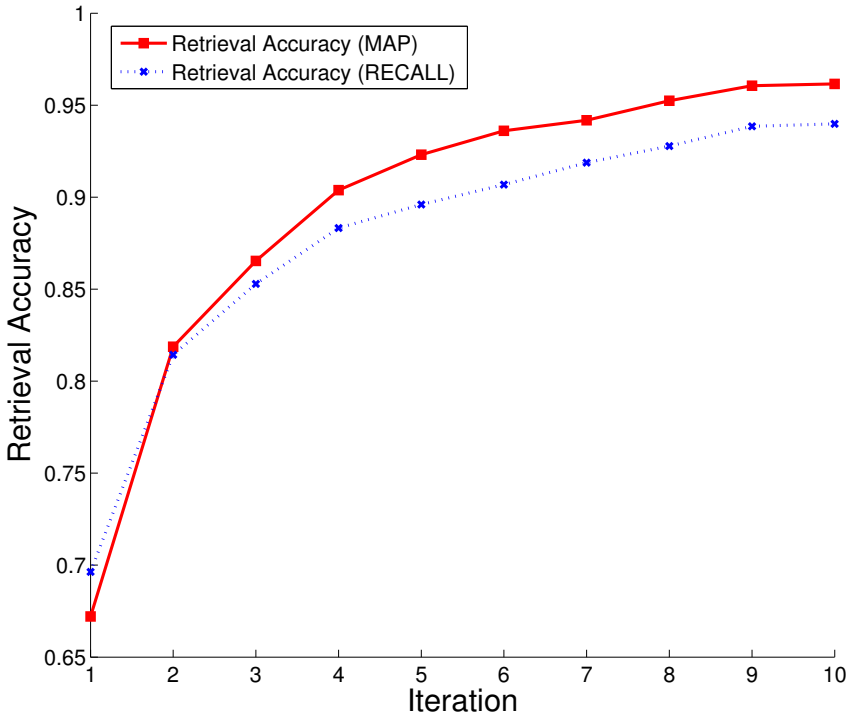

Figure 4.6: Retrieval accuracy of our iterative approach on MAP and Recall-1000. Results are based on the TREC8 50 queries, when nodes are pruned based on the NDCG-like metric.

improves quickly from iteration 1 to 5, and then improves more slowly from iteration 5 to 10. It is worth noting that only 4 iterations are required for our approach to improve from around 67% to 90% of the performance of a deterministic approach. The final retrieval accuracy reaches 96% at iteration 10. Note that the the standard deviations across trials are very small, and do not affect our observations here. This indicates that our approach has stable performance.

These experiments suggest that, for frequently occurring queries, a centralized PAC architecture can perform at a similar level as a deterministic system.

4.4 Summary and Future Work

The non-deterministic probably approximately correct search architecture is an interesting alternative search architecture. However, the performance of the PAC architecture must be improved if it is to become competitive with traditional deterministic systems.

In this chapter, we proposed adding a centralized query coordination node to the architecture. This configuration is not as distributed as the original PAC proposal. However, it retains much of its benefits, at the expense of introducing a single point of failure. Of course, in practice such a node could itself be replicated for fault tolerance and load balancing.

Using a centralized PAC architecture, and in response to a query, the random selection of nodes is replaced by a pseudo-random selection, seeded with the query. This has the advantage, as noted in

chapter 3 of eliminating the variation in results sets in response to the same query. More importantly, for frequently occurring queries, we considered the problem of iteratively refining the pseudo-random choice of z computers in order to improve the performance.

A theoretical analysis provided a proof that there exists (with near certainty) a configuration of z nodes on which all relevant documents will be found. The analysis also provided an upper bound on the performance of any iterative algorithm. The analysis also allowed us to estimate the expected number of relevant documents that should be present on any single machine. This information was then used to partly guide a heuristic search.

Two heuristic search algorithms were proposed. The first scored computers based simply on the number of relevant documents they retrieved. The second scored computers based on a nDCG-like score that gives higher weight to higher ranked documents.

Simulations showed that the search algorithm very closely followed the theoretical upper bound when the number of relevant documents was less than $k = 1000$. However, as the number increased to 2000 or 4000, deviation from the upper bound increased. Nevertheless, in all cases, retrieval performance continued to improve with each iteration. For $k = 4000$, the case where twice as many machines ($z = 2000$) were queried per iteration was examined. In this case, the initial accuracy is 86% and the search once again closely follows the theoretical upper bound. This suggests that when $z = 1000$, the iterative algorithm is retaining a sub-optimal choice of machines. Improvements to the heuristic search algorithms remains a source of on-going research.

Experiments on the TREC-8 dataset showed that the nDCG-based algorithm provided superior performance. In particular, the MAP score relative to a deterministic system increased from an initial 67% to 90% in just 4 iterations. And after 10 iterations performance is 96% of a deterministic system. These experiments suggests that, for frequently occurring queries, a centralized PAC architecture can perform at a similar level as a deterministic system.

This iterative approach is only applicable for queries that occur frequently (e.g. more than 5 times). It is reported [BYGJ$^+$07] that real query logs often follow a power law distribution with a long tail, e.g. 44% are singleton queries out of the whole query volume. We could alternatively increase the search size for those less frequently occurring queries to improve their performance. The impact of unpopular queries to the average performance of a centralized PAC architecture, and alternative approaches to improve the performance of unpopular queries, are subjects of future work.

Chapter 5

Document Replication Policies

In this chapter, we consider different documents replication policies for a PAC search system that is deployed in a peer-to-peer (P2P) retrieval environment. We have so far discussed the PAC architecture based on the uniform sampling/replication of documents in the previous chapters. However, it might be more useful to replicate documents non-uniformly under a decentralized, peer-to-peer retrieval setting.

Prior work in peer-to-peer search has considered the expected search size, i.e. the number of nodes that must be queried to find a particular document, as a function of the document replication policy. It has been shown that replicating documents across nodes based on the square root of the query frequency is optimum in the sense of minimizing the expected search size. While the expected search size is a useful characterization, in practice, it is likely that search size is constrained to an upper limit. Moreover, prior work implicitly assumed a one-to-one correspondence between a query and a document. However, for general information retrieval, many users are interested in a *set* of retrieved documents, i.e. the top-k documents.

In this chapter, we analyze a variety of replication policies based on the accuracy measure introduced in Chapter 3. Note this accuracy measure is based on the intersection of the number of documents retrieved by a constrained, probabilistic search and the number of documents that would have been retrieved by an exhaustive search, and is independent of any retrieval model. We provide equations for the probability of achieving a particular accuracy for different replication policies. We then show that (i) the square root replication policy is not optimum for maximizing accuracy, (ii) give an optimality criterion for accuracy, and (iii) provide an approximate optimal solution that maximizes accuracy. Our proposed optimal replication policy achieves better performance than the square root replication policy for the same implementation complexity.

A uniform distribution of documents ensures that all queries perform, on average, equally well. All non-uniform replication policies improve the performance of popular queries at the expense of less popular queries. Unfortunately, given the typical power law distribution of queries [BYGJ$^+$07], a very significant volume of queries have worse performance with a non-uniform replication policy. To alle-

viate this, we further propose a hybrid replication policy that combines both uniform and non-uniform distributions. Theoretical results show that such an arrangement significantly improves the accuracy of less popular queries at the expense of only a small degradation in the accuracy of popular queries.

The rest of this Chapter is organized in the following way. Section 5.1 reviews Cohen and Shenker's replication model in detail. Then, section 5.2 presents PAC under different replication policies, in comparison with Cohen and Shenker's model. Section 5.3 proposes a hybrid replication solution which balances the performance between popular and unpopular queries. Section 5.4 provides experimental results on a simulated network of 10,000 nodes and about 1.7 million documents that supports for our models and analysis. Finally, Section 5.5 provides a summary and discussion of future work.

5.1 Cohen and Shenker's Model

In this section we review Cohen and Shenker's work [CS02] on documents replication policies in detail. Cohen and Shenker provided both a theoretical and empirical analysis of different replication policies. Given an unstructured P2P network of size n, each node has capacity ρ, i.e. ρ is the maximum number of files that can be stored on each node. Let $R = n\rho$ denote the total capacity of the system. It is also assumed that m unique files are stored in the P2P system and each file i is replicated on r_i different nodes. Clearly, $\sum_i r_i = R$, and $mr = R$ if $r_i \equiv r$ is a constant. Let $p_i = \frac{r_i}{R}$, be the fraction of the total system capacity allocated to file i, and q_i be the normalized query popularity for the ith file, i.e. $\sum_{i=1}^{m} q_i = 1$

The search size, z_i, of file i, is defined as the number of nodes searched in response to a query to find file i. As shown in [CS02], the search size is a random variable drawn from a geometric distribution, and the average search size for file i is $E(z_i) = \frac{n}{r_i}$ and the expected search size, $E(z)$, of all m files is

$$E(z) = \sum_i q_i \times \mu_z(i) = n \sum_i \frac{q_i}{r_i} \tag{5.1}$$

For a uniform replication policy, $r_i \equiv r$ is a constant and $mr = R$. Thus, the expected search size with uniform replication, $E(z^u)$, is $E(z^u) = \frac{n}{r} \sum_i q_i = \frac{m}{\rho}$

Two alternatives to a uniform replication policy, namely a proportional replication policy and a square root replication policy, are also considered. A proportional replication policy replicates a file based on its popularity, i.e. proportional to the number of queries requesting it. Such a replication policy results in the same expected search size as the uniform replication, i.e. $\frac{m}{\rho}$. In effect, while popular documents will be found querying fewer nodes than for a uniform replication policy, this is balanced by the need to visit far more nodes in order to find unpopular documents.

A square root replication policy is derived from minimizing the expected search size, given in Equation (5.1). Solving this optimization problem [CS02], results in $r_i = \frac{R}{\sum_i \sqrt{q_i}} \sqrt{q_i}$. This is the square

root replication policy which produces the optimal expected search size given by

$$E(z^s) = n \sum_i \frac{q_i}{\lambda \sqrt{q_i}} = \frac{1}{\rho} (\sum_i \sqrt{q_i})^2 \tag{5.2}$$

Note when the query distribution is uniform Equation 5.2 reduces to the uniform case. In practice, the query distribution typically follows a power law, i.e. $q_i = \frac{1}{c} i^{-\alpha}$ where c is a normalization constant. We note that analysis of search logs suggest a value of α between 0.7 and 1.5 [aol, BYGJ+07]. Note, however, that while the expected number of nodes we need to search in order to find an item is examined, Cohen and Shenker do not thoroughly study the probability of finding a document for a fixed length search.

5.2 Documents Replication Policies in PAC

The previous work on probabilistic search studied the expected average search size to retrieve a specific document under different document replication policies. In practice, it is likely that an upper limit on the search size is imposed by bandwidth and latency constraints. Thus, another interesting question to ask is, how much relevant document can we find, given a query can only be sent to a constant number of nodes, under different documents replication policies? This question can be addressed within the PAC search architecture.

5.2.1 Uniform Replication

We first explain the original PAC architecture proposed in Chapter 3 under a peer-to-peer setting, and clarify the connections between PAC and Cohen and Shenker's model. In a PAC search architecture, a network of n nodes totally hosts m unique documents, and each node can at most store ρ documents. A query is only sent to z random nodes in the search stage, and the probability of finding a specific document d_i is given by

$$P(d_i) = 1 - (1 - \frac{\rho}{m})^z$$

In information retrieval, we are often interested in the top-k documents for a query. In this case, the correctness of a PAC search is measured by *retrieval accuracy*. Let $\mathcal{D}$ denote the set of top-k documents retrieved from the full index, i.e. an exhaustive search, and $\mathcal{D}'$ denote the set of top-k documents retrieved when querying z nodes, i.e. partial index. The retrieval accuracy, a, is defined as

$$a = \frac{|\mathcal{D} \cap \mathcal{D}'|}{|\mathcal{D}|} = \frac{k'}{k}$$

where k' denote the size of the overlap of the two sets. Note that this definition of correctness is independent of the underlying retrieval model, unlike traditional information retrieval metrics such as precision and recall. Thus, this definition of accuracy is valid irrespective of which retrieval model is implemented

on the P2P network.

As shown in Chapter 3, the size of the overlap, k', is a random variable drawn from a binomial distribution, and is given by

$$P(k') = \binom{k}{k'} P(d_i)^{k'} (1 - P(d_i))^{k-k'} \tag{5.3}$$

Thus, the expected value of k' is $kP(d_i)$ and the expected retrieval accuracy $E(a)$ is

$$E(a) = \frac{E(k')}{k} = \frac{k \times P(d_i)}{k} = 1 - \left(1 - \frac{\rho}{m}\right)^z, \tag{5.4}$$

If we assume that a document is, on average, replicated onto r random nodes, then the total capacity of the network, R, satisfies $R = m \times r$, and Equation 5.4 becomes

$$E(a) = 1 - \left(1 - \frac{\rho}{m}\right)^z = 1 - \left(1 - \frac{r}{n}\right)^z \tag{5.5}$$

Note that the expected accuracy is independent of the size, k, of the original (exhaustive) result set.

5.2.2 Non-uniform Replication

The original work on PAC, described above, assumed a uniform replication policy for documents. Here, we extend the probabilistic analysis to the cases where the documents are replicated (i) in proportion to their popularity, (ii) in proportion to the square root of their popularity, as discussed in [CS02], and (iii) optimally to maximize accuracy.

In general, given a query distribution, we are interested in what replication policy can yield the highest global average retrieval accuracy for all queries. Let $\mathcal{Q}$ denote the set of all queries, and let q_j denote the query rate for query j, such that $\sum q_j = 1$. The replication level for document i is r_i. Thus, from Equation 5.5, the local expected retrieval accuracy, i.e. the expected probability of retrieving document i is

$$E(a_i) = P(d_i) = 1 - \left(1 - \frac{r_i}{n}\right)^z \tag{5.6}$$

Let $\mathcal{D}_k(j)$ denote the set of top-k documents retrieved for a query q_j. Thus, the global average retrieval accuracy for the top k documents, A_k, averaged over all queries is given by

$$\begin{aligned} A_k &= \sum_j q_j E(a_j) \\ &= \sum_j q_j \frac{\sum_{d_i \in \mathcal{D}_k(j)} (1 - (1 - \frac{r_j(i)}{n})^z)}{k} \end{aligned} \tag{5.7}$$

5.2.2.1 Retrieving the Top-1 Document

Now consider the case where we are only interested in the top ranked document, i.e. top-1 ($k = 1$), as in [CS02]. Thus, from Equation (5.7) we have

$$A_1 = \sum q_j(1 - (1 - \frac{r_j}{n})^z) \tag{5.8}$$

For a proportional replication policy, where $r_j = Rq_j$, the expected accuracy A_1^p is then given by

$$A_1^p = \sum q_j(1 - (1 - \rho q_j)^z) \tag{5.9}$$

and for square root replication policy, where $r_j = R\frac{\sqrt{q_j}}{\sum \sqrt{q_j}}$, the expected accuracy A_1^s is given by

$$A_1^s = \sum q_j(1 - (1 - \rho\frac{\sqrt{q_j}}{\sum \sqrt{q_j}})^z) \tag{5.10}$$

Neither the proportional or square root policies are guaranteed to maximize the expected accuracy, A_1 in Equation (5.8). The optimum distribution can be determined based on a convex optimization method, the mathematical details of which can be found in the Appendix. The optimal accuracy, A_1^o, is given by

$$\forall r_j \in (1, n), 1 \leq j < b' \leq m + 1, n - r_j \propto {q_j}^{-\frac{1}{z-1}}$$
$$\Rightarrow r_j = n - \frac{n(b'-1)-R'}{\sum_1^{b'-1} {q_j}^{-\frac{1}{z-1}}} {q_j}^{-\frac{1}{z-1}}$$

where $R' = R - m + b' - 1$, and b' is an auxiliary parameter to enforce the constraint $1 \leq r_i \leq n$. See Appendix C for more detail.

To examine the effect of the different replication policies, we considered a PAC configuration in which it is assumed that there are 1 million documents in the collection ($m = 10^6$), and 10,000 nodes in the network ($n = 10^4$). Each node is able to index 1000 documents ($\rho = 1000$). Thus, the total capacity of the network is $R = 10^7$ documents. To simplify the problem, we also assume that only top-1 document for each query is of concern, and top-1 document of different queries do not overlap. Thus, we guarantee a unique one-to-one between a query and a corresponding document, as in [CS02].

Figure 5.1 shows the expected accuracy, A_1, when retrieving the top-1 document as a function of the power law exponent for different replication policies. Here, we have assumed that the query distribution follows a power law, and have fixed the search size to 1000 nodes, ($z = 1000$). The uniform replication policy is independent of the query distribution. For all other replication policies, as the query distribution moves from uniform, i.e. $\alpha > 0$, we observe a very rapid improvement in accuracy, that monotonically increases with α. The optimal replication policy performs better than both the square

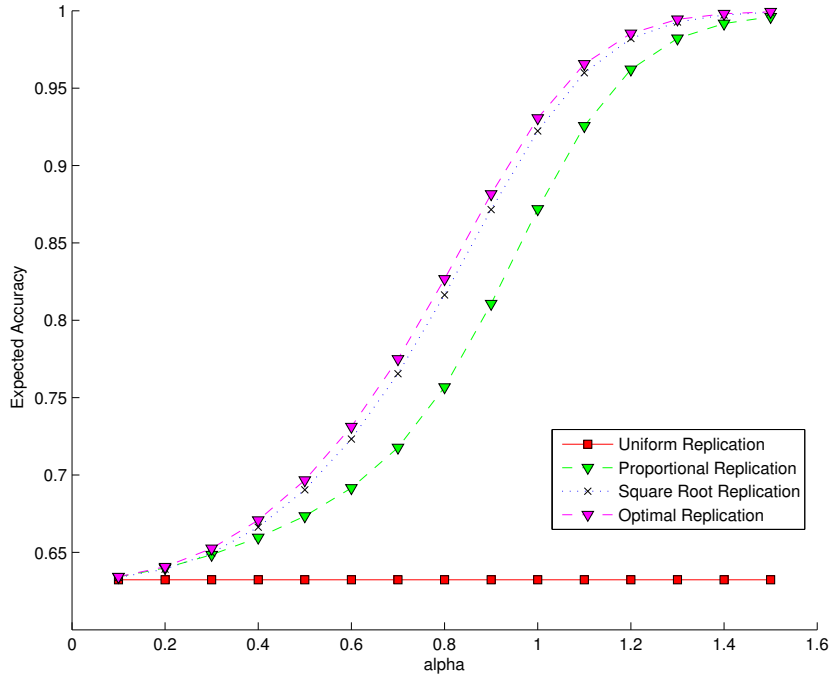

Figure 5.1: Expected accuracy for retrieving the top-1 document, A_1, as a function of the power law exponent, α, for different replication policies, for a fixed search size $z = 1000$.

