Probably Approximately Correct Search

Ingemar J. Cox[1], Ruoxun Fu[1], and Lars Kai Hansen[2]

[1] University College London
[2] Technical University of Denmark
ingemar@cs.ucl.ac.uk

Abstract. We consider the problem of searching a document collection using a set of independent computers. That is, the computers do *not* cooperate with one another either (i) to acquire their local index of documents or (ii) during the retrieval of a document. During the acquisition phase, each computer is assumed to randomly sample a subset of the entire collection. During retrieval, the query is issued to a random subset of computers, each of which returns its results to the query-issuer, who consolidates the results. We examine how the number of computers, and the fraction of the collection that each computer indexes, affects performance in comparison to a traditional deterministic configuration. We provide analytic formulae that, given the number of computers and the fraction of the collection each computer indexes, provide the probability of an approximately correct search, where a "correct search" is defined to be the result of a deterministic search on the entire collection. We show that the randomized distributed search algorithm can have acceptable performance under a range of parameters settings. Simulation results confirm our analysis.

1 Introduction

Searching the Web is critical to the Web's success. Search is now common - Americans alone are estimated to have performed over 13 billion searches in February 2009 [9]. 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 [16].

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 computer architectures have been proposed. Commercial search engines such as Google, use an architecture where the the index is distributed (and arguably "virtually centralized") over a number of disjoint partitions [1]. 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 [6]. 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 [6]. Note that while the index is distributed across machines, the machines themselves are typically housed within a central server facility.

L. Azzopardi et al. (Eds.): ICTIR 2009, LNCS 5766, pp. 2–16, 2009.

Peer-to-peer networks 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. Li *et al.* [5] 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.

In this paper, 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 [2,8]. The random sampling of computers within a network is straightforward when the computers are a part of a "centralized" cluster. And recent work [4,14,13], based on distributed hash tables, provides algorithms for choosing a random peer within a peer-to-peer network.

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 [15].

In Section 2 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 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 4 provides simulation results that support the previous theoretical analysis. Finally, Section 5 summarizes our results and suggests avenues for future work.

2 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 2.1, we first define a variety of terms and concepts. Section 2.2 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 2.3 to derive an analytic model for the correctness of a PAC IR system.

2.1 Definitions

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

Let K represent the total number of computers available to perform searches. Note that in the case of peer-to-peer networks, K is not constant. However, in such a case, let us assume K 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 n unique documents, which form an individual sample from the collection. We assume that $n \leq N$, and, in practice, normally $n \ll N$. 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, k', of computers, and the union of their indices is known as the "retrieval index".

2.2 Sampling the Collection

In order to index the N distinct documents, the K 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 2.3), 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 N documents in the collection by each of the K computers is analogous to having an urn with N labeled balls. Each of K people, then randomly choose n balls each. An individual chooses his/her n balls without replacement, thereby guaranteeing that there is no repetition on a single computer. After indexing the n 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 K people have each randomly picked n balls? The answer to this question determines the coverage obtained after all K computers have sampled n documents.

Obviously, the coverage ranges from $\frac{n}{N}$ in the worse case, where all computers sample the same set of n documents, to $\frac{\min(N,Kn)}{N}$ 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 $\frac{n}{N}$, and the probability of *not* being picked is therefore $1 - \frac{n}{N}$. Thus the probability, $P(\bar{d}_i)$, that document d_i will *not* be picked by *any* of the K people is

$$P(\bar{d}_i) = \left(1 - \frac{n}{N}\right)^K \tag{1}$$

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

$$P(d_i) = 1 - P(\bar{d}_i) = 1 - \left(1 - \frac{n}{N}\right)^K \tag{2}$$

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

$$\hat{N} = P(d_i)N = \left(1 - \left(1 - \frac{n}{N}\right)^K\right) N \tag{3}$$

To simplify our further analysis, let us now set

$$\epsilon = P(\bar{d}_i) = \left(1 - \frac{n}{N}\right)^K \tag{4}$$

Then

$$\hat{N} = N(1 - \epsilon). \tag{5}$$

And the expected coverage can be defined as

$$E(Coverage) = \frac{\hat{N}}{N} = 1 - \epsilon. \tag{6}$$

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

We can use Equation (4) - Equation (6) to determine the coverage. We are interested in the relationship between coverage, the size of the individual sample, n, and the number of computers, K, given a certain collection size, N. 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 $c = Kn$. The collection sample, c, is treated as a constant in the following analysis. Also to simplify our analysis, let us first assume that $c \leq N$. From Equation (4), we have

$$\epsilon = \left(1 - \frac{n}{N}\right)^K = \left(1 + \frac{-\frac{c}{N}}{K}\right)^K \tag{7}$$