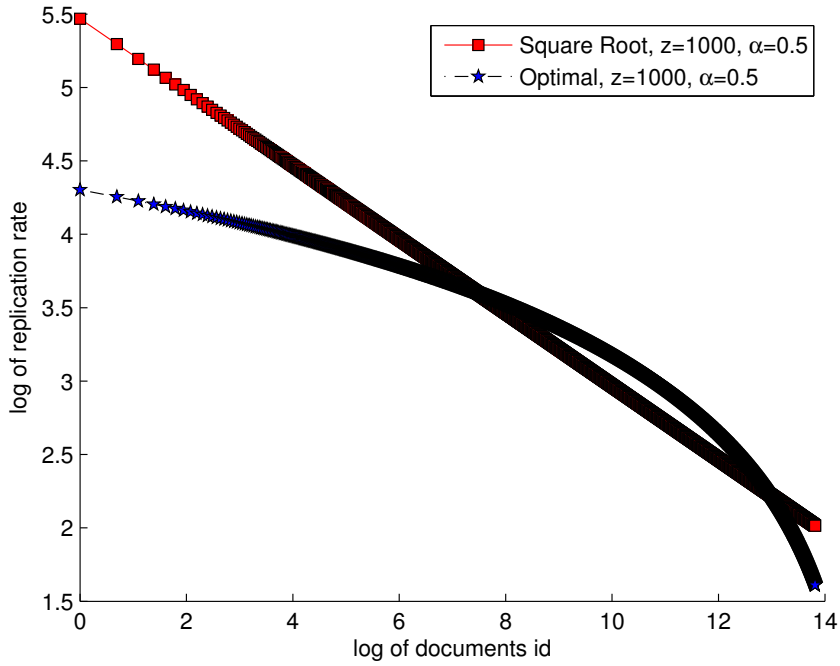

Figure 5.2: Replication rates of all documents in the collection, for square root and optimal replication policies, where $\alpha = 0.5$.

root and the proportional replication policy, but the difference between the optimal and the square root is small.

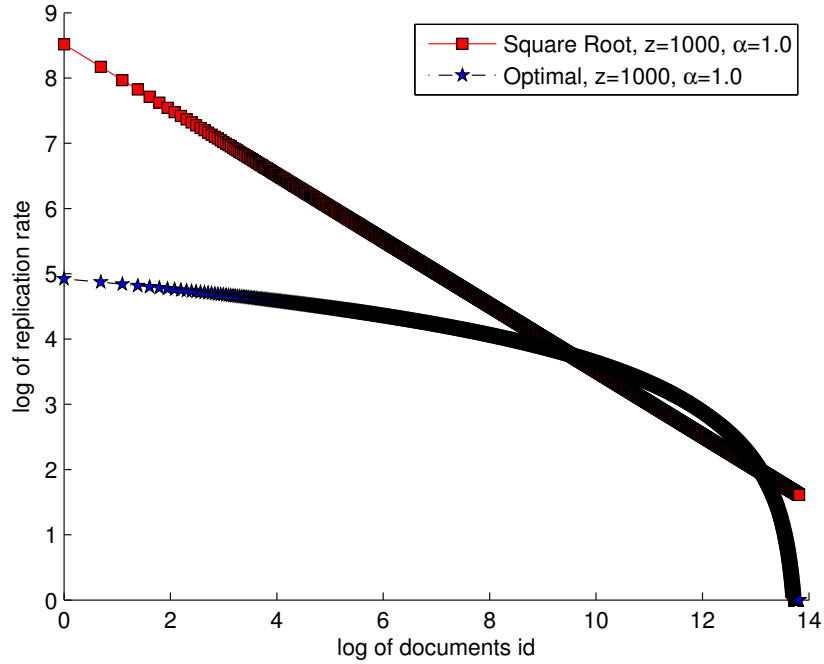

Figure 5.3: Replication rates of all documents in the collection, for square root and optimal replication policies, where $\alpha = 1.0$.

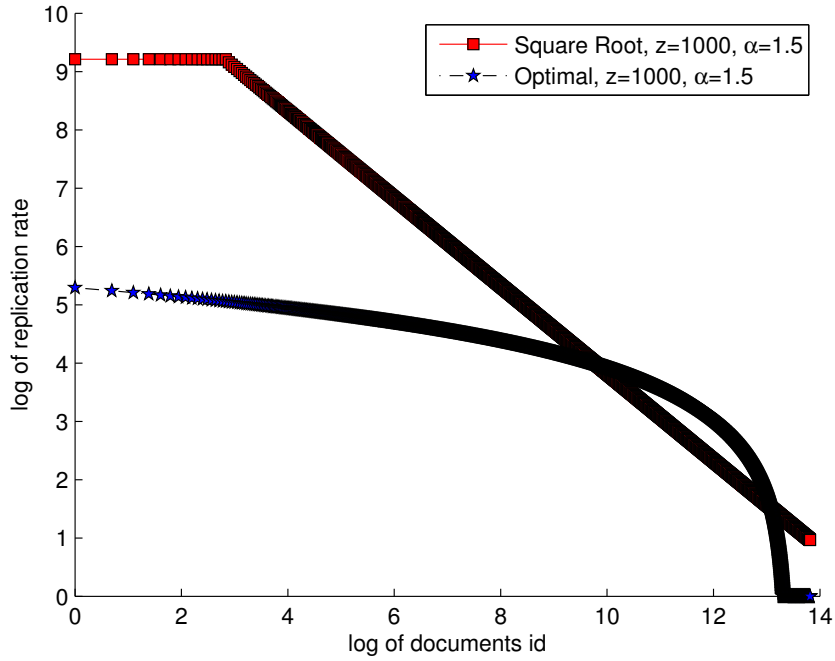

Figure 5.4: Replication rates of all documents in the collection, for square root and optimal replication policies, where $\alpha = 1.5$.

Figures 5.2 - 5.4 show the replication allocation when using the square root and optimal replication policies, when querying a fixed number of nodes, $z = 1000$. As α increases, we observe an increased difference between the two policies. In particular, the optimum replication policy does not replicate the

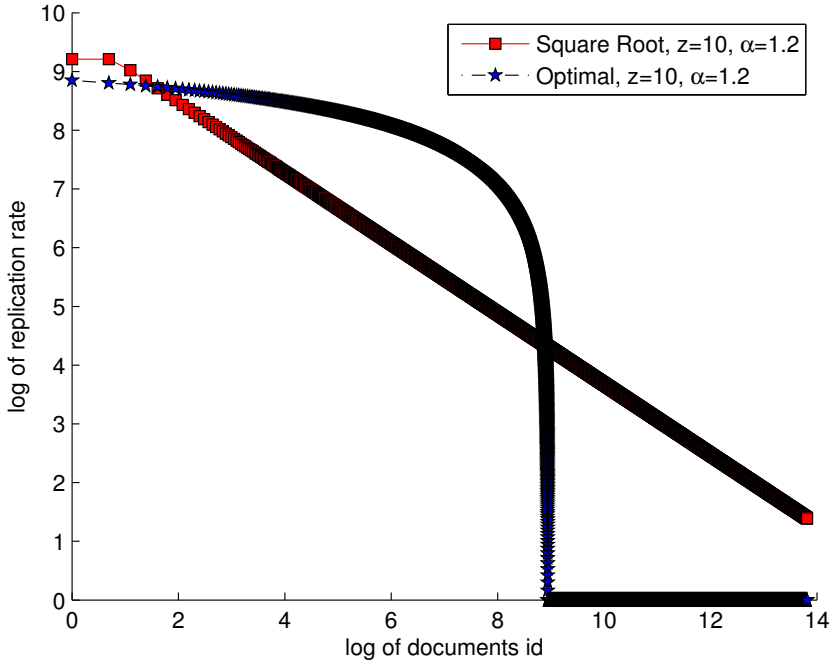

Figure 5.5: Replication rates of all documents in the collection, for square root and optimal replication policies, where $z = 10$.

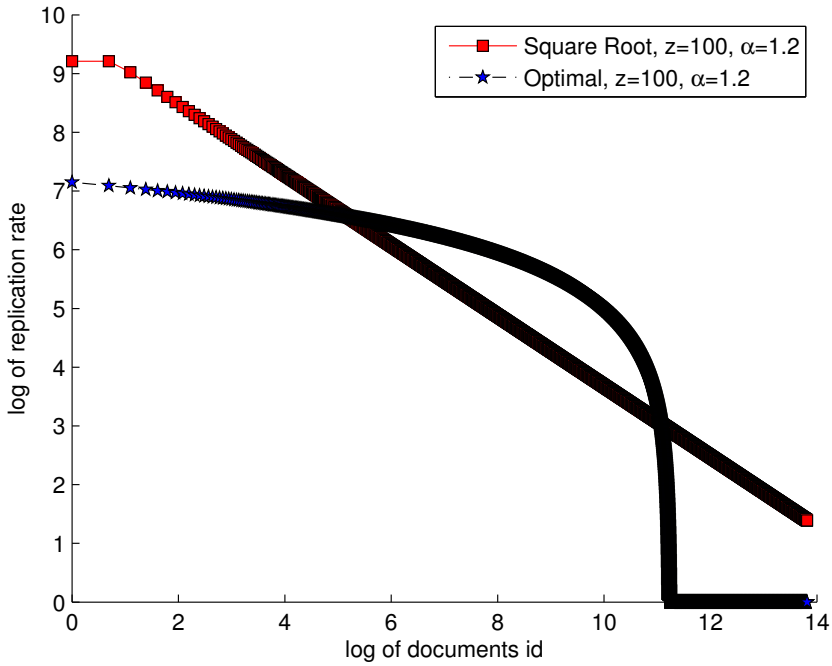

Figure 5.6: Replication rates of all documents in the collection, for square root and optimal replication policies, where $z = 100$.

popular documents as much as the square root policy. However, despite the substantial difference in the replication of documents, the difference in the resulting accuracy is small, as shown in Figure 5.1.

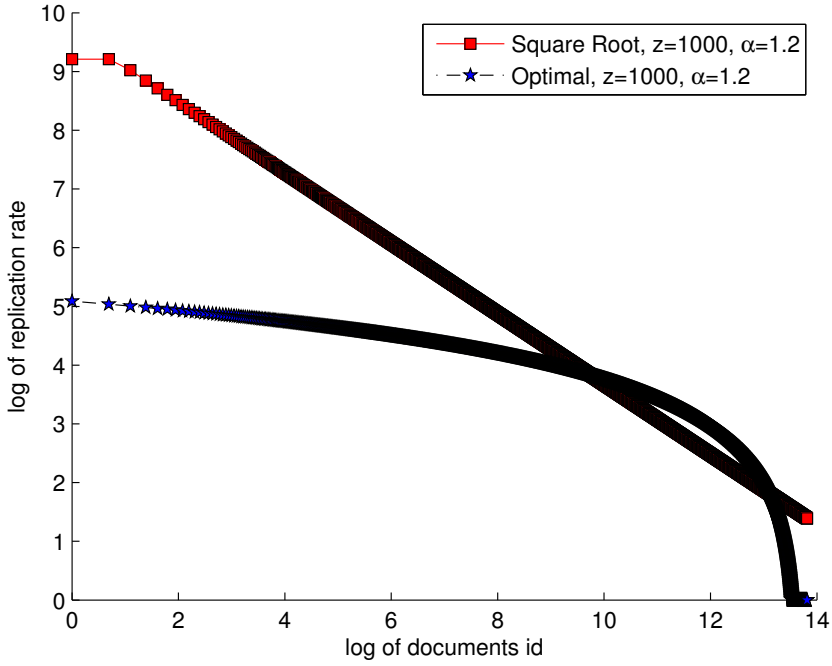

Figure 5.7: Replication rates of all documents in the collection, for square root and optimal replication policies, where $z = 1000$.

However, the difference in performance is more pronounced when the search size is small, as discussed shortly.

Figures 5.5 - 5.7 compare the replication allocation between the square root and optimal policies for various search sizes $z = 100, 500, 1000$, for a fixed query distribution, $\alpha = 1.2$. Interestingly, when the search size is small, $z = 10$, we observe in Figure 5.5 that the optimum policy actually replicates popular documents, but not the most popular documents, more that the square root policy. As the search size increases, we see a trend in which the optimum policy replicates the most popular documents less frequently than the square root policy, but replicates popular document more frequently, and the least popular less frequently.

Figure 5.11 shows the expected accuracy, A_1, when retrieving the top-1 document, as a function of the search size, z, for different replication policies, and for an fixed scaling exponent $\alpha = 1$. When the search size is small, the differences between the three replication policies are greater. Note that the proportional replication actually performs better than the square root for search sizes of less than $z = 300$. However, as the search size increases, the expected accuracy for the proportional policy increases more slowly and the square root policy is then superior. As expected, the optimal policy performs best for all search sizes.

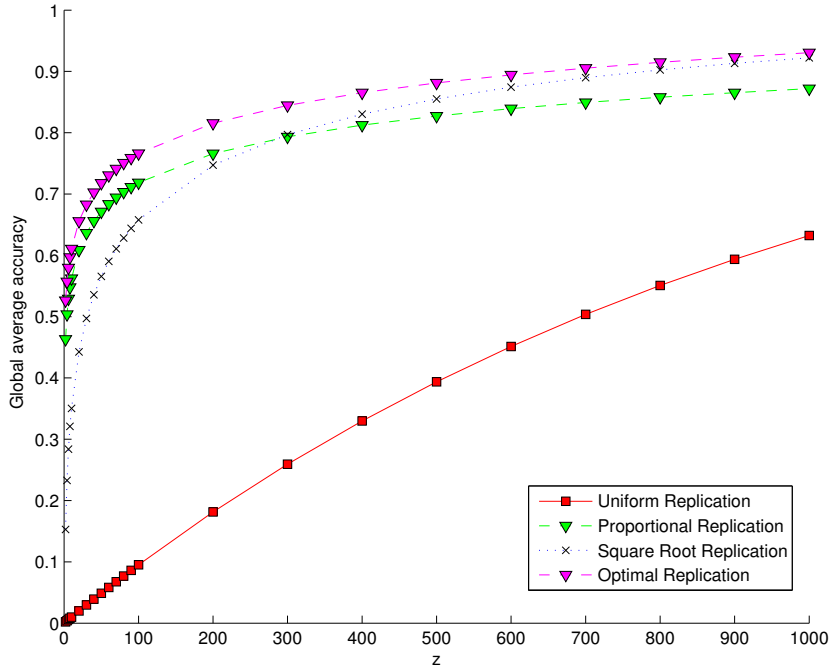

Figure 5.8: Expected accuracy for retrieving the top-1 document, A_1, as a function of the search size, z, for different replication policies, where $\alpha = 1$.

5.2.2.2 Retrieving Top-k Document

We now extend our analysis to the case where we are interested in the top-k retrieved documents, rather than only the top-1. Note that there is a difference between PAC and the model studied by Cohen and Shenker [CS02]. Cohen and Shenker assumed that each query is issued towards an object stored in the network, thus the total number of queries equals the number of documents m. However, in PAC's information retrieval perspective, there is no such unique one-to-one mapping between queries and documents. Theoretically, an infinite number of queries can be issued by users, and many queries can retrieve the same documents. However, to simplify our analysis of expected accuracy we assume a finite number of queries, $|Q|$, which is true for a finite period of time. The next issue is that the top-k documents retrieved by each query j, $\mathcal{D}_k(j)$, are likely to be non-disjoint, i.e. two queries might retrieve some documents in common. This highlights the fact that replication should be based on the distribution of the retrieval frequencies of documents, rather than the query distribution directly.

The intuition to solve the top-k retrieval problem is to convert the top-k retrieval problem back to the top-1 retrieval problem. Let us define an auxiliary frequency set $\mathcal{Q}'$ which holds the access frequency for each document in the collection, thus, for each $q'_i \in \mathcal{Q}'$, we have

$$q'_i = \frac{1}{k} \sum_{j=1}^{|\mathcal{Q}|} q_j \zeta(j,i) \tag{5.11}$$

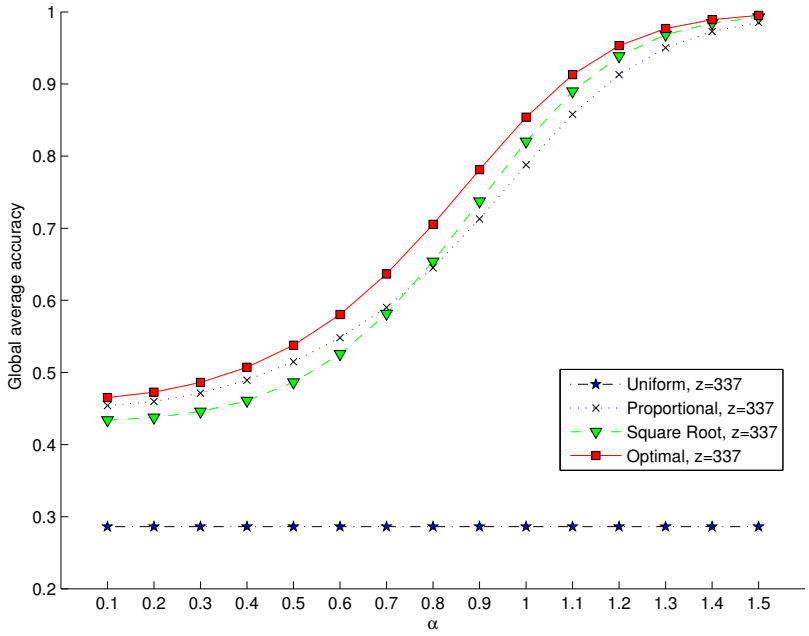

Figure 5.9: The global average accuracy for retrieving the top-10 document, A_{10}, as a function of the power law exponent, α, for different replication policies, for a fixed search size $z = 337$.

where

$$\zeta(j,i) = \begin{cases} 1 & \text{if document } i \text{ is in query } j\text{'s top-}k \text{ result list.} \\ 0 & \text{otherwise.} \end{cases} \tag{5.12}$$

Hence, we convert the top-k retrieval to the top-1 retrieval by transforming the Equation (5.7) to

$$\begin{aligned} A_k &= \sum q_j \frac{\sum_i (1-(1-\frac{r_j(i)}{n})^z)\zeta(j,i)}{k} \\ &= \sum_i (1-(1-\frac{r_i}{n})^z)\frac{1}{k}\sum_j q_j\zeta(j,i) \\ &= \sum_i q'_i(1-(1-\frac{r_i}{n})^z) \end{aligned} \tag{5.13}$$

where r_i is computed based on q'_i accordingly.

To examine the effect of the different replication policies, we considered a configuration in which it is assumed that there are 1 million documents in the collection ($m = 10^6$), and 10,000 nodes in the network ($n = 10^4$). Each node is able to index 1000 documents ($\rho = 1000$). Thus, the total capacity of the network is $R = 10^7$ documents.

Figure 5.9 shows the global average accuracy for retrieving the top-10 documents, A_{10}, as a function of the power law exponent, α, for different replication policies. Here, we have assumed that the query distribution follows a power law, and the search size is fixed to 337 nodes, ($z = 337$), which is the average search size for time-to-live set to 4 in our experimental configuration, as discussed in Section 5.4.

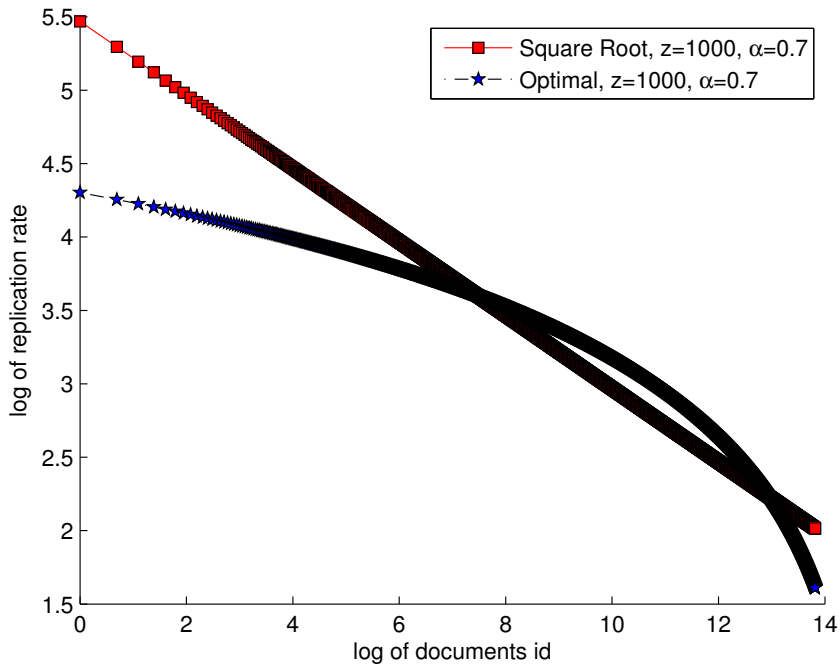

Figure 5.10: Replication rates of all documents in the collection, for square root and optimal replication policies, where $\alpha = 0.7$.

The uniform replication policy is independent of the query distribution. For all other replication policies, as the query distribution moves from uniform, i.e. $\alpha > 0$, we observe a very rapid improvement in accuracy, that monotonically increases with α. The optimal replication policy performs better than both the square root and the proportional replication policy. Under a practical $\alpha \approx 0.7$, which we calculate from [aol], the optimal replication achieves about 10% improvement over the square root replication.

Figure 5.10 shows the replication allocation when using the square root and optimal replication policies, when querying a fixed number of nodes, $z = 1000$. An interesting observation is that the optimum replication policy does not replicate the popular documents as much as the square root policy.

Figure 5.11 shows the global average accuracy, A_{10}, when retrieving the top-10 document, as a function of the search size, z, for different replication policies, and for an fixed scaling exponent $\alpha = 0.7$. When the search size is small, the differences between the three replication policies are greater. However, when the search size is larger, the accuracy is much less affected by the replication policy. Note that the proportional replication actually performs better than the square root for search sizes of less than $z = 300$. However, as the search size increases, the expected accuracy for the proportional policy increases more slowly and the square root policy is then superior. As expected, the optimal policy performs best for all search sizes.

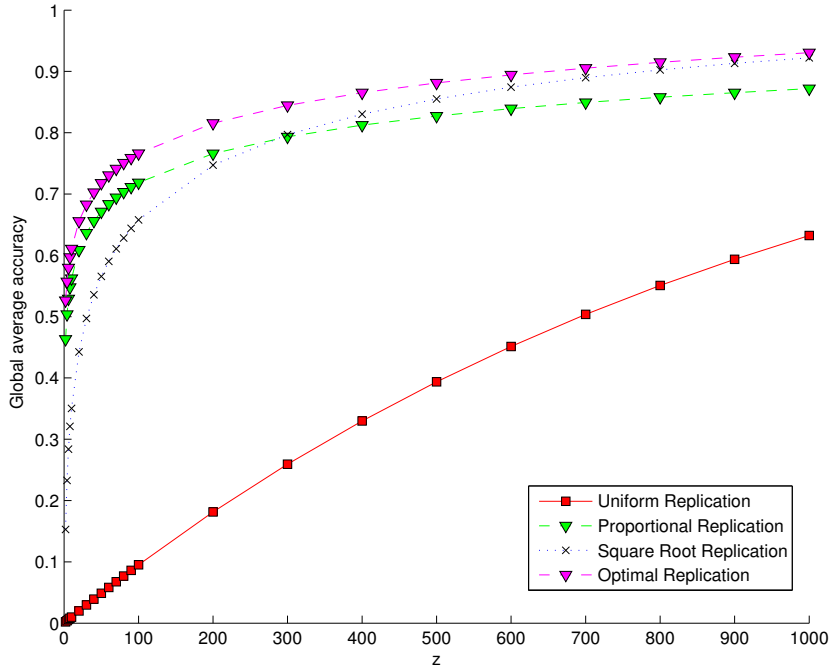

Figure 5.11: The global average accuracy for retrieving the top-10 document, A_{10}, as a function of the search size, z, for different replication policies, where $\alpha = 0.7$.

5.2.3 Scalability

An important issue in any distributed system is scalability, i.e. how a system performs in response to scalable factors in the system. In an unstructured P2P network, reasonable scalable factors include the network size and the collection size.

We first study how the variation of the network size affects the expected accuracy. To intuitively observe the effect of the varying network size, we use the same PAC configuration as in last subsection, where it is assumed that there are 1 million documents in the collection ($m = 10^6$), and each node is able to index 1000 documents ($\rho = 1000$). We vary the network size from 5,000 to 20,000 nodes. The expected accuracy of each replication policy under different network size is constant. To explain this we refer to Equation (5.8). The accuracy for each query is determined by $\frac{r_i}{n}$ given the search size, z, is fixed. In the case of uniform replication, $\frac{r_i}{n} = \frac{\rho}{m}$ is constant and thus unaffected by the network size. In the three non-uniform cases, the replication policy, r_i is approximately proportional to n (note $R = n\rho$). Thus, varying the network size n almost does not change the $\frac{r_i}{n}$, and thereby does not change the expected accuracy.

Next, we investigate the effect of varying the collection size. We use the same PAC configuration as above. However, we fix the network size $n = 10,000$, and vary the collection size, m from 500,000 to 2,000,000 documents. We calculate the expected accuracy for two query distributions, one for $\alpha = 0.8$ and the other for $\alpha = 1.0$. The expected accuracy curves for the different replication policies are shown

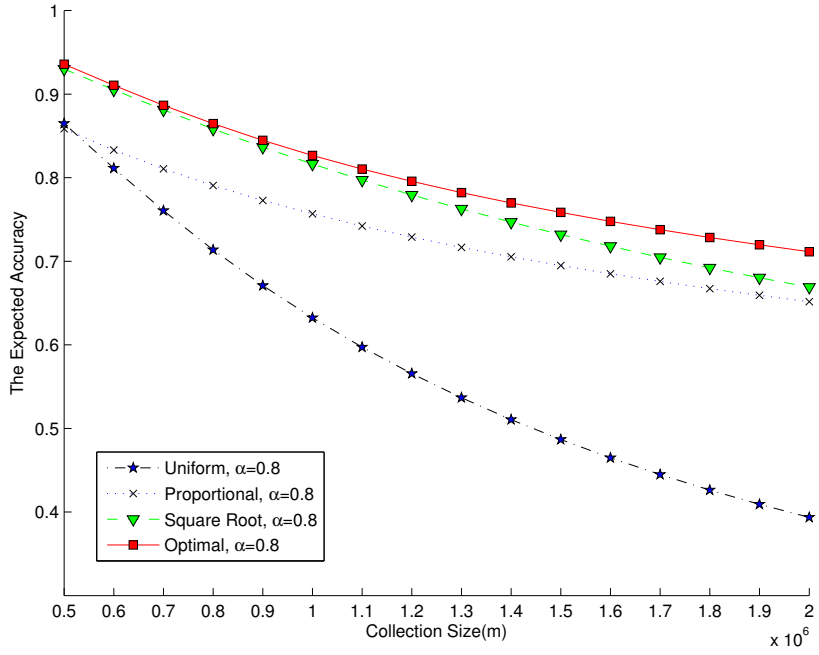

Figure 5.12: The expected accuracy for retrieving top-1 document, A_1, as a function of the collection size, m, for different replication policies.The α of the query distribution is set to be 0.8, and the search size $z = 1000$.

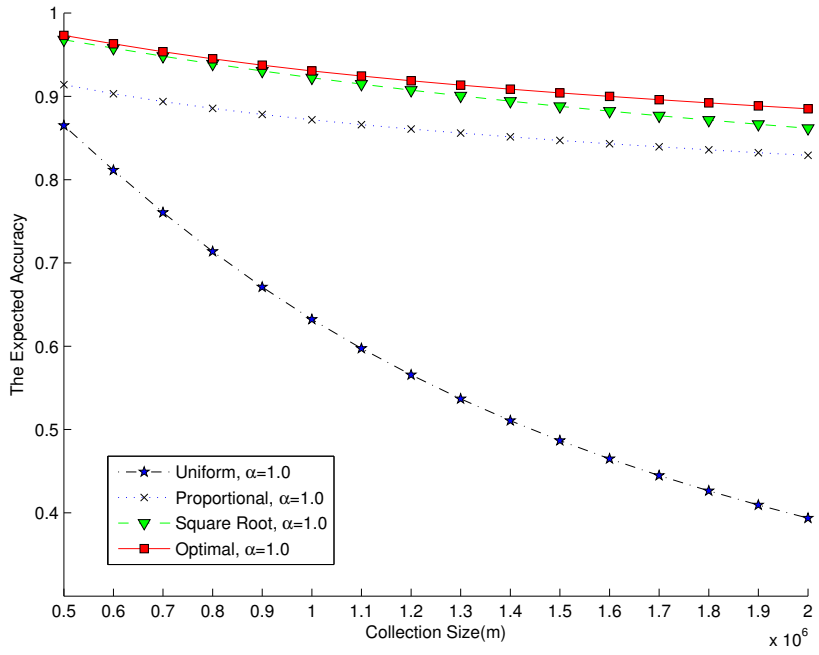

Figure 5.13: The expected accuracy for retrieving top-1 document, A_1, as a function of the collection size, m, for different replication policies.The α of the query distribution is set to 1.0, and the search size $z = 1000$.

in Figure 5.12 and 5.13. Both Figures show a similar trend - as the collection size grows, the expected accuracy for the non-uniform replication policies decrease more slower than for a uniform replication policy. Note however, that among the non-uniform replication policies, the square root policy degrades faster. Comparison of Figures 5.12 and 5.13 show that for larger α, the non-uniform policies degrade much less as the collection size grows.

5.3 Hybrid Replication Policy

Our analysis so far has compared the expected accuracy of the different replication policies. This is, of course, an average over all queries. In practice, it is also useful to understand the variation in accuracy across queries.

5.3.1 Variation in Accuracy

To study the variation in accuracy across different queries, we analyze each query's performance based on the same PAC configuration in the previous section. It is assumed that there are 1 million documents in the collection ($m = 10^6$), and the network size is ($n = 10000$). Each node is able to index 1000 documents ($\rho = 1000$). A query is sent to a fixed number of nodes, $z = 1000$. Once again we assume that only the top-1 document for each query is of interest, and that the top-1 document of different queries do not overlap. For a given query distribution, i.e. a fixed α, the standard deviation in accuracy, $std(A_1)$, is given by

$$std(A_1) = \left(\sum_i q_i (\mu_a(i) - A_1)^2 \right)^{\frac{1}{2}} \tag{5.14}$$

where A_1 is the global average accuracy for retrieving top-1 document of each query.

Figure 5.14 shows the standard deviation, $std(A_1)$, of the expected accuracy of individual queries when retrieving the top-1 document, as a function of the query distribution exponent, α, for different replication policies. All the three non-uniform replication policies exhibit a similar concave property in which the standard deviation increases as α increases from a small value of $\alpha = 0.1$, peaks in a region $0.7 < \alpha < 1$ and decreases as α continues to grow. The optimal and the square root replication policies always have lower standard deviation than the proportional.

Next, we examined the variation in accuracy across queries as a function of the search size, z. We assume two query distributions, with one $\alpha = 0.8$ and the other $\alpha = 1.0$. The results are shown in Figures 5.15 and 5.16. In general, the standard deviation declines as the search size increases for all three policies. The standard deviation of the optimal replication has the largest declining rate among the three when the search size grows, and this declining rate is also affected by the α value. When $\alpha = 0.8$, the optimal policy needs to search 1600 nodes to match the square root's standard deviation, however, this number is reduced to 1400 when $\alpha = 1.0$.

If we look at the AOL log [aol], we can find that the α of the query distribution is about 0.7, which

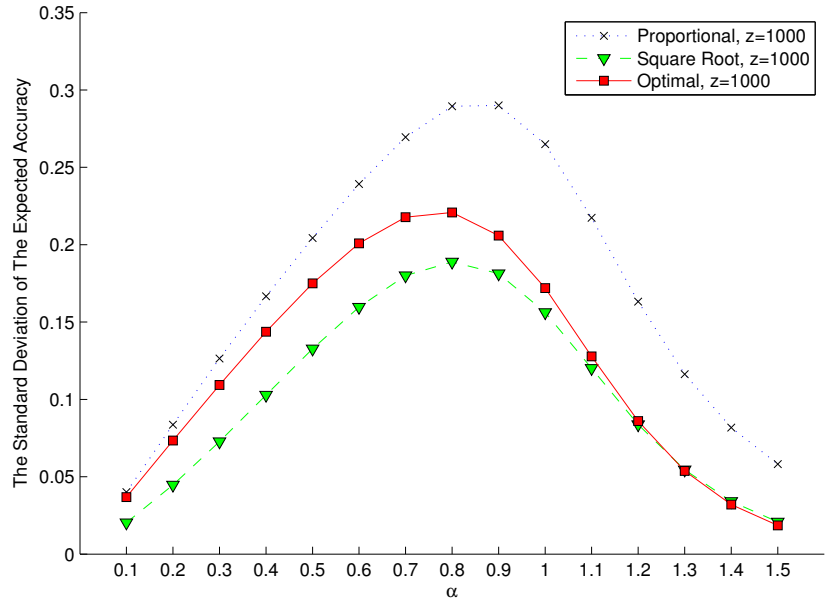

Figure 5.14: The standard deviation of the expected accuracy of individual queries for retrieving top-1 document, $std(A_1)$, as a function of the query distribution exponent, α, for different replication policies. Each query is sent to 1000 nodes ($z = 1000$).

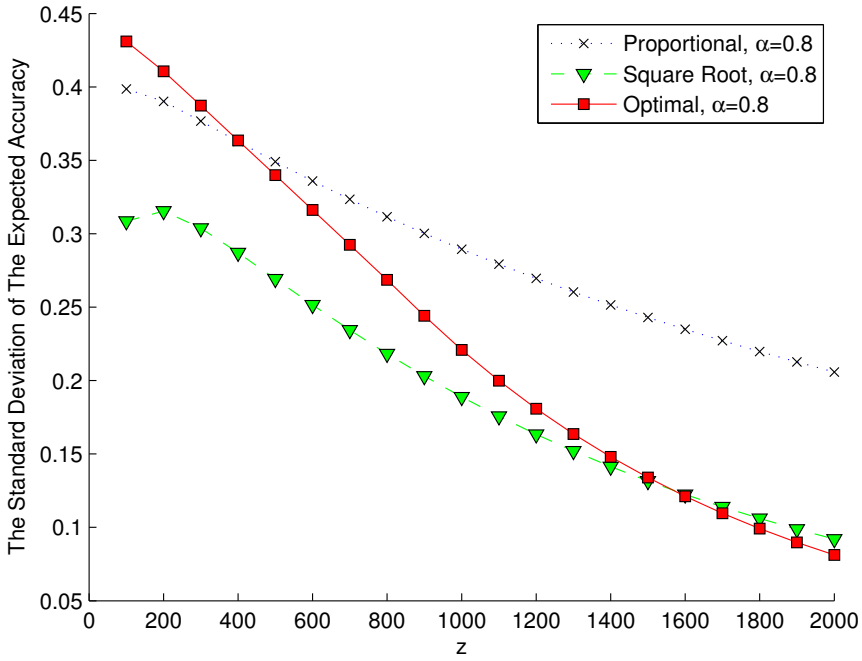

Figure 5.15: The standard deviation, $std(A_1)$, of the expected accuracy of individual queries for retrieving top-1 document, as a function of the search size, z, for different replication policies. The α of the query distribution is set to be 0.8.

is in the peak region of the standard deviation. This implies a large variation of search quality across different queries when we apply non-uniform replications. Figure 5.17 shows the variation in local ex-

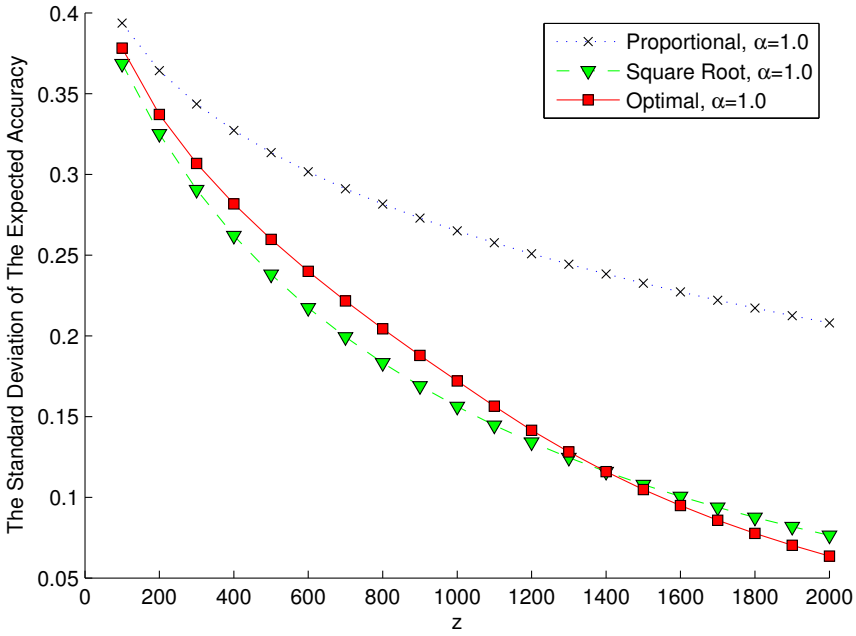

Figure 5.16: The standard deviation of the expected accuracy of individual queries for retrieving top-1 document, $std(A_1)$, as a function of the search size, z, for different replication policies. The α of the query distribution is set to be 1.0.