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

From the property of exponential functions, we know that

$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n \tag{8}$$

From Equations (7) and (8), we have that

$$\lim_{K \to \infty} \epsilon = \lim_{K \to \infty} \left(1 + \frac{-\frac{c}{N}}{K}\right)^K = e^{-\frac{c}{N}} \tag{9}$$

Next, the derivative of ϵ with respect to K is

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

From the property of natural logarithms, we also have

$$\ln(1 + h) \geq (\frac{h}{1 + h}), \text{for } h \geq -1 \tag{10}$$

Since

$$n \leq N \Rightarrow nK \leq NK \Rightarrow -\frac{c}{NK} = -\frac{nK}{NK} \geq -1$$

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

$$\frac{\partial \epsilon}{\partial K} = \epsilon\left(\ln\left(1+\left(-\frac{c}{NK}\right)\right) - \left(\frac{-\frac{c}{NK}}{1+(-\frac{c}{NK})}\right)\right) \geq 0 \tag{11}$$

Combining Equations (9) and (11), we show that the ϵ increases monotonically with an upper bound of $e^{-\frac{c}{N}}$, as K increases. Thus, the expected coverage ranges from $(1-e^{-\frac{c}{N}}$ to $\frac{c}{N}]$. Figure 1 plots ϵ and coverage for the case where $N = 1000000$ and $c = N$.

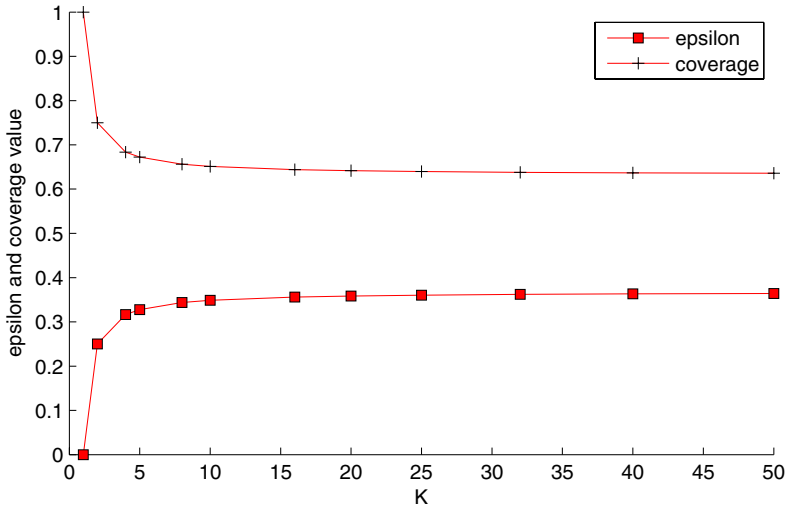

Fig. 1. Simulation calculating ϵ and coverage as a function of the number of computers, K, when the number of computers, $N = 1000000$, and the collection sample is $c = N$

This monotonic property remains true when we relax the assumption that $c \leq N$, and allow $c > N$, provided $n \leq N$. In this case, K cannot start from 1 since it would imply that $n > N$. Let us define K_{min} as the smallest value of K such that the property $n \leq N$ is maintained. Then, a more general form of coverage can be written as $(1-e^{-\frac{c}{N}}, 1-(1-\frac{c}{NK_{min}})^{K_{min}}]$.

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

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 K promises a larger coverage, it results in a larger individual sample, n.

Let us define k' 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

$$T = \frac{K}{k'} \times \frac{p}{n} \tag{12}$$

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

$$T = \frac{K^2 \times p}{k' \times c} \propto K^2 \tag{13}$$

Obviously larger K increases the query throughput, but, as we discussed earlier, a larger K decreases coverage, when our sample collection size, c, is fixed. Thus, for a given c, choosing appropriate values of K and k' 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, N. Using Equation 9, we can easily infer that ϵ tends to be $e^{-1} = 0.367$ with a large K. Inserting this value in Equation (6), we see that when we sample N 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 (9) we have

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

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

2.3 Retrieval

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

When performing retrieval within such an architecture, we wish to send the query to k' randomly chosen nodes, where $k' \leq K$, and normally $k' \ll K$. 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 k' machines, we cannot guarantee the coverage provided by all K machines. However, Equation (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 K with k'.

$$\hat{N'} = P(d'_i)N = \left(1 - \left(1 - \frac{n}{N}\right)^{k'}\right) N \tag{14}$$

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

$$P(d'_i) = 1 - \left(1 - \frac{n}{N}\right)^{k'} \tag{15}$$

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-r documents from the deterministic system, what is the probability that our PAC IR system will retrieve r' documents from r, where $r' \leq r$.