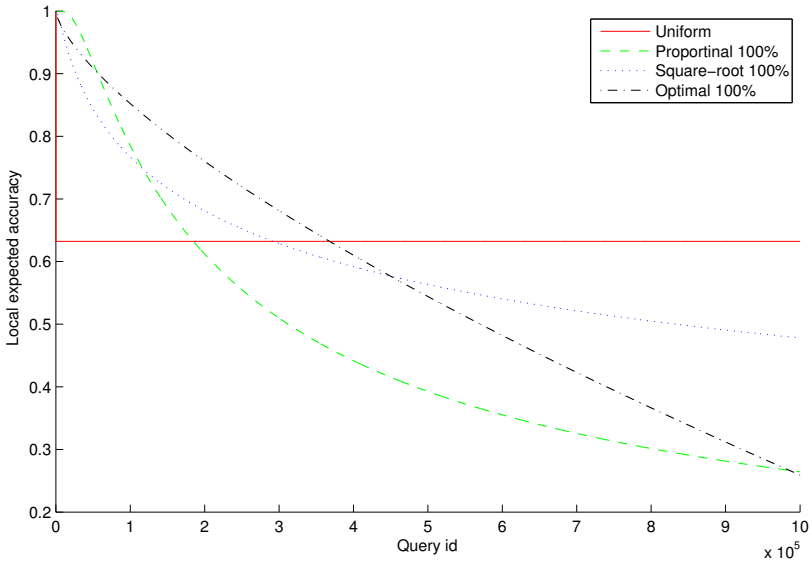

Figure 5.17: The distribution of the local expected accuracy of individual queries for retrieving top-1 document, for different document replication policies. The α of the query distribution is set to be 0.7.

pected accuracy across different queries under a specific query distribution where $\alpha = 0.7$, based on the same PAC configuration above. Clearly, compared to the uniform replication, the non-uniform replications improve global average accuracy at the cost of reducing search quality of unpopular queries that are in the tail part of the query distribution. This reduction may significantly impact the search system's

performance for some real world query logs such as in [BYGJ+07], where it is claimed that about 44% of of the query volume is singleton queries. Another interesting observation is that the optimal replication actually has more queries that have higher performance than the uniform replication, comparing to the square-root and the proportional replication, though it has a certain amount of unpopular queries that have lower performance than the square-root replication. In next section, we will seek to compensate for the loss of search quality of those unpopular queries, with only a small degradation to the overall search quality.

5.3.2 Combining Uniform and Non-uniform

In general, we would like to increase the global average accuracy to improve the overall search quality. However, users are likely to prefer a minimum acceptable performance for each query, and the performance lower than this minimum value is treated as unsatisfactory. Hence, beside the average accuracy, another important parameter we are interested in is the minimum acceptable local expected accuracy, $\bar{a}$, for an individual query. Our goal is to improve the global average accuracy based on a guaranteed minimum performance, $\bar{a}$, for each individual query. This could be achieved by combining the uniform and the non-uniform replication policies, where the uniform part provides the guaranteed minimum performance for each query, and the non-uniform part boosts the global search performance. We refer to this combination of the uniform and non-uniform replication as the hybrid replication policy.

The implementation of the hybrid replication policy follows a simple principle, in which each document has a minimum replication rate based on the proportion of the uniform storage. For example, suppose each document is replicated r times when the full storage is used for the uniform replication. If we next only use 80% of the storage for the uniform replication in a hybrid replication policy, then each document is at least replicated $0.8r(0.8r \geq 1)$ times in this hybrid replication. Therefore, we only need to modify the constraint $n \geq r_i \geq 1$ in the analysis of the last section to be $n \geq r_i \geq 0.8r$, and use the same algorithm to calculate the replication of each document in different combinations of the uniform and the non-uniform document replications. Let us use τ to denote the non-uniform storage ratio, i.e. the fraction of storage used for non-uniform replication, and $n \geq r_i \geq (1-\tau)r$. Since the minimum acceptable local expected accuracy, $\bar{a}$, is provided by the fraction of storage used for the uniform replication, it is easy to find the connection between τ and $\bar{a}$ as

$$\bar{a} = 1 - (1 - (1-\tau)\frac{r}{n})^z \tag{5.15}$$

where r is the average replication rate for a document if the full storage is used for the uniform replication. Thus τ can be determined by a given $\bar{a}$ as

$$\tau = 1 - \frac{n}{r}(1 - (1-\bar{a})^{\frac{1}{z}}) \tag{5.16}$$

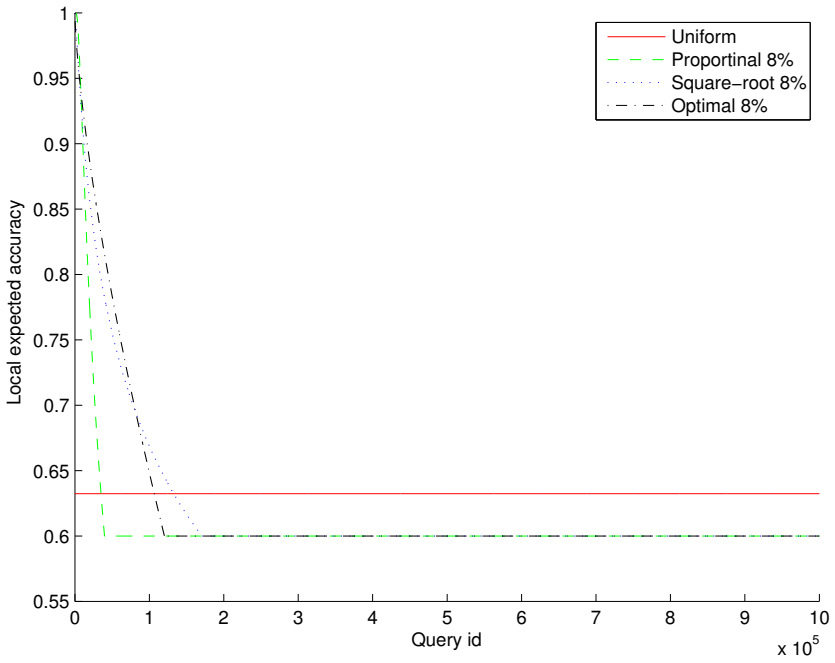

Figure 5.18: The distribution of the local expected accuracy of individual queries for retrieving top-1 document, for uniform and hybrid replication policies. The α of the query distribution is set to be 0.7.

Now consider that with the same system configuration above, we would like to guarantee a 60% minimum acceptable expected accuracy for any query, i.e. $\bar{a} = 0.6$. Substitute this $\bar{a}$ into Equation 5.16, we can obtain $\tau = 0.084$, thus, the constraint of the replication of each document becomes $n \geq r_i \geq 0.916r$. Using the method provided in section 5.2.2, we can obtain the local expected accuracy for each query as shown in Figure 5.18.

It is clearly shown that all hybrid replication policies have truncated curves, where less popular documents are given a guaranteed minimum accuracy, i.e. a guaranteed minimum replication rate.

Table 5.1: The global average accuracy of hybrid replication policies, where hybrid-prop indicates the combination of the uniform and the proportional replication, hybrid-sqrt indicates the combination of the uniform and the square-root replication and hybrid-opt indicates the combination of the uniform and the optimal replication.

τ	0%(uniform)	8%	100%
Hybrid-prop	0.6323	0.7142	0.7177
Hybrid-sqrt	0.6323	0.7427	0.7655
Hybrid-opt	0.6323	0.7450	0.7749

Table 5.1 shows the global average accuracy of different hybrid replication policies under different non-uniform storage ratios. Hybrid-prop indicates the combination of the uniform and the proportional replication, Hybrid-sqrt indicates the combination of the uniform and the square-root replication and Hybrid-opt indicates the combination of the uniform and the optimal replication. If all storage are used

for uniform replication ($\tau = 0\%$), all hybrid replication policies return back to the uniform replication policy and have the same global average accuracy as 0.6323. If we use about 8% of the storage for non-uniform replication, all hybrid replication policies have about 10% improvement in the global accuracy, which is close to using the full storage for non-uniform replication.

Table 5.2: The query acceptance ratio of hybrid replication policies, where hybrid-prop indicates the combination of the uniform and the proportional replication, hybrid-sqrt indicates the combination of the uniform and the square-root replication and hybrid-opt indicates the combination of the uniform and the optimal replication.

τ	0%(uniform)	8%	100%
Hybrid-prop	100%	100%	21%
Hybrid-sqrt	100%	100%	37.6%
Hybrid-opt	100%	100%	41.5%

Let us further denote the query acceptance ratio as the fraction of unique queries that are not lower than the minimum acceptable accuracy. Table 5.2 shows the query acceptance ratio of different hybrid replication policies under different non-uniform storage ratio. It is noticeable that when the full storage is used for non-uniform replication, about only 40% or less unique queries are acceptable. Comparing with Table 5.1, it might be arguable that using only part of the storage for non-uniform replication is better than replicating non-uniformly alone, in terms of the overall search quality and search stability.

Among all three hybrid replication policies, Hybrid-prop has some different properties from the other two. As mentioned in [CS02], the square-root replication is somewhere between the uniform replication and the proportional replication. Thus, in Hybrid-prop, it is not always increasing the global average accuracy as we increase the non-uniform ratio τ.

Figure 5.19 shows the variation of the global average accuracy of different hybrid replication policies under different non-uniform ratio τ. An interesting observation in Hybrid-prop is that the global average accuracy is not monotonically increasing as τ increases, instead, starts decreasing at $\tau = 0.3$. This suggests that under above system configuration, in any case it makes little sense to allocate more than 30% of the storage for proportional replication in Hybrid-prop, since more non-uniform storage simply decreases both the global average accuracy and the query acceptance ratio.

5.4 Experiments

In order to verify the preceding analysis, we performed a variety of cycle-based simulations using the PeerSim simulator [JMJV] The network settings used for the simulation are shown in Table 5.3. This configuration is similar to those studied in [SGG+02]. We assume a homogeneous network where each node has equal capacity, though in practice heterogeneous networks are more common. However, a heterogeneous network can be converted to a homogeneous one by using virtual nodes, i.e. a super node is modeled as several standard virtual nodes. This conversion does not affect the retrieval quality

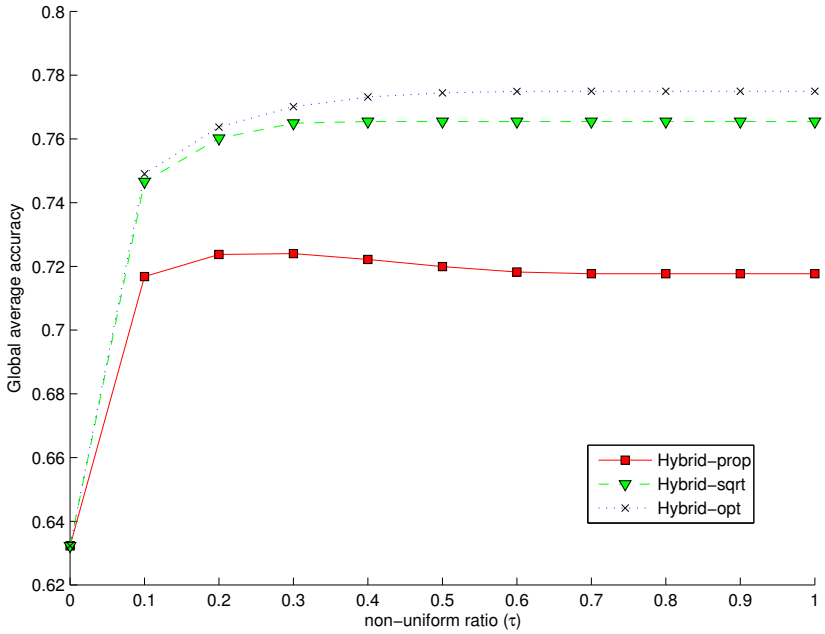

Figure 5.19: The global average accuracy of different hybrid replication policies under different non-uniform ratio τ.

Table 5.3: General Network Setting

Parameter	Value
Network size	10,000 nodes
node capacity	4000 documents
top-k	10

achievable. Hence, we believe that using a homogeneous network is adequate to verify our theoretical analysis.

Since we use a simple flooding technique to forward queries, the only parameter we can specify is the time-to-live (TTL) value for a query. The search size is indirectly determined by the TTL. Table 5.4 shows the corresponding search size, z, for TTL values we can use in our simulated network.

Table 5.4: The correspondence between TTL and average search size

TTL	1	2	3	4	5
z	5	21	85	337	1304

We first tested all four individual replication policies in our simulation, namely uniform, proportional, square root, and optimal, under different search size z. We then investigated the performance of the three hybrid replication policies, namely Hybrid-prop, Hybrid-sqrt and Hybrid-opt, under different non-uniform storage ratio τ. We assumed that each replication policy was implemented by a super node

in the network. How to implement each replication policy without a super node, e.g. using Gossip-based protocol to approximate query distribution on each node [JMB05], is not considered and is suggested for future work.

We use the WT10G document collection to test the performance under different replication policies. The collection size is 1,692,096 ($m = 1,692,096$). We use the AOL log [aol] as the test query collection, since the AOL log is the only full query log that provides the statistics of the full query volume and unique queries, i.e. the queries in AOL log are almost not filtered. The AOL log consists of about 30 million queries in which there are about 9.5 million unique queries. These queries roughly follow a truncated power distribution, i.e. $q_j = \frac{1}{c} j^{-\alpha}$, where $\alpha = 0.7$. Due to time and resource limitations, we sample 10,000 unique queries from the original AOL query log, following a power distribution with the same characteristics as the original query log, i.e. $\alpha = 0.7$ and average query frequency is about 3. For each sampled query we retrieve its corresponding global top-k documents in WT10G by Terrier [OAP+06], using the BM25 ranking model [RWJ+96]. To implement a replication policy, the super node first collects query statistics for a while, and calculates the allocation for each document. Then the documents are replicated in the network under the coordination of the super node according to the desired replication policy. After the replication, we begin the test query process to collect the performance data. In the test query process, we randomly sample 100,000 queries from the unique query distribution to make a test query collection. This volume of sampled queries assure that most queries appear at least once. For each time cycle (a time cycle can be regarded as 1 second), we fetch 1000 queries from the sampled query collection. We then randomly sample 1000 nodes in the network as the query issuers for these 1000 queries, one node for a query. For each time cycle we repeat this process till all queries in the sampled query collection are issued.

The metric we are interested in is the global average accuracy and the query acceptance ratio, as defined in Section 5.2.1 and 5.3. For each query, we measure its local expected accuracy, i.e. the expected overlap ratio of the returned result set over the true top-k document set for that query. We then average the local expected accuracy for each query to obtain the global average accuracy for the full query sample set.

5.4.1 Effect of Query Distribution

We performed a PAC search with the TTL set to 5, i.e. $z = 1304$ nodes visited in average, for each query. Figure 5.20 shows the simulation results of the expected accuracy for retrieving the top-10 documents, A_{10}, as a function of the power law exponent, α, for different replication policies. The optimal policy performs best. Comparison with the theoretical curves of Figure 5.1 shows very good qualitative agreement. The expected accuracy measured in the simulation is higher than the corresponding theoretical value, but this is due to the fact that the theoretcial value is calculated for $z = 1000$, while the simulation

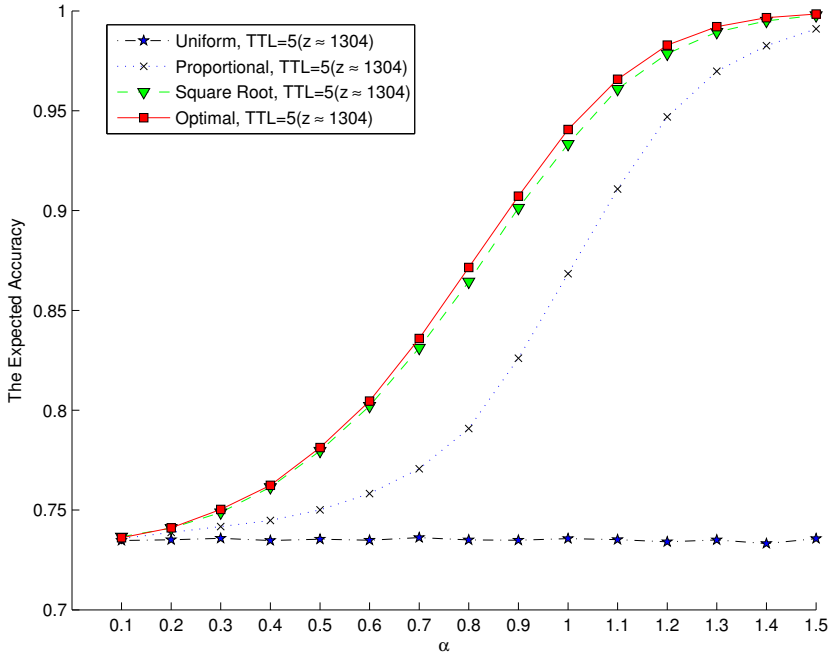

Figure 5.20: Simulation results of the expected accuracy for retrieving the top-10 documents, A_{10}, as a function of the power law exponent, α, for different replication policies. The TTL is set to 5, i.e. $z = 1304$ nodes visited on average for each query. Note that the top-10 result sets are disjoint.

queried, on average, $z = 1304$ nodes. However, the difference between the optimal and the square root is relatively small, comparing to the proportional replication. Nevertheless, the performance difference between the optimal and the square root is relatively large in the α range from 0.7 to 1.0, which gives an evidence to favor the optimal replication since the α computed from a real web query log [aol] is about 0.7.

5.4.2 Effect of Search Size

Figure 5.21 shows the measured global average accuracy as a function of TTL, for a query distribution where $\alpha \approx 0.7$. Each TTL implies an average search size given in Table (5.4). Similar theoretical results can be obtained from the formulas in Section 5.2.1, which adds support for our theoretical models. As noted earlier, we observe that the performance improvement of the optimal over the square root replication policy increases as the TTL, i.e. the average search size, decreases. This observation suggests that the optimal replication is especially advantageous under a small search size. Another interesting observation is that the proportional replication actually performs better than the square-root replication while the optimal replication keeps performing best. This could be due to that the square-root is not optimized for search accuracy but search size instead.

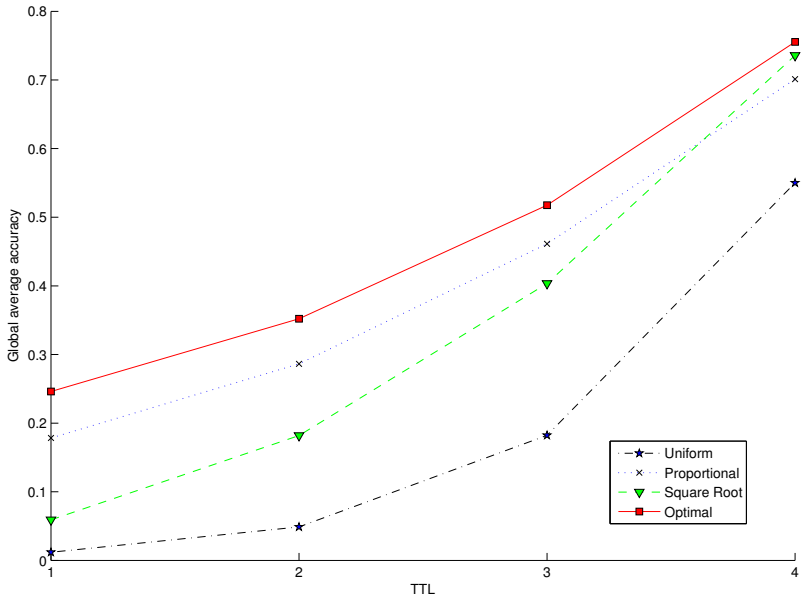

Figure 5.21: Simulation results of the global average accuracy for retrieving the top-10 documents, A_{10}, as a function of TTL, for different replication policies. The query distribution is roughly for an $\alpha = 0.7$.