We know from the Equation (15) that the probability of a specific document being present in the result set is $P(d'_i)$. Thus, the probability of exactly r' documents from r being present in the result set is

$$P(r') = \binom{r}{r'} P(d'_i)^{r'}(1 - P(d'_i))^{r-r'} \tag{16}$$

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

$$E(r') = rP(d'_i) \tag{17}$$

Equation (17) 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 two practical applications.

3 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 [1]. Finally a variety of peer-to-peer decentralized architectures have been proposed and deployed [11,12,3,7,18,17,10] with a variety of search capabilities. In the following two subsections we examine the second case and the third case, respectively.

3.1 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 [1]) contains a "randomly chosen subset of documents from the full index" [1]. 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 [6].

In [6] it is claimed that Google partitions its index into 1000 disjoint sets. Thus, the number of documents indexed by a single machine is $\frac{n}{N} = \frac{1}{1000}$. It is further claimed that the data in any partition is replicated over 300 machines, so the total number of machines is $K = 300,000$, and the total number of samples is $Kn = 300N$. 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 K machines, independently samples 0.1% of the Web. Solving for ϵ in Equations (7) and (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, $k' = 1000$ and $\frac{n}{N} = \frac{1}{1000}$, as before. Substituting in Equation (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.

Table 1. The probability of exactly r' documents being present in the top-10

r'	$P_r(d_1 \cdots d_{r'})$
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

Assuming we are primarily interested in the top-10 results, i.e. $r = 10$, and given $P(d'_i) = 0.63$, substituting in Equation (17), gives

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

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 (16) to calculate the probabilities for all possible r'. These probabilities are enumerated in Table 1 for $r = 10$ and $0 \leq r' \leq 10$.

Table 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.

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 Peer-to-Peer Decentralized Search

Another possible implementation of PAC IR is in peer-to-peer decentralized search. Following the estimation data in [5], suppose we have a peer-to-peer net-

work with one million machines ($K = 10^6$). Let us further assume that every machine can provide 1GB for storing the index. If every document has, on average, 1000 distinct terms, and each term posting requires 20 bytes, then every document consumes 20k bytes in the index, and each machine can therefore index 50k documents ($n = 5 \times 10^4$). Thus the whole network has $Kn = 5 \times 10^{10}$ documents.

Now let us consider the case where we wish the peer-to-peer PAC IR system to index 17 billion documents, which is the same of the estimated size of Google's collection. Thus, the coverage obtained by the collection sampling is

$$E(Coverage) = 1 - \left(1 - \frac{5 \times 10^4}{1.7 \times 10^{10}}\right)^{10^6} = 0.947$$

This is not particularly surprising given that the size of our collection sample, Kn, is about 3 times the collection size.

During retrieval we must once again transmit the query to only a subset of the 1 million machines. If we assume that the query is sent to 10000 machines [5], then $k' = 10000$, and we have

$$P(d_i') = 1 - \left(1 - \frac{5 \times 10^4}{1.7 \times 10^{10}}\right)^{10000} = 0.03$$

The expected number of documents common to the top-10 generated by a deterministic search is then $E(r') = 10 \times 0.03 = 0.3$, i.e. on average, there is less than one document in common.

It is tempting to assume that this poor performance is due to the random nature of PAC IR. However, if we consider the expected number of distinct documents in a random selection of 10000 machines, we have

$$n_{distinct} = \left(1 - \left(1 - \frac{5 \times 10^4}{1.7 \times 10^{10}}\right)^{10000}\right) \times (1.7 \times 10^{10}) \approx 4.93 \times 10^8$$

In comparison, if each machine sample is disjoint from one another, we have $5 \times 10^4 \times 10^4 = 5 \times 10^8$ distinct documents. Thus, the coverage provided by the random sampling is $\frac{4.99}{5} = 98.6\%$ of the best possible coverage.

In fact, the root cause of the poor performance is due to the low capacity of each machine. If we wish to reach a PAC performance of 63%, we need to query $340,000$ machines.

This conclusion can also be reached in the Bubble Storm algorithm[13], in which the probability of a query meeting a document is $1 - e^{-\frac{k'g}{K}}$, where k' is the query replication number, g is the document replication number and K is the total number of machines in the network. Following the estimation data above, the average document replication number is $g = \frac{Kn}{N} \approx 3$, $k' = 10000$ and $K = 10^6$. So the probability of a query meeting a document is $1 - e^{-\frac{3 \times 10000}{10^6}} = 0.03$, which is similar to the result of our PAC IR algorithm. And, of course, too low to be practical.

The simple solution to the problem is to increase each machine's capacity. Suppose each machine can provide 340GB for storing data, then

$$P(d_i') = 1 - \left(1 - \frac{5 \times 10^4 \times 340}{1.7 \times 10^{10}}\right)^{1000} = 0.63$$

Thus $E(r') = 10 \times 0.63 = 6.3$. However, it seems unlikely that most peers can provide this storage requirement.

4 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 (n), number of machines (K) and collection size (N). In the first simulation, we manually generated a collection with $1e+6$ documents ($N = 1e+6$), and set $n = 1000, K = 1000$. This synthetic collection was simply a set of document identifiers. In each trial, each of the K machines samples n 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 K 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. $N = 556079$. All other settings are set the same as for the first simulation.

The results from Tables 2 and 3 show that the variation across trials is very small. This is very encouraging and supports our analysis in section 2, 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.

Table 2. Comparison of expected coverage, average coverage and standard deviation across 20 trials

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

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

5 Conclusion

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 k 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.

Acknowledgements

The authors thank Vishwa Vinay of Microsoft Research, Cambridge, and Brad Karp, David Rosenblum and Jun Wang of UCL for useful discussion on earlier drafts of this paper.

References

1. Barroso, L.A., Dean, J., Holzle, U.: Web search for a planet: The google cluster architecture. IEEE Micro. 23(2), 22–28 (2003)
2. Baykan, E., de Castelberg, S., Henzinger, M.: A comparison of techniques for sampling web pages. In: Unknow, vol. 1000,
3. Harren, M., Hellerstein, J.M., Huebsch, R., Loo, B.T., Shenker, S., Stoica, I.: Complex queries in dht-based peer-to-peer networks. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 242. Springer, Heidelberg (2002)
4. King, V., Saia, J.: Choosing a random peer. In: PODC, pp. 125–130 (2004)
5. Li, J., Loo, B.T., Hellerstein, J.M., Kaashoek, M.F., Krager, D.R., Morris, R.: On the feasibility of peer-to-peer web indexing and search. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 207–215. Springer, Heidelberg (2003)
6. Raiciu, C., Huici, F., Handley, M., Rosenblum, D.: ROAR: Increasing the flexibility and performance of distributed search. In: Proc. ACM SIGCOMM 2009 Conference on Data Communication, SIGCOMM 2009 (2009)
7. Reynolds, P., Vahdat, A.: Efficient peer-to-peer keyword searching. In: Proceedings of the International Middleware Conference (2003)
8. Rusmevichientong, P., Pennock, D.M., Lawrence, S., Giles, C.L.: Methods for sampling pages uniformly from the world wide web. In: Proc. AAAI Fall Symposium on Using Uncertainty Within Computation, pp. 121–128 (2001)
9. `http://news.ebrandz.com/google/2009/2495-google-continues-to-lead-february-2009-us-search-engine-rankings-comscore-.html` (2009)
10. Skobeltsyn, G., Luu, T., Zarko, I.P., Rajman, M., Aberer, K.: Web text retrieval with a p2p query-driven index. In: SIGIR, pp. 679–686 (2007)
11. Stoica, I., Morris, R., karger, D., Kaashoek, F., Balakrishnan, H.: Chord: Scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference, pp. 149–160 (2001)

12. Tang, C., Xu, Z., Mahalingam, M.: psearch: Information retrieval in structured overlays. In: HotNets-I (2002)
13. Terpstra, W.W., kangasharju, J., Leng, C., Buchmann, A.P.: Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In: SIGGCOMM 2007 (2007)
14. Terpstra, W.W., Leng, C., Buchmann, A.P.: Bubblestorm: Analysis of probabilistic exhaustive search in a heterogeneous peer-to-peer system. In: Technical Report TUD-CS-2007-2 (2007)
15. Valiant, L.G.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)
16. `http://www.worldwidewebsize.com/` (2009)
17. Yang, K.-H., Ho, J.-M.: Proof: A dht-based peer-to-peer search engine. In: Conference on Web Intelligence, pp. 702–708 (2006)
18. Yang, Y., Dunlap, R., Rexroad, M., Cooper, B.F.: Performance of full text search in structured and unstructured peer-to-peer systems. In: INFOCOM (2006)