5.4.3 Effect of Non-uniform Ratio

As discussed in Section 5.3, pure non-uniform replication may incur large variation of search accuracy across different queries. Figure 5.22 shows the distribution of the local expected accuracy of each unique query for retrieving top-10 documents in our experiments. The expected accuracy of each unique query is obtained by averaging the accuracies of all instances of that query. For the nature of random sampling, queries that are not sampled even once simply have accuracy 0 in our figure. Due to the overlap of the top-10 result sets of different queries, the performance of different queries presents a higher randomness than our previous theoretical analysis for retrieving only top-1 document. However, the performance of all non-uniform replications still follow a declining tendency in general as the query frequency decreases. Comparing to the uniform replication, all non-uniform replications again have much lower performance in the tail part of the query distribution, where unpopular queries concentrate.

Next we consider a 50% minimum acceptable expected accuracy for any query, i.e. $\bar{a} = 0.5$. Using Equation 5.16, we can obtain $\tau = 0.13$, which indicates that we should use 13% of the storage for non-uniform replication, in order to theoretically guarantee a 50% local expected accuracy for each query. We then implement each hybrid replication policies based on $\tau = 0.13$. Figure 5.23 shows the distribution of the local expected accuracy of individual queries for retrieving top-10 documents, for uniform and hybrid replication policies. Apart from the similar randomness in the performance of each query, all hybrid replications show a much more smoothed performance in their tail part of the query distribution. The performance of relatively unpopular queries concentrate more closely around the threshold 50% set

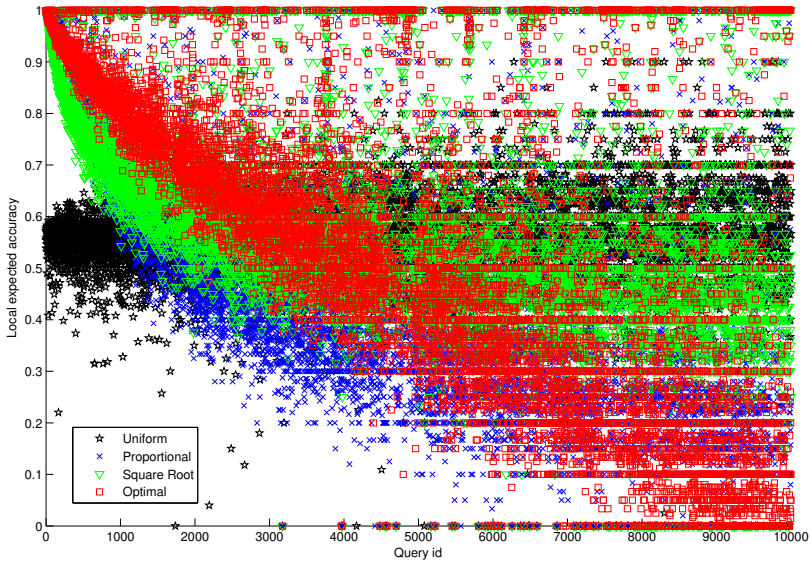

Figure 5.22: The distribution of the local expected accuracy of individual queries for retrieving top-10 document, for different document replication policies.

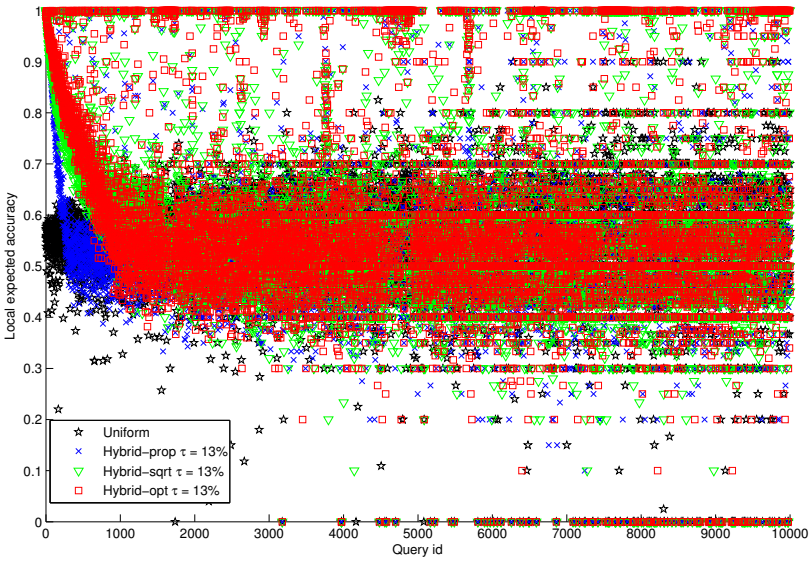

Figure 5.23: The distribution of the local expected accuracy of individual queries for retrieving top-10 document, for uniform and hybrid replication policies.

above. The extremely bad performing queries are expected to be reduced.

Table 5.5 shows the global average accuracy of different hybrid replication policies under different non-uniform storage ratio. While the uniform replication has only 55% global average accuracy, Hybrid-prop improves 22%, Hybrid-sqrt and Hybrid-opt improves about 30% and 31%. Table 5.6 shows the query acceptance ratio of hybrid replication policies. Here we treat the query acceptance ratio of the uniform replication as a standard, since the uniform replication provides most even performance across

Table 5.5: The global average accuracy of hybrid replication policies.

τ	0%(uniform)	13%	100%
Hybrid-prop	0.55	0.6716	0.7011
Hybrid-sqrt	-	0.7157	0.7357
Hybrid-opt	-	0.7208	0.7554

Table 5.6: The query acceptance ratio of hybrid replication policies.

τ	0%(uniform)	13%	100%
Hybrid-prop	72.7%	61.5%	37.5%
Hybrid-sqrt	-	69.2%	61.1%
Hybrid-opt	-	67.7%	56%

queries. All hybrid replication policies improve the query acceptance ratio towards the 72.7% standard. The significant improvement occurs in Hybrid-prop that has 64% increase, while Hybrid-opt and Hybrid-sqrt have 21% and 13% improvement respectively. These two tables suggest that hybrid replication provides more stable performance for different queries, with only a small degradation to the overall performance averaged over all queries.

5.5 Summary and Future Work

Search over an unstructured P2P network is probabilistic. We assumed that the search size was fixed and then measured the performance of a system using an accuracy measure, which is defined based on the size of the intersection of the documents retrieved by a fixed size, probabilistic search and the documents that would have been retrieved by an exhaustive search. In this scenario, we then considered how documents should be replicated in order to maximize the accuracy.

We showed that the well-known square root replication policy does not maximize accuracy, even though it minimizes the expected search size. We then defined the optimality criterion for maximizing accuracy and provided an optimal solution. We then compared the performance of four different replication policies, namely uniform, proportional, square root and optimal. Our comparisons examined how the expected accuracy varied as a function of (i) the query distribution, which was assumed to follow a power law with exponent α, and (ii) the search size, z. As expected, the optimum policy outperformed all other policies. The difference in performance between the optimum and square root policies increased as the search size decreased. A more detailed comparison of the two replication policies revealed that the optimum policy tended to replicate the most popular documents less frequently than the square root policy, while increasing the frequency of documents with middle popularity and decreasing the replication frequency of the least popular documents.

We also examined the variation in accuracy across queries. It was observed that all three non-

uniform replication policies exhibited qualitatively similar characteristics - the standard deviation increases as α, the exponent of the query power law, increases from 0.1 to between 0.8 to 1. As α continues to increase, the standard deviation then decreases. We then argue that for a real world query log which query distribution has $\alpha = 0.7$, any non-uniform distribution of documents improves the retrieval performance of popular documents at the expense of less popular documents. To compensate for this, we propose a hybrid replication policy consisting of a combination of uniform and non-uniform distributions. Theoretical results show that such an arrangement significantly improves the accuracy of less popular documents at the expense of only a small degradation in accuracy averaged over all queries.

A simulation of search in an unstructured P2P network was then performed using the PeerSim simulator. Experimental results confirmed our theoretical analysis.

Future work is needed to extend our theoretical analysis to the case where each replication policy can be implemented with only local information available to each peer. And another interesting problem for future work might be how each replication policy is affected by the churn of the network.

Chapter 6

Query Caching

In this chapter, we explore the query caching problem in PAC search architeture. It has been observed that the query distribution is usually non-uniform, and typically follows a power law distribution [aol, BYGJ+07]. The previous Chapter investigated replicating documents across nodes based on different policies in order to improve search performance in PAC architecture. A uniform document replication policy distributes all documents with equal probability and therefore has no bias in retrieval. Non-uniform document replication policies distribute documents across peers based on their popularity in order to increase the probability of retrieving popular content. This is achieved at the expense of less popular documents, which become harder to find. Clearly a compromise must be made between the probability of finding popular and unpopular documents. One such compromise is for each peer to allocate a proportion of its storage to a uniform distribution, thereby guaranteeing a minimum probability of finding any document, and the remainder of its storage to a popularity-based distribution, thereby increasing the probability of finding popular documents. We have studied a hybrid document replication policy which uses this latter storage for a non-uniform document replication, however, it is also possible, and might be more efficient to consider this latter storage as a cache that stores query results. In this chapter, we propose to cache only the results/postings corresponding to the queries on each peer, no longer the full index of popular documents, in order to retrieve popular documents. This requires substantially less space and, for a fixed cache size, permits a larger number of references to popular documents to be cached. Note that previous works on query caching [BCG+03, THT11] are more aimed at reducing the computational cost and latency of retrieval, and not at improving search quality. In comparison, our major concern is to improve search quality through query caching under a communication budget. Of course, the cached query results could also be used to reduce computational cost.

The main contributions of this chapter are (i) an analysis of query caching techniques to improve search quality, (ii) experimental comparison, based on simulations, between the proposed query caching techniques and those of uniform, proportional and square root document replication strategies.

The rest of this Chapter is organised as follow. Section 6.1 then describes the query caching tech-

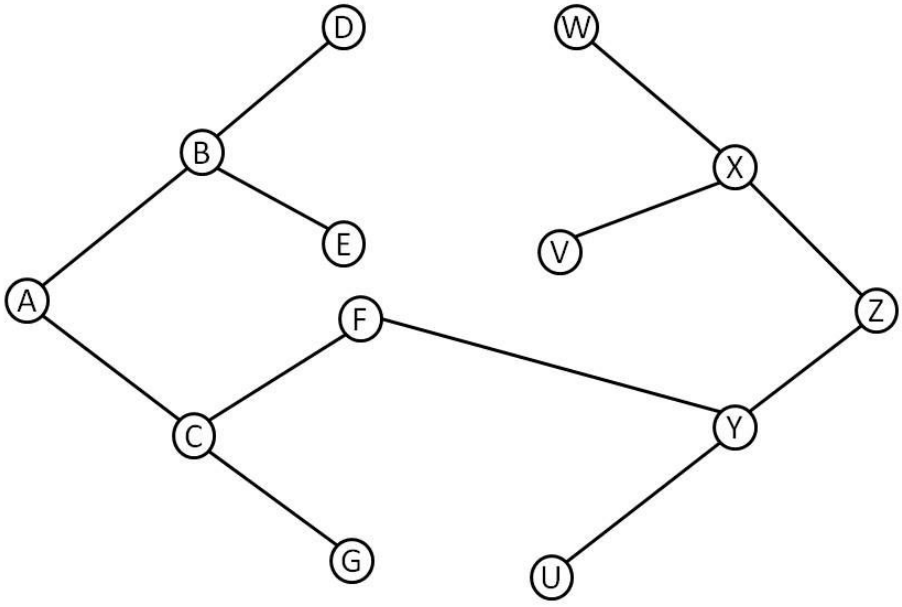

Figure 6.1: A simple example of search in a P2P network.

niques we propose in this chapter. Section 6.2 provides experimental results on a simulated network of 10,000 nodes using the WT10G database of 1.7 million documents. Finally, Section 6.3 provides a summary and discussion of future work.

6.1 Model

Equation 5.5 quantifies what is intuitively obvious, namely that the more documents a local node stores (indexes), r, and the more nodes we query, z, the greater the accuracy. Query caching improves both. However, to show this, we first discuss how query caching can be implemented.

We assume a network with a uniformly random connection topology. Other topologies are possible, but are beyond the scope of this chapter. Given this network topology, a peer in the network has an expected out-degree, o, which is the peer's average number of immediate neighbours. When a peer issues a query, the query is sent to all of its neighbours together with a time-to-live (TTL) parameter. Each neighbour decrements the TTL and if greater than zero, sends the query to its immediate neighbours. Clearly the TTL parameter controls the total number of peers a query is sent to. On average, for a network with out-degree, $o = 4$, a TTL of 4 is equivalent to $z = 337$.

Figure 6.1 illustrates this for a TTL of 2. Node A is a a query issuer which sends a query to all of its immediate neighbours, i.e. nodes B and C. On receipt of a query, a peer searches its local storage and retrieves a local top-k result set. Meanwhile, it forwards the query to all of its immediate neighbours

again, and waits for them to return their search results. It then merges these result sets with its own local top-k and returns this partial result set. Note that more distant neighbours, e.g. Node E, respond by sending their local top-k result set back via intermediate neighbours, e.g. Node B, as illustrated in Figure 6.1. We note that this is a common assumption of many P2P networks [DLAV].

After the querier, node A, has received the partial top-k results from its immediate neighbours, B and C, it re-ranks them together with its local top-k results to arrive at an amalgamated top-k. It is this list that is provided to the user.

We considered two methods of caching query result sets. The first method replicates the query result set amalgamated at node A, and then sends this query result set to each of the queried peers, who also replicate the query result set. This has the advantage of ensuring that all nodes involved in the query replicate the same, locally best, query result set. The disadvantage is that it incurs an additional layer of communication cost, which we may wish to avoid. We refer to this method as Forward Query Caching.

The second method, replicates the intermediate partial query-result sets. Thus, when node E receives the query, it sends its local top-k to node B, and replicates its local query result set in its local index. When node B receives the results from nodes D and E, it amalgamates them with its local result set and sends the partial top-k to node A. Meanwhile, node B replicates locally amalgamated query-result set. Finally, node A amalgamates the results from its local query result set and the results received from nodes B,and C to produce the locally best top-k, and replicates this in node A's cache. The advantage of this method is that no additional communication cost is incurred. The disadvantage is that the query result sets replicated on each node are different, and only the querier, node A, replicates the locally best query result set. We refer to this method as Backward Query Caching. We expect the performance of the Backward Query Caching to be inferior to the Forward Query Caching, and this is observed in simulations, described in Section 6.2.

Consider now the case where node Z issues the same query. In this example, the query is sent to nodes X and Y, which forward it to nodes V, W and U, F respectively. If we assume Forward Query Caching, then node F has the top-k results obtained from querying the nodes A through F. Node Z amalgamates the results from F as well as the local top-k results from U through X, to create a new top-k which is then distributed to all the queried nodes. The expected accuracy of the retrieval results at node Z is a function of the number of nodes queried, see Equation 5.5. However, this is for uniform replication. In practice, node F contains the results of querying $z = 6$ random nodes, so node Z contains the results of querying the equivalent of $z = 11$ random nodes. In addition, the Forward Query Caching increases the replication rate for the specific query, so r in Equation 5.5 is also increased. Both increases in z and r significantly improve the accuracy of the search at node Z. A similar argument holds for the Backward Query Caching method. However in this case, the increase in equivalent z is not as great.

Both methods cache query result sets directly proportional to their popularity. As discussed in

[CS02], it is shown that replicating documents in proportion to the square root of their popularity was optimum from the perspective of minimizing the expected search length. However, a square root replication strategy is *not* optimal for optimizing accuracy. In fact, for small z, i.e. the number of nodes queried, the proportional replication strategy can actually be better, as shown in Figure 6.2. The Figure assumes a network size of $n = 10,000$, with each node locally indexing 0.1% of the entire collection. Note that in this case, the initial replication rate for each document is 10. We observe that the proportional replication strategy is actually superior to the square root replication strategy when the number of nodes queried, $z \leq 435$. The optimal replication strategy, which is optimized directly to the retrieval accuracy, is of course the best performer all the time. The simulation discussed in Section 6.2 queries $z = 337$ nodes on a network of 10,000 nodes, and in this case replicating based directly on popularity should be better than replicating square-root.

6.2 Experiments

In order to verify and compare the different replication techniques, we performed a variety of cycle-based simulations using the PeerSim simulator [JMJV]. The network size was 10,000 nodes and the time-to-live, TTL=4. The topology was a random graph with degree 4. This configuration is similar to those studied in [SGG+02].

We assume a homogeneous network where each node has equal capacity, though in practice heterogeneous networks are more common. However, a heterogeneous network could be converted to a homogeneous one by using virtual nodes, i.e. a more capable node can be modeled as several standard less capable virtual nodes. This conversion does not affect the retrieval quality achievable. Hence, we believe that using a homogeneous network is adequate to verify our theoretical analysis.

Since we use a simple flooding technique to forward queries, the only parameter we can specify is the time-to-live (TTL) value for a query. The search size, z, is indirectly determined by the TTL. We used a TTL=4 which corresponds to querying $z = 337$ nodes, on average.

We investigated both Forward and Backward Query Caching approaches. Comparison is also made with the proportional, square root [CS02] and optimal document replication policies.

We used the WT10G document collection. The collection consists of 1,692,096 ($m = 1,692,096$) unique documents. After stemming and filtering stop-words, we observed that the collection has a total of 675,625,071 terms. Thus, on average, each document consists of 399.28 terms. We assume that each node can store 0.1% of the collection, i.e. 1692 documents, in its local storage. When indexing a document, each unique term in the document requires a corresponding posting in the inverted index. Note that we use a posting as a basic unit to measure the local storage, as all the replication strategies store indices to documents rather than the documents themselves.

We used an AOL log [aol] as the test query collection. The AOL log consists of about 30 million

queries in which there are about 9.5 million unique queries. These queries roughly follow a truncated power distribution, i.e. $q_j = \frac{1}{c} j^{-\alpha}$, where $\alpha = 0.67$. Due to time and resource limitations, we sampled 10,000, 20,000 and 40,000 unique queries from the original AOL query log. For each set of unique queries, we then generated a sequence of 1 million queries based on these unique queries, and which follows a power law distribution with $\alpha \approx 0.7$. For each sampled query we retrieve its corresponding global top-10 documents in WT10G by Terrier [OAP+06], using the BM25 ranking model [RWJ+96]. This result set is equivalent to exhaustively searching the entire network and serves as ground truth data with which to determine the accuracy of the various replication policies.

For the 10,000 query set, we observed that of the 100,000 documents retrieved, 74,690 of the documents were unique. This corresponds to 4.4% of the entire collection. Similarly, the 20,000 query set retrieved 135,864 unique documents, corresponding to 8% of the collection. And the 40,000 query set retrieved 236,228 unique documents, or 14% of the collection. These percentages were used to set a limit on the storage available for replication, discussed next.

Each node initially indexes 0.1% of the collection, i.e. 1692 documents. These documents are initially drawn from a uniform distribution. When a document or query-result set is to be replicated any replication policy must decide which documents or terms must be deleted from a node's local storage in order to provide space for the replicated item. An obvious and commonly used policy is a least-recently-used (LRU) policy in which the least recently retrieved documents/terms are discarded. However, for query test sets consisting of between 10 to 40 thousand unique queries, the majority of documents in the collection are never retrieved. Consequently, we would expect that most replication policies will perform well if given access to a very large fraction of storage space which can be discarded with no repercussion on the measured accuracy. To circumvent this problem, we only permitted the documents belonging to the unique set of retrieved documents to be discarded. Thus, for example, using a 10,000 unique query set, only documents belonging to the 74,690 unique set of retrieved documents can be discarded. Similarly, for the 20,000 unique queries, only documents belonging to the 135,864 unique documents retrieved were permitted to be discarded. And for the 40,000 unique query set, only documents from the 236,228 retrieved documents could be discarded. For Forward and Backward replication policies we discard documents based on a least recently used basis.

For each time cycle (a time cycle can be regarded as 1 second), we fetched 1000 queries from the sampled query collection of 1 million queries. We then randomly sampled 1000 corresponding nodes in the network, each node issuing one of the queries. At each subsequent time cycle we repeated this process until all queries in the sampled query collection were issued. For each query, we measured the accuracy, i.e. the fraction of documents in the true top-k that are retrieved by a node. We report the expected accuracy, i.e. the average over the accuracies for all queries.

As noted earlier, each simulation of the Forward and Backward query caching begins with doc-

uments uniformly distributed across nodes. There is therefore a period during which the accuracy increases as the replication policy begins to take affect. Eventually, a steady-state period should occur where the replication policy has achieved its best performance. We assumed that a steady state would be reached after issuing 1 million queries, i.e. after 1,000 cycles of the simulation. After issuing 1 million queries we then issued a further 3,000 queries and measured the accuracy based on these queries. Note that we implemented all the proportional, square-root and optimal replication policies in a centralized manner in which the nodes in the network were initialised with the required distribution prior to queries being issued. One reason for this is that a distributed implementation of the the square-root replication policy requires a variable search length. However, our search length is fixed.

Table 6.1 enumerates the measured expected accuracies of five different replication policies, namely (i) uniform distribution (no replication) (U), (ii) proportional replication (P), (iii) square-root replication (SQ), (iv) optimal replication (O), (v) forward query caching (F), and (vi) backward query caching (B). As expected, the measured expected accuracy for the uniform distribution is worse, with an accuracy of 28.4%, i.e. on average we retrieve only 28.4% of the result set obtained when performing an exhaustive search. The theoretical accuracy, as given be Equation (5.4) is 28.6%, which is in close correspondence. We also observe that this accuracy does not change with the number of unique queries in the 1 million query sample. Again, this is to be expected.

The proportional replication policy outperforms the square root policy for all three query sets. This might at first seem surprising. However, remember that the square root distribution optimises expected search length, not retrieval accuracy. In fact, as discussed earlier, a proportional replication policy can performs better with respect to accuracy, when search size is small. Referring back to Figure 6.2 the accuracy is plotted as a function of the number of queried nodes for three different document replication policies: (i) proportional, (ii) square-root, (iii) uniform, and (iv) optimal. As expected, a uniform distribution that is independent of the popularity of documents, performs worse. However, for the proportional and square root replication policies, we observe that the proportional distribution exhibits superior accuracy for search sizes of less than 435 nodes. For search sizes greater than 435, the square root distribution exhibits better accuracy. Table 6.1 is for the case where $z = 337$, and is therefore in the region where proportional replication is better than square root replication.

The proportional and square root policies significantly improve the accuracy compared with the uniform distribution. The observed improvements are (91%,95%,100%), (85%,87%,90%) and (112%,115%,117%) for the proportional, square root and optimal policies respectively. Each triplet of percentages refers to query distributions drawn from 10, 20 and 40 thousand unique queries respectively. Note that all the three document replication policies improve as the number of unique queries increases. We suspect that this may be due to the fact that the available storage for replication increases from 4.4% to 8% to 13.7% of the entire collection as the number of unique queries increases from 10,000

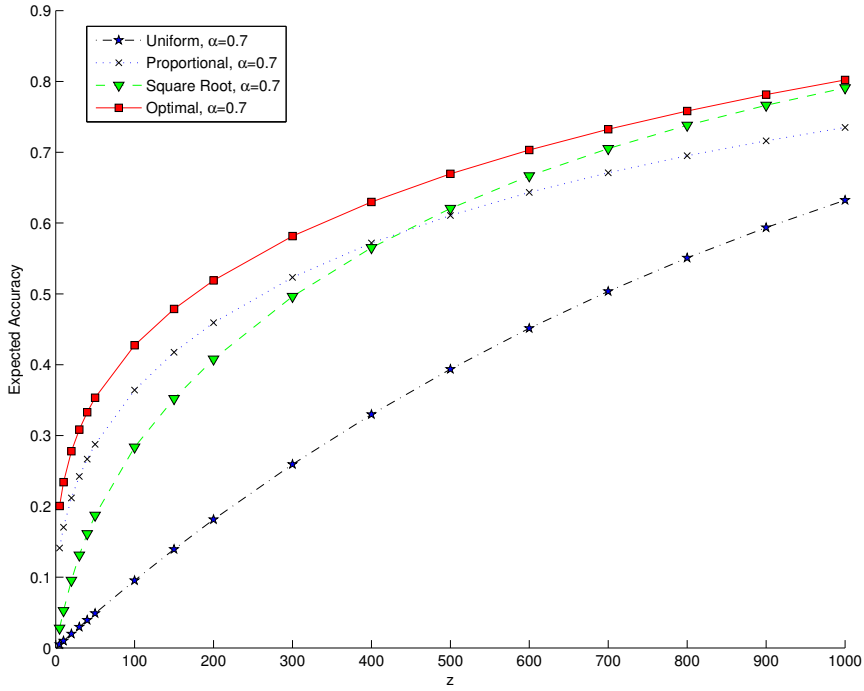

Figure 6.2: Accuracy as a function of the number of nodes queried, z, for three distributions of documents - (i) uniform, (ii) proportional, (iii) square-root, and (iv) optimal - for a network of 10,000 nodes, with each node indexing 0.1% of the collection. The accuracy curves for proportional and square-root distributions intersect at z=435.

to 20,000 to 40,0000. Further experimentation is required to confirm this.

Both the forward and backward query caching significantly outperform proportional, square-root and optimal document replication. The accuracy of the Forward caching represents a (66%,74%,65%), (72%,81%,74%) and (50%,58%,52%) improvement over proportional, square-root and optimal, for 10, 20 and 40 thousand unique queries, respectively. Similarly, the accuracy of the Backward caching represents a (58%,70%,53%), (64%,77%,61%) and (43%,55%,40%) improvement over proportional, square-root and optimal. The improvement over uniform is (217%,240%,230%) and (203%,232%,205%) for the Forward and Backward query caching, respectively. As expected, the Forward query caching performs best. However, the degradation in performance of the Backward query caching is relatively small, i.e. less than 8%, and is achieved using almost half the communications bandwidth of the Forward caching. We also observe a small degradation in performance of both the Forward and Backward caching as the number of unique queries increases from 20,000 to 40,000. The reason for this is unclear and is in our future work.

Table 6.1: Comparison of expected accuracies of five different replication policies: (i) uniform distribution (no replication) (U), (ii) proportional document replication (P), (iii) square-root document replication (SQ), (iv) optimal document replication (O), (v) forward query caching (F), and (vi) backward query caching (B). The first column is the number of *unique* queries in the query sample. The remaining columns enumerate the average accuracy obtained by each replication policy.

Query Set	U	P	SQ	O	F	B
10000	0.284	0.543	0.524	0.601	0.899	0.860
20000	0.285	0.557	0.534	0.612	0.969	0.946
40000	0.287	0.573	0.545	0.624	0.946	0.876

Any replication policy replaces unpopular items with more popular items. Consequently, there is a risk that the overall performance of the system is highly skewed, i.e. the accuracy for popular documents is very good which the accuracy for less popular documents is very poor. The expected accuracy does not reveal this. Figures 6.3-6.7 plot the cumulative percentage of queries with a recorded accuracy greater than a specified accuracy, for Forward query caching, Backward query caching, proportional, square-root and optimal document replication policies, respectively.

Consider, first, Figure 6.3 for the Forward replication policy. We observer that 27.5% of queries are answered with an accuracy of 1. That is, the retrieved result set is identical to the result set that would have been acquired based on an exhaustive search. In comparison, only 12.5%, 0.04% and 0% of queries have an accuracy of 1, for proportional, square root and optimal replication, respectively. The shape of the curve in Figure 6.3 is also noticeable different from those for proportional, square-root and optimal (Figures 6.5, 6.6 and 6.7. In particular, the cumulative percentage of queries above a given accuracy rapidly increases as the accuracy decreases in Figure 6.3. Thus, 80.5% of queries have an accuracy of 0.7, or greater. However, for proportional, square root and optimal replication, only 17.6%, 15.6% and

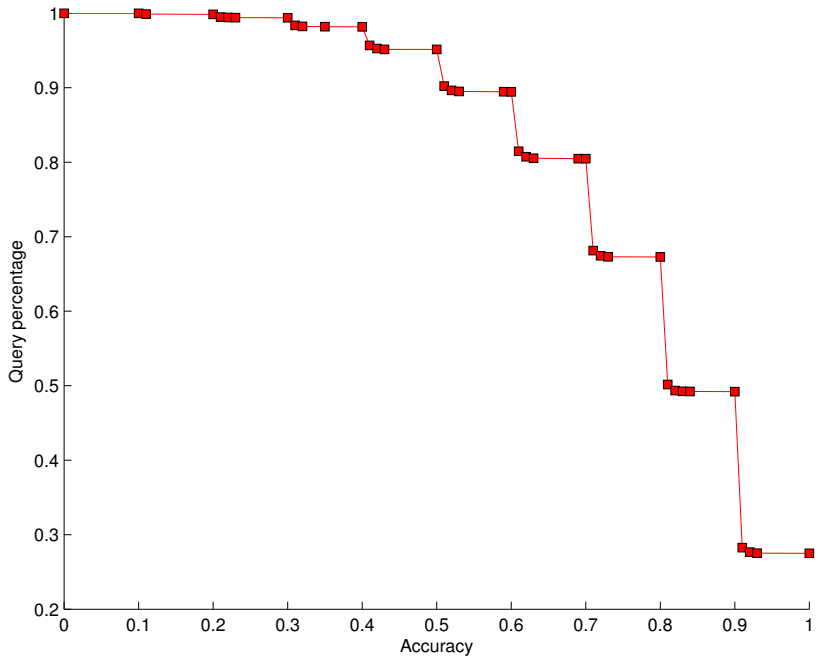

Figure 6.3: Query percentage larger than accuracy x for Forward query caching.

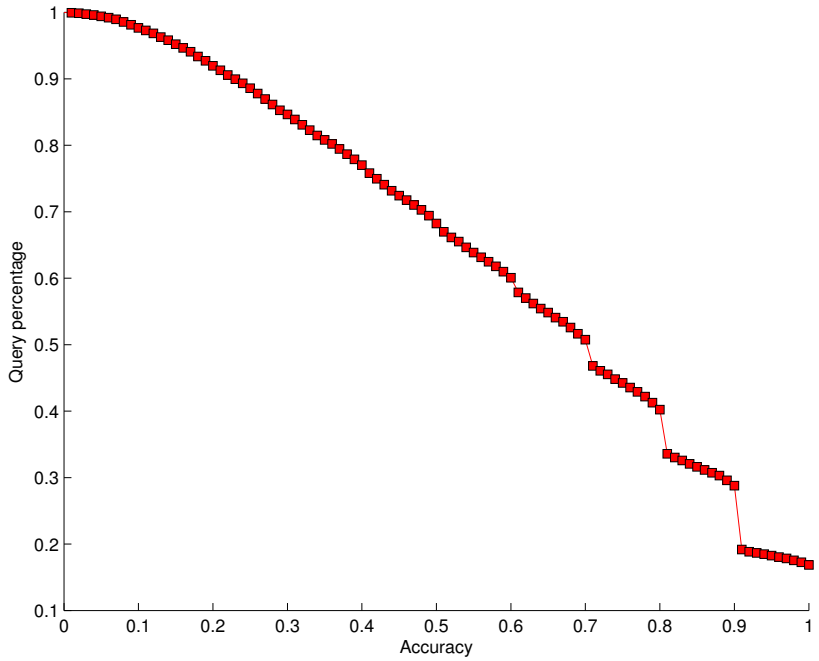

Figure 6.4: Query percentage larger than accuracy x for Backward query caching.

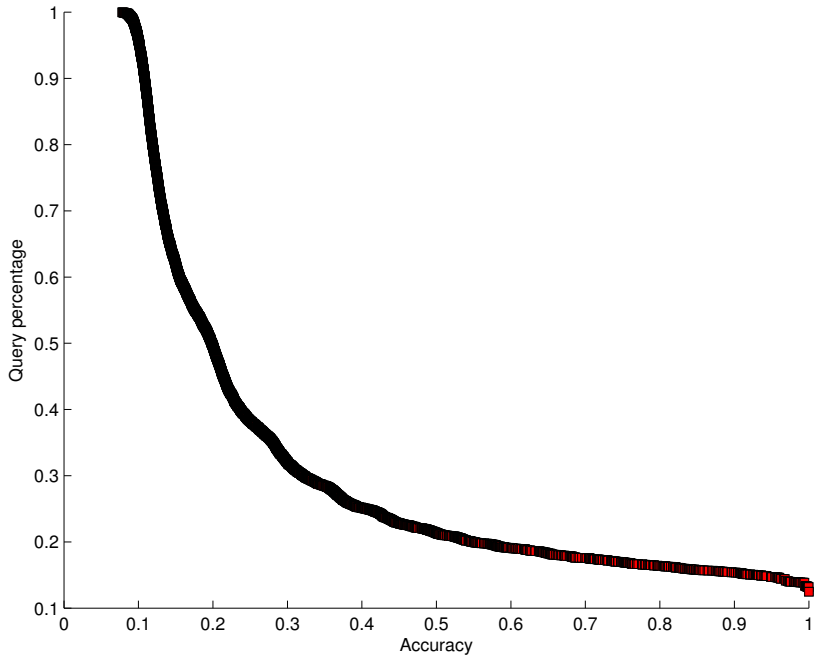

Figure 6.5: Query percentage larger than accuracy x for proportional document replication.

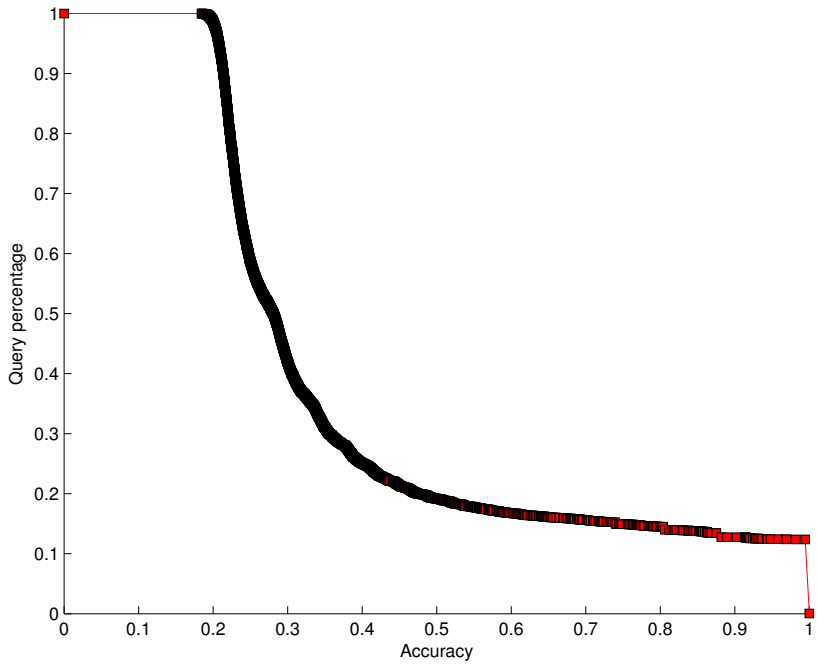

Figure 6.6: Query percentage larger than accuracy x for square-root document replication.

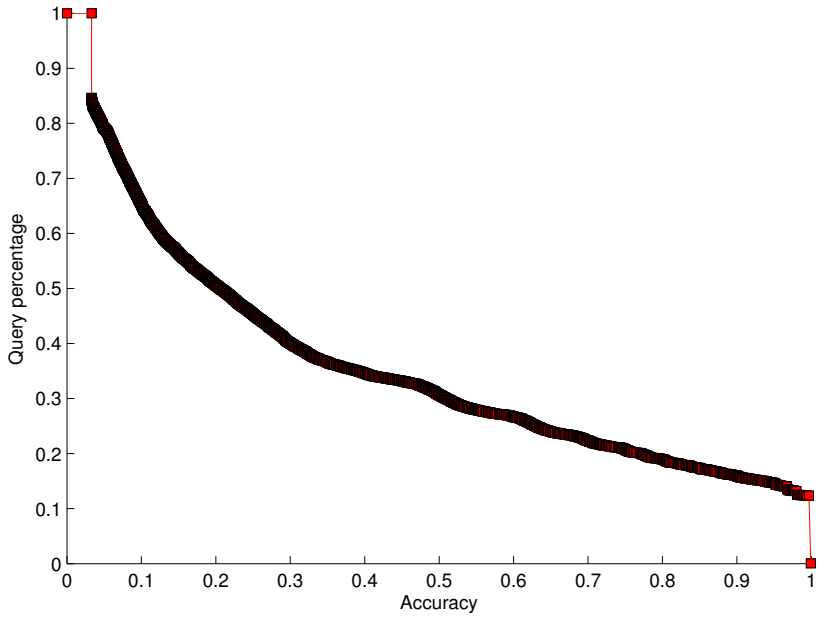

Figure 6.7: Query percentage larger than accuracy x for optimal document replication.

22.3% of queries are observed to have an accuracy greater than or equal to 0.7. This is much worse that Forward replication. Thus, not only is the measured expected accuracy of Forward replication very much better than either proportional or square-root, but high accuracy is also achieved across a much wider number of unique queries.

Figure 6.4 shows the percentage of queries with measured accuracy greater than a specified value for Backward query caching. We observe that 16.7% of queries are answered with an accuracy of 1. This is better than proportional, square root and optimal replication but worse than Forward query caching. The curve is also flatter than that for the Forward Query Caching. As a result, the corresponding number of queries with an accuracy of 0.7 is 50.8%, which is less than the 80.5% for the Forward policy.

While the Forward query caching is clearly superior to the Backward query caching, this performance improvement is achieved at the expense of increased communication bandwidth. In particular, in Forward query caching, the query issuer must send the final amalgamated result set to all of the peers that responded to the query. This effectively doubles the bandwidth required to perform the search. An obvious question to ask is whether a similar performance improvement could be obtained by simply doubling the number of nodes queries and using no replication policy, i.e. leaving the documents uniformly distributed. We can calculate this expected accuracy using, Equation (5.4) - the accuracy increases from 0.286 to 0.491. Thus, the Forward query caching provides much better results than simply doubling the number of nodes queried.

We also investigated how quickly the Forward and Backward query caching converged to their steady states. Figure 6.8 shows the measured expected accuracy of the Forward and Backward query

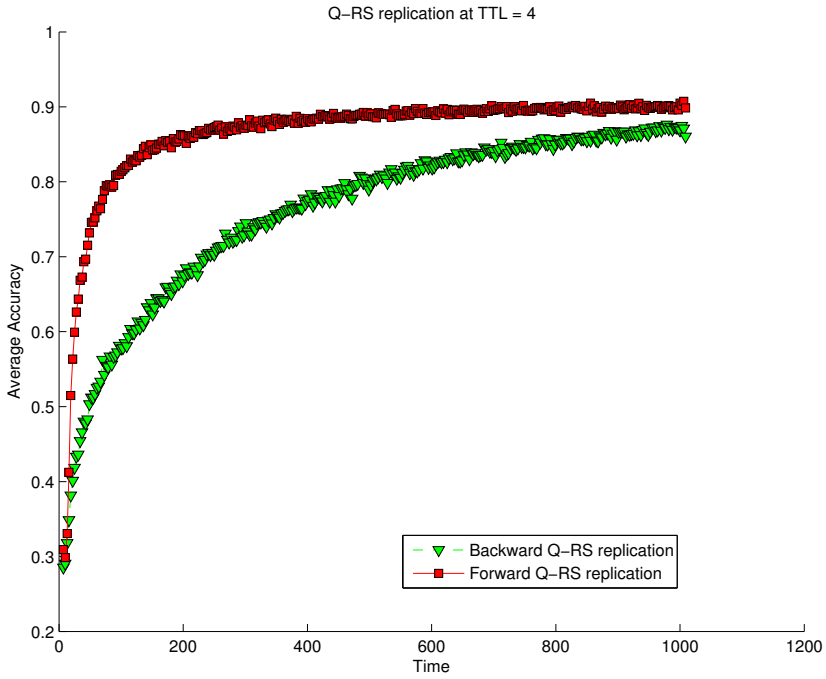

Figure 6.8: Measured expected accuracies as a function of time for the two query caching approaches.

caching as a function of time. Both approaches begin with an expected accuracy that is the same of the uniform distribution, as this is the initial distribution of documents. We observe that the Forward query caching converges to its steady-state within about 560 iteration. In contrast, Backward query caching is still continuing to improve after 1000 cycles, i.e. 1 million queries. However, despite the slow convergence, the performance of the algorithm is still better than proportional or square root document replication.

6.3 Summary

It is well-known that non-uniform replication of documents in a peer-to-peer network can improve retrieval performance. Previous work has focused on minimizing the expected search length and has considered both proportional and square-root replication policies. In this chapter we fix the search length and focus on search accuracy, which is defined as the overlap between the actual documents retrieved and those that would have been retrieved with an exhaustive search.

Implementations of proportional and square-root policies have focused on file sharing applications in which the purpose of replication is to both improve search and dissemination. From an information retrieval perspective, we believe it is useful to decouple search from dissemination. Thus it is not necessary to replicate the entire document. Instead, a peer can simply index the document together with a pointer to where the document can be downloaded. From an IR perspective, replicating the entire document or indexing the document are equivalent, but the latter requires significantly less storage. This is

advantageous as a peer is now able to index more documents, which aids retrieval.

This chapter proposed a replication policy based on replicating popular queries and their corresponding result sets. Thus, for example, a query consisting of the three terms "Hadrian wall Scotland" results in a three-term n-gram being inserted into a peer's local inverted index together with associated postings pointing to the top-k results. Replicating query-result sets requires much less storage than replicating documents. Typically, a document may have several hundreds unique terms, and a posting must be added to each term in the inverted index. If we think of the inverted index as a table in which the rows correspond to terms and the columns to documents, then document replication adds a column to the table with non-zero entries for each term in the document. In contrast, replicating query result set is equivalent to inserting a row in the table, if the n-gram terms of the query are not already present, and adding k postings for the top-k documents. Typically $k = 10$ and is therefore much smaller than the number of unique terms in a document.

Two implementations of query caching were considered. The first, Forward Query Caching, copies the final query result set obtained by the query issuer to those peers responding to the query. This incurs an additional communication overhead, but its performance is superior. The second, Backward Query Caching, copies the partial result sets obtained at each responding peer. This incurs no additional communication overhead. However, experimental results showed that the performance of the Backward algorithm was about 8% worse than for the Forward algorithm.

A simulation of query caching was performed on a network of 10,000 nodes. The underlying topology was random, with an average out-degree of 4. Each peer has local storage sufficient to index 0.1% of the total document collection. We used the WT10G database of 1.7 million documents and queries from an AOL query log. Experimental results showed very significant improvements in accuracy over a uniform document distribution, i.e. no replication, and both proportional and square-root replication. In particular, Forward Query Caching exhibited a 74% and 81% improvement over proportional and square root document replication and 240% over a uniform distribution. The corresponding improvements for Backward Query Caching were 70%, 77% and 232%, respectively.

For skewed query distributions, e.g. power law distributions, the expected accuracy can be high despite the fact that only a few popular queries have high accuracy, while the remaining queries exhibit low accuracy. We therefore also examined the cumulative percentage of queries above a given accuracy. Interestingly, the Forward and Backward Query Caching performed much better. For example 80.5% of queries had an accuracy greater than 0.7 for Forward Query Caching. In contrast, only 17.6 and 15.6 % of queries for proportional and square root had an accuracy of greater than 0.7.

There are a number of possible lines of future work. First, the accuracy of probably approximately correct (PAC) search is a function of both (i) the number of nodes queried, (ii) the number of times a document is replicated. Query-result set replication can be considered to increase both. Future work is

needed to better understand this interplay.

Second, our experimental results limited the storage available for caching. This constraint was determined based on the number of unique documents retrieved by each of the three query sets. Further work is needed to understand the relationship between performance and storage.

Chapter 7

Conclusion

This dissertation explored four specific issues in order to construct a low cost search engine based an unstructured distributed network. These are:

- A Probably Approximately Correct (PAC) search framework to mathematically model important stages in information retrieval on an unstructured distributed network.
- Adaptive node selection to iteratively improve centralized PAC search performance to be comparable to those of deterministic ones.
- Document replication policies that increase retrieval accuracy.
- Query caching techniques to improve search quality.

This final Chapter summarises the results presented in earlier parts of the dissertation, and then considering possible future directions for work arising from this thesis.

7.1 Results Summary

Chapter 3 examined the problem of non-deterministic search in which a set of computers (i) independently sample the collection/Web and (ii) queries are sent to a random subset of computers. Equations are derived for the expected coverage of the sample collection, and the accuracy of the retrieval results. The latter is measured with respect to the results provided by a deterministic IR system. Under the assumption that the deterministic system provides correct result, we consider the probability of being approximately correct. We therefore describe our approach as PAC search.

Our analysis of PAC IR in the context of commercial search engines suggest that a performance level of 63% can be achieved using the same amount of storage and computation. However, while the performance is lower, we believe that the PAC IR architecture may be simpler to manage. Moreover, more sophisticated implementations might close this performance gap.

PAC IR was also analyzed in the context of peer-to-peer decentralized web search. The key problem with such a configuration appears to be the much small storage available on any machine. Consequently,

it would be necessary to send the query to many more computers, and the communication overhead may then be too high.

Chapter 4 proposed adding a centralized query coordination node to the architecture. This configuration is not as distributed as the original PAC proposal. However, it retains much of its benefits, at the expense of introducing a single point of failure. Of course, in practice such a node could itself be replicated for fault tolerance and load balancing.

Using a centralized PAC architecture, and in response to a query, the random selection of nodes is replaced by a pseudo-random selection, seeded with the query. This has the advantage, as noted in Chapter 3 of eliminating the variation in results sets in response to the same query. More importantly, for frequently occurring queries, we considered the problem of iteratively refining the pseudo-random choice of z computers in order to improve the performance.

A theoretical analysis provided a proof that there exists (with near certainty) a configuration of z nodes on which all relevant documents will be found. The analysis also provided an upper bound on the performance of any iterative algorithm. The analysis also allowed us to estimate the expected number of relevant documents that should be present on any single machine. This information was then used to partly guide a heuristic search.

Two heuristic search algorithms were proposed. The first scored computers based simply on the number of relevant documents they retrieved. The second scored computers based on a nDCG-like score that gives higher weight to higher ranked documents.

Simulations showed that the search algorithm very closely followed the theoretical upper bound when the number of relevant documents was less than $k = 1000$. However, as the number increased to 2000 or 4000, deviation from the upper bound increased. Nevertheless, in all cases, retrieval performance continued to improve with each iteration. For $k = 4000$, the case where twice as many machines ($z = 2000$) were queried per iteration was examined. In this case, the initial accuracy is 86% and the search once again closely follows the theoretical upper bound. This suggests that when $z = 1000$, the iterative algorithm is retaining a sub-optimal choice of machines. Improvements to the heuristic search algorithms remains a source of on-going research.

Experiments on the TREC-8 dataset showed that the nDCG-based algorithm provided superior performance. In particular, the MAP score relative to a deterministic system increased from an initial 67% to 90% in just 4 iterations. And after 10 iterations performance is 96% of a deterministic system. These experiments suggests that, for frequently occurring queries, a centralized PAC architecture can perform at a similar level as a deterministic system.

Chapter 5 considered how documents should be replicated in order to maximize the retrieval accuracy. We showed that the well-known square root replication policy does not maximize accuracy, even though it minimizes the expected search size. We then defined the optimality criterion for maximizing

accuracy and provided an *approximate* optimal solution. We then compared the performance of four different replication policies, namely uniform, proportional, square root and optimal. Our comparisons examined how the expected accuracy varied as a function of (i) the query distribution, which was assumed to follow a power law with exponent α, and (ii) the search size, z. As expected, the optimum policy outperformed all other policies. The difference in performance between the optimum and square root policies increased as the search size decreased. A more detailed comparison of the two replication policies revealed that the optimum policy tended to replicate the most popular documents less frequently than the square root policy, while increasing the frequency of documents with middle popularity and decreasing the replication frequency of the least popular documents.

We also examined the variation in accuracy across queries. It was observed that all three non-uniform replication policies exhibited qualitatively similar characteristics - the standard deviation increases as α, the exponent of the query power law, increases from 0.1 to between 0.8 to 1. As α continues to increase, the standard deviation then decreases. We then argue that for a real world query log which query distribution has $\alpha = 0.7$, any non-uniform distribution of documents improves the retrieval performance of popular documents at the expense of less popular documents. To compensate for this, we propose a hybrid replication policy consisting of a combination of uniform and non-uniform distributions. Theoretical results show that such an arrangement significantly improves the accuracy of less popular documents at the expense of only a small degradation in accuracy averaged over all queries.

A simulation of search in an unstructured P2P network was then performed using the PeerSim simulator. Experimental results confirmed our theoretical analysis.

Chapter 6 studied the problem of query caching in PAC search architecture. Different from previous works, the main purpose of query caching in PAC is to improve retrieval quality within a specific budget of communication cost. We proposed a query caching approach based on replicating popular queries and their corresponding result sets. Replicating query result sets requires much less storage than replicating documents. Typically, a document may have several hundreds unique terms, and a posting must be added to each term in the inverted index. If we think of the inverted index as a table in which the rows correspond to terms and the columns to documents, then document replication adds a column to the table with non-zero entries for each term in the document. In contrast, replicating query result set is equivalent to inserting a row in the table, if the n-gram terms of the query are not already present, and adding k postings for the top-k documents. Typically k is much smaller than the number of unique terms in a document.

Two implementations of query caching were considered. The first, Forward Query Caching, copies the final query result set obtained by the query issuer to those peers responding to the query. This incurs an additional communication overhead, but its performance is superior. The second, Backward Query Caching, copies the partial result sets obtained at each responding peer. This incurs no additional

communication overhead. However, experimental results showed that the performance of the Backward algorithm was about 8% worse than for the Forward algorithm.

A simulation of query caching was performed on a network of 10,000 nodes. The underlying topology was random, with an average out-degree of 4. Each peer has local storage sufficient to index 0.1% of the total document collection. We used the WT10G database of 1.7 million documents and queries from an AOL query log. Experimental results showed very significant improvements in accuracy over a uniform document distribution, i.e. no replication, and both proportional and square-root replication. In particular, Forward Query Caching exhibited a 74% and 81% improvement over proportional and square root document replication and 240% over a uniform distribution. The corresponding improvements for Backward Query Caching were 70%, 77% and 232%, respectively.

For skewed query distributions, e.g. power law distributions, the expected accuracy can be high despite the fact that only a few popular queries have high accuracy, while the remaining queries exhibit low accuracy. We therefore also examined the cumulative percentage of queries above a given accuracy. Interestingly, the Forward and Backward Query Caching performed much better. For example 80.5% of queries had an accuracy greater than 0.7 for Forward Query Caching. In contrast, only 17.6 and 15.6 % of queries for proportional and square root had an accuracy of greater than 0.7.

7.2 Future Directions

The work in this dissertation studied the problems arising in constructing a information retrieval system over an unstructured distributed network. Though this is a firm step toward eliminating the barrier to developing low cost, large scale information retrieval systems, there are much to be done in the field of distributed information retrieval.

We have implicitly assumed that the deterministic and non-deterministic IR systems both implement the same underlying retrieval model. Usually, most retrieval models have parameter values that are based on the statistics of the collection. However, for the PAC IR system, each computer only has access to its local sample. Future work is needed to determine if, and under what conditions, the statistics of the local samples will be sufficiently close to the statistics of the overall collection.

The iterative approach presented in Chapter 4 is only applicable for queries that occur frequently (e.g. more than 10 times). For less frequently occurring queries, alternative approaches must be developed, which could be an interesting subject of future work.

The non-uniform document replication policies, namely the square-root document replication and the optimal document replication, are dependent on a central coordinating node to control the process of replication. This could introduce a single point of failure and is not favorable. In the future, it would be very helpful to extend our theoretical analysis to the case where each replication policy can be implemented with only local information available to each peer. Moreover, a loose controlled distributed

network is often challenged by the dynamic memberships of its peers, thus, understanding how each replication policy is affected by the churn of the network is an obvious future problem.

It is shown that the accuracy of probably approximately correct (PAC) search is a function of both (i) the number of nodes queried, (ii) the number of times a document is replicated. Query caching presented in Chapter 6 can be considered to increase both. Future work is needed to better understand this interplay.

Besides the aforementioned challenges, the practical implementation of the theory and simulation discussed in this thesis could demand more efforts in the transmission layer of the distributed networking. The communication cost and query latency must be carefully investigated to validate the feasibility of such an unstructured search architecture. It might be a good practice to develop a small scale prototype on a reliable test-bed to examine the practicability of important concepts.

We provide an alternative choice to the previous distributed architectures for information retrieval, and we expect that our work could inspire the research work in the related research fields, e.g. P2P file sharing, digital archiving and volunteer computing.

Appendix A

Math and Notations for PAC

In this appendix, we describe symbols used for the PAC architecture in this thesis.

- m: The size of the document collection.
- ρ: The capacity of each computer, i.e. the number of unique documents that each computer can store.
- $f = \frac{\rho}{m}$: The fraction of the collection available to each computer.
- n: The network size, i.e. the number of nodes (computers) in the distributed network.
- $c = n\rho$: The size of the collection sample.
- $\epsilon = (1 - \frac{\rho}{m})^n$: The probability a given document is not sampled by any computer. Approximately $\exp -\frac{c}{m}$.
- Y: The number of distinct documents in the collection sample.
- $C = \frac{Y}{m}$: Coverage, i.e. the fraction of the entire collection that is present in the collection sample.
- z: The search size, i.e. the number of computers queried during retrieval.
- $\epsilon' = (1-\frac{\rho}{m})^z$: The probability a given document is not sampled by any of the z queried computers.
- $\mathcal{K}$: The top-k document set retrieved by a deterministic search for a query.
- $\mathcal{K}''$: The top-k document set retrieved by a non-deterministic search for a query.
- $\mathcal{K}'$: The intersection of $\mathcal{K}$ and $\mathcal{K}''$.
- k: The number of documents in $\mathcal{K}$.
- k': The number documents in $\mathcal{K}'$.
- r: The average replication rate of a document.

- r_i: The replication rate of the ith document.
- $\mathcal{Q}$: The full query set.
- q_j: The jth query in the full query set $\mathcal{Q}$.
- q: A single query in the full query set $\mathcal{Q}$.
- a_j: The accuracy for query j.
- $A_j = 1 - \epsilon'$: The expected accuracy for query j.
- $A = \sum_j q_j A_j$: The global average accuracy for all queries in the full query set $\mathcal{Q}$.
- $E(C) = 1 - \epsilon$: The expected coverage of the entire collection.
- $Var(C) = \frac{\epsilon(1-\epsilon)}{m}\{1 - \frac{g(1-\epsilon}{g(f)}\}$: The variance of the coverage. For large m, $Var(C) \approx \frac{\epsilon(1-\epsilon)}{m}$.

Appendix B

Experimental Tools and Data

In this appendix, we describe the data collections and tools used in our experiments.

B.1 Datasets

The Text REtrieval Conference (TREC) [TRE] was started as part of the TIPSTER Text program, and is co-sponsored by the National Institute of Standards and Technology (NIST) and U.S. Department of Defense. It is a well recognized source of the standard infrastructure necessary for large-scale evaluation of text retrieval methodologies. TREC is held every year and provides a forum for presentations of the latest research results in the information retrieval. To assist such efforts, NIST releases standard collections of text documents on which research teams run retrieval experiments and report results. Besides document collections, every year TREC provides a set of topics (queries) that are used for experiments. TREC participants run their own retrieval systems on the document collection against these topics, and report to NIST a list of the top-ranked documents. NIST pools the individual results, and then hires assessors to map each query to a set of relevant documents in the collection thereby providing relevance judgements for evaluating IR systems.

B.1.1 TREC Disks 4 & 5

The collection of TREC disks 4 and 5 is primarily used for ad-hoc retrieval, and consists of 556,080 documents. Disk 4 includes material from the Financial Times Limited (1991, 1992, 1993, 1994), the Congressional Record of the $103rd$ Congress (1993), and the Federal Register (1994). Disk 5 includes material from the Foreign Broadcast Information Service (1996) and the Los Angeles Times (1989, 1990).

B.1.2 TREC Web Corpus : WT10g

TREC Web Corpus WT10g is a Web data collection targeting Web search experiments. It has following properties:

- 1,692,096 documents

- 11,680 servers
- an average of 144 documents per server
- a minimum of 5 documents per server
- 171,740 inter-server links (within the collection)
- 9977 servers with inter-server in-links (within the collection)
- 8999 servers with inter-server out-links (within the collection)
- 1,295,841 documents with out-links (within the collection)
- 1,532,012 documents with in-links (within the collection)

Compared to early test collections such as WT2g or VLC2, WT10g eliminates large quantities of redundant or duplicate data, and stores a larger number of inter-server links. By substantially increasing the size of the collection, WT10g has better support for distributed information retrieval experiments.

B.2 Tools

There are many choices of tools for information retrieval experiments. However, due to cross-platform requirements, we used tools based on the Java platform for our experiments.

B.2.1 Panda

Panda is a Java-based text retrieval platform developed by the Media Futures group at University College London. It has a flexible framework and can efficiently deal with medium-sized document collections such as TREC Disks 4 & 5. It supports popular ranking models in information retrieval, and also can be easily extended to support new ranking models. However, it does not support complex query models.

B.2.2 Terrier

Terrier [OAP+06] is a open source full-text search engine, developed at the School of Computing Science, University of Glasgow. Terrier implements state-of-the-art indexing and retrieval functionalities, and provides a robust, transparent, and modular framework for research and experimentation in information retrieval. It can be easily deployed on large-scale collections of documents, and allows the rapid development of large-scale retrieval applications. Especially, research on standard TREC and CLEF document collections can be easily carried out in Terrier. More information can be found on: http://terrier.org/.

B.2.3 PeerSim

PeerSim [MJ09] is an peer-to-peer simulation environment that supports dynamic scenarios such as churn and other failure models. It can significantly simplify the simulation of large-scale peer-to-peer

systems. It consists of two simulation engines, namely a simplified cycle-based one and an event-driven one. The engines are supported by many extendable and pluggable components, and have a flexible configuration mechanism. PeerSim is a very useful tool to evaluate a new protocol in its early stages of development before it being deployed in a real environment. More information can be found on: http://peersim.sourceforge.net/.

Appendix C

Optimization of Document Replication

From Equation 5.8, we seek to maximize

$$f(\mathbf{r}) = \sum_{i=1}^{m} q_i[1 - (1 - \frac{r_i}{n})^z] \tag{C.1}$$

where $\mathbf{r}$ denotes the vector of $(r_1, r_2, ..., r_m)$ of replications rates for each of the m documents. We would like to maximize this objective function subject to the following constraints

$$\sum_{i=1}^{m} r_i = R \tag{C.2}$$

$$1 \leq r_i \leq n \tag{C.3}$$

$$r_i \text{is integer} \tag{C.4}$$

where R is a positive constant representing the total amount of storage available in the network. We would like to find the optimal replication policy $\hat{\mathbf{r}} = argmax_{\mathbf{r}} f(\mathbf{r})$. This is an integer programming problem, and is known to be NP-hard [Sch86]. However, if we relax the constraint of Equation (C.4), i.e. we no longer require that documents are replicated an integer number of times, then we can convert the problem to be a convex optimization problem that is solvable in polynomial time. By relaxing this constraint, we obtain is an upper bound on the original integer programming problem. This give us helpful guidance on the performance we can expect from an optimal solution, and an approximation to the optimal can be achieved by rounding the r_i to their nearest integer values, taking care to ensure that the other constraints are obeyed.

The objective function (C.1) can be written as $f(\mathbf{r}) = f(\mathbf{x}) = \sum_{i=1}^{m} q_i(1 - x_i^z)$, where $x_i = 1 - \frac{r_i}{n}$. To maximize $f(\mathbf{x})$, we can minimize the derivative $f'(\mathbf{x}) = \sum_{i=1}^{m} q_i x_i^z$. We assume $\forall q_i, 1 \geq q_i \geq q_j \geq 0$, given $1 \leq i < j \leq m$. Hence, for a optimal solution, $\forall x_i, 0 \leq x_i \leq x_j \leq 1$ and thereby $\forall r_i, n \geq r_i \geq r_j \geq 1$. We further assume there exists two parameters b and b' such that $r_i = n$ when $1 \leq i \leq b$, i.e. the most popular b documents are replicated on each of all n nodes in the network. Further, for documents d_i,

such that $b < i < b'$ we assume $n > r_i > 1$ and for the least popular documents, d_i, where $b' \leq i \leq m$ we assume $r_i = 1$, i.e. there is one copy of each document in the network. Therefore, for $i \in (b, b')$, we transform the original integer programming problem to the equivalent convex optimization problem below:

$$\text{Maximize} \quad f(\mathbf{r}) = \sum_{i=b}^{b'} q_i[1 - (1 - \frac{r_i}{n})^z] \tag{C.5}$$

$$\text{Subject to} \quad g(\mathbf{r}) = \sum_{i=b}^{b'} r_i = R' \tag{C.6}$$

$$1 < r_i < n \tag{C.7}$$

where $R' = R - bn - m + b' - 1$ This nonlinear program problem can be solved using Lagrange multipliers [Bel56]. The standard auxiliary function is given by

$$\begin{aligned} \Lambda(\mathbf{r}, \lambda) &= f(\mathbf{r}) + \lambda(g(\mathbf{r}) - R) \\ &= \sum_{i=b}^{b'} q_i[1 - (1 - \frac{r_i}{n})^z] + \lambda(\sum_{i=b}^{b'} r_i - R') \end{aligned}$$

Solving for the case of $\nabla_{\mathbf{r},\lambda}\Lambda(\mathbf{r}, \lambda) = 0$

$$\begin{cases} \frac{zq_i}{n}(1 - \frac{r_1}{n})^{z-1} + \lambda = 0 \\ \frac{zq_i}{n}(1 - \frac{r_2}{n})^{z-1} + \lambda = 0 \\ \ldots \\ \frac{zq_i}{n}(1 - \frac{r_m}{n})^{z-1} + \lambda = 0 \\ \sum_{i=1}^{m} r_i - R' = 0 \end{cases}$$

We can infer that $\forall b < i < j < b', \frac{n-r_i}{n-r_j} = \frac{q_j^{\frac{1}{z-1}}}{q_i^{\frac{1}{z-1}}}$. Hence, $\forall b < i < b', n - r_i \propto q_i^{-\frac{1}{z-1}}$. Let $n - r_i = cq_i^{-\frac{1}{z-1}}$, then $r_i = n - cq_i^{-\frac{1}{z-1}}$. Substituting r_i into $\sum_{i=b}^{b'} r_i = R'$ we obtain $c = \frac{n(b'-b-1)-R'}{\sum_{b+1}^{b'-1} q_j^{-\frac{1}{z-1}}}$. This, the optimal solution for $\mathbf{r}$ is $\hat{\mathbf{r}}$ where

$$r_i = n - \frac{n(b' - b - 1) - R'}{\sum_{b+1}^{b'-1} q_i^{-\frac{1}{z-1}}} q_i^{-\frac{1}{z-1}} \tag{C.8}$$

Note the available values for b and b' are also bounded. Any b must satisfy

$$bn + (m - b) < R \tag{C.9}$$

$$b < \frac{R - m}{n - 1} \tag{C.10}$$

And any b' must satisfy

$$m - b' + 1 + (b' - 1)n \quad > \quad R \tag{C.11}$$

$$b' > \frac{R - m}{n - 1} + 1 \tag{C.12}$$

For any b and b' satisfying the above constraints, it is easy to prove that any r_i obtained in Equation (C.8) is guaranteed to be smaller than n, thus, we drop the parameter b ($b = 0$), and rewrite the optimal solution (C.8) as

$$r_i = n - \frac{n(b' - 1) - R'}{\sum_1^{b'-1} {q_i}^{-\frac{1}{z-1}}} {q_i}^{-\frac{1}{z-1}} \tag{C.13}$$

where $R' = R - m + b' - 1$

Since $r_i > 1$, the optimal solution must satisfy

$$n - \frac{n(b'-1) - R'}{\sum_1^{b'-1} {q_i}^{-\frac{1}{z-1}}} {q_i}^{-\frac{1}{z-1}} > 1 \tag{C.14}$$

$$q_i > \left(\frac{n(b'-1) - R'}{(n-1)\sum_1^{b'-1} {q_i}^{-\frac{1}{z-1}}} \right)^{z-1} \tag{C.15}$$

Since we assume that q_i is monotonically decreasing, the smallest q_i in the solution is $q_{(b'-1)}$. Thus, we require that

$$q_{(b'-1)} \quad > \quad \left(\frac{n(b' - 1) - R'}{(n - 1)\sum_1^{b'-1} {q_i}^{-\frac{1}{z-1}}} \right)^{z-1} \tag{C.16}$$

The right side of Equation (C.16) monotonically decreases as b' decreases, while the left side, $q_{(b'-1)}$ monotonically increases as b' decreases. As a result, there must be a boundary $\hat{b'}$, such that $\forall b' \leq \hat{b'}$, Equation (C.16) holds, and $\forall b' > \hat{b'}$, Equation (C.16) fails. This $\hat{b'}$ can be found by simple binary search. Any solution given by a b' which is less than $\hat{b'}$, it must be covered by the solution given by $\hat{b'}$. Consequently, $\hat{b'}$ yields the true optimal solution. A search for $\hat{b'}$ takes $O(\log m)$ time, and the computation in each search takes $O(m)$ time, so the computation complexity is $O(m \log m)$.

Appendix D

Publications

1. I. J. Cox, R. Fu, and L. K. Hansen. Probably approximately correct search. In Proc. of the International Conference on Theoretical Information Retrieval (ICTIR), 2009.

2. I. J. Cox, J. Zhu, R. Fu, and L. K. Hansen. Improving query correctness using centralized probably approximately correct (pac) search. In Proc. of 32nd European Conference on Information Retrieval (ECIR), 2010.

3. H. Asthana, R. Fu, and I. Cox. On the feasibility of unstructured peer-to-peer information retrieval. Advances in Information Retrieval Theory, pages 125-138, 2011.

Bibliography

[ACK+02] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[And04] D.P. Anderson. BOINC: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

[aol] Aol query log. `http://www.gregsadetsky.com/aol-data/`.

[BCG+03] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi. Efficient peer-to-peer searches using result-caching. *Peer-to-Peer Systems II*, pages 225–236, 2003.

[BDH03a] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The google cluster architecture. *Micro, IEEE*, 23(2):22–28, 2003.

[BDH03b] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[Bel56] R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[BGK+09] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.

[BHK+09] E. Baykan, M. Henzinger, S.F. Keller, S. De Castelberg, and M. Kinzler. A comparison of techniques for sampling web pages. *Arxiv preprint arXiv:0902.1604*, 2009.

[BJC+04] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, David A. Grossman, and Ophir Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR*, pages 321–328, 2004.

[Blo70] B. H. Bloom. Space/time trade-off in hash coding with allowable errors. In *Communications of the ACM*, page 13(7), 1970.

[BMT+05a] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in p2p search engines. In *SIGIR'05: Proceedings of the 28th International ACM SIGIR conference on Research and Development in Information Retrieval, Salvador, Brazil*, 2005.

[BMT+05b] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: collaborative p2p search. In *VLDB'05: Proceedings of the 31st International conference on Very Large Data Bases, Trondheim, Norway*, 2005.

[BS04] S.A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *Arxiv preprint cs/0412017*, 2004.

[BYGJ+07] Ricardo A. Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. The impact of caching on search engines. In *SIGIR*, pages 183–190, 2007.

[Cal00] J. Callan. Distributed information retrieval. In *Advances in Information Retrieval*, pages 127–150, 2000.

[CAPMN03] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *HPDC'03: Proceedings of the 12th International Symposium on High Performance Distributed Computing, Seattle, WA, USA*, 2003.

[CFH09] Ingemar J. Cox, Ruoxun Fu, and Lars Kai Hansen. Probably approximately correct search. In *Proc. of the International Conference on Theoretical Information Retrieval (ICTIR)*, 2009.

[CG96] S.F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[Cle93] C. Cleverdon. The Cranfield tests on index language devices. In *Aslib proceedings*, volume 19, pages 173–194. MCB UP Ltd, 1993.

[Coh03] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72. Citeseer, 2003.

[CS02] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 177–190. ACM, 2002.

[CSWH01] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[CZFH10] Ingemar J. Cox, Jianhan Zhu, Ruoxun Fu, and Lars Kai Hansen. Improving query correctness using centralized probably approximately correct (pac) search. In *Proc. of 32nd European Conference on Information Retrieval (ECIR)*, 2010.

[Dea09] J. Dean. Challenges in building large scale information systems. In *Keynote Presentation at ACM WSDM*, 2009.

[DLAV] W.K. Dedzoe, P. Lamarre, R. Akbarinia, and P. Valduriez. ASAP Top-k Query Processing in Unstructured P2P Systems. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10. IEEE.

[dMdSF+05] E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving Web search efficiency via a locality based static pruning method. In *Proceedings of the 14th international conference on World Wide Web, May 10-14, 2005, Chiba, Japan*, 2005.

[FFM04] M.J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with Coral. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1*, page 18. USENIX Association, 2004.

[FPSO06] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Transactions on Information Systems (TOIS)*, 24(1):51–78, 2006.

[GCX+05] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, page 4. USENIX Association, 2005.

[gnu] http://en.wikipedia.org/wiki/gnutella.

[goo06] N. patel. learning from googles data centers. http://www.pronetadvertising.com/, 2006.

[goo10] The google search query - a technical look. http://www.webmasterworld.com/google/3694079.htm, 2010.

[HHH+02] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *1st Int. Workshop on Peer-to-Peer Systems(IPTPS02)*, 2002.

[JC08] H. Jin and H. Chen. Semrex: Efficient search in a semantic overlay for literature retrieval. In *Future Generation Computer Systems*, page 24(6), 2008.

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[JMB05] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[JMJV] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[KB+] Y. Kulbak, D. Bickson, et al. The emule protocol specification. *eMule project, http://sourceforge. net*.

[KS04] V. King and J. Saia. Choosing a random peer. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 125–130. ACM, 2004.

[LC03] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *CIKM'03: Proceedings of the 12th International conference on Information and Knowledge Management, New Orleans, LA, USA*, 2003.

[LC05] J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *ECIR'05: Proceedings of the 27th European conference on IR Research, Santiago de Compostela, Spain*, 2005.

[LC06] J. Lu and J. Callan. User modeling for full-text federated search in peer-to-peer networks. In *SIGIR'06: Proceedings of the 29th International ACM SIGIR conference on Research and Development in Information Retrieval, Seattle, WA, USA*, 2006.

[LIJM+10] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, August 30-September 03, 2010, New Delhi, India*. ACM, 2010.

[LLH+03] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Krager, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In F. Kaashoek and I. Stoica, editors, *Int. Workshop on Peer-to-Peer Systems (IPTPS)*, volume LNCS 2735, pages 207–215. Springer Lecture Notes in Computer Science, 2003.

[Lu07] J. Lu. Full-text federated search in peer-to-peer networks. In *PhD thesis, Carnegie Mellon, University, USA*, 2007.

[min] http://dtc.umn.edu/mints/.

[MJ09] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, 2009.

[MRS09] C. D. Manning, P. Raghavan, and H. Schtze. *An Introduction to Information Retrieval*. Cambridge University Press, 2009.

[OAP+06] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[PC98] J.M. Ponte and W.B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01: Proceedings of the International conference on Distributed Systems Platforms, Heidelberg, Germany*, 2001.

[RD10] R. Rodrigues and P. Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, 2010.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM'01: Proceedings of the ACM SIGCOMM conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, San Diego, CA, USA*, 2001.

[RHHR09a] C. Raiciu, F. Huici, M. Handley, and D.S. Rosenblum. Roar: increasing the flexibility and performance of distributed search. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 291–302. ACM, 2009.

[RHHR09b] Costin Raiciu, Felipe Huici, Mark Handley, and David S. Rosenblum. Roar: increasing the flexibility and performance of distributed search. *SIGCOMM Comput. Commun. Rev.*, 39(4):291–302, 2009.

[Rob93] S.E. Robertson. The probability ranking principle in IR. *Journal of documentation*, 33(4):294–304, 1993.

[RPLG01] P. Rusmevichientong, D.M. Pennock, S. Lawrence, and C.L. Giles. Methods for sampling pages uniformly from the world wide web. In *AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001.

[RV03] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of the International Middleware Conference*, 2003.

[RWJ+96] S. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Proc. of the Third Text REtrieval Conference (TREC 1994)*, pages 109–126, 1996.

[Sch86] A. Schrijver. *Theory of linear and integer programming*, volume 11. Wiley New York, 1986.

[Sea09] http://news.ebrandz.com/google/2009/2495-google-continues-to-lead-february-2009-us-search-engine-rankings-comscore-.html, 2009.

[SGG+02] S. Saroiu, P.K. Gummadi, S.D. Gribble, et al. A measurement study of peer-to-peer file sharing systems. In *proceedings of Multimedia Computing and Networking*, volume 2002, page 152. Citeseer, 2002.

[SGG03] S. Saroiu, K.P. Gummadi, and S.D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia systems*, 9(2):170–184, 2003.

[SHMM99] Craig Silverstein, Monika Rauch Henzinger, Hannes Marais, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[SLZ+07] G. Skobeltsyn, T. Luu, I. P. Zarko, M. Rajman, and K. Aberer. Web text retrieval with a p2p query-driven index. In *SIGIR*, pages 679–686, 2007.

[SMk+01] I. Stoica, R. Morris, D. karger, F. Kaashoek, and H. Balakrishnan. Chord: Scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[SWY75] G. Salton, A. Wong, and C.S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[THT11] A. Tigelaar, D. Hiemstra, and D. Trieschnigg. Search result caching in peer-to-peer information retrieval networks. *Multidisciplinary Information Retrieval*, pages 134–148, 2011.

[TKLB07] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P. Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM*, pages 49–60, 2007.

[TLB+07] W.W. Terpstra, C. Leng, A.P. Buchmann, et al. Bubblestorm: Analysis of probabilistic exhaustive search in a heterogeneous peer-to-peer system. Technical report, Technical Report TUD-CS-2007-2, Technische Universität Darmstadt, Germany, 2007.

[TRE] Text retrieval conference (trec). `http://trec.nist.gov/`.

[TXM02] C. Tang, Z. Xu, and M. Mahalingam. psearch: Information retrieval in structured overlays. In *HotNets-I*, 2002.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[VGLN06] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt. Mapping the PPLive network: Studying the impacts of media streaming on P2P overlays. 2006.

[web09] http://www.worldwidewebsize.com/, 2009.

[WMB99] I. J. Witten, A. Moffat, and T. C. Bell. *Managing Gigabvtes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2nd edition, 1999.

[WPP+04] L. Wang, K.S. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, page 14. USENIX Association, 2004.

[www] http://www.worldwidewebsize.com/.

[XC99] J. Xu and B. Croft. Cluster-based language models for distributed retrieval. In *SIGIR'99: Proceedings of the 22nd International ACM SIGIR conference on Research and Development in Information Retrieval, Berkeley, CA, USA*, 1999.

[YA06] E. Yilmaz and J.A. Aslam. Estimating average precision with incomplete and imperfect judgments. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 102–111. ACM, 2006.

[YDRC06] Y. Yang, R. Dunlap, M. Rexroad, and B.F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. In *IEEE INFOCOM*, pages 2658–2669, 2006.

[YH06] K.H. Yang and J.M. Ho. Proof: A dht-based peer-to-peer search engine. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 702–708. IEEE Computer Society, 2006.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *Technical report, Berkeley, CA, USA*, 2001.

[ZL04] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[ZLLY05] X. Zhang, J. Liu, B. Li, and T.S.P. Yum. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *proceedings of IEEE Infocom*, volume 3, pages 13–17. Citeseer, 2005.