Improving query correctness using centralized probably approximately correct (PAC) search

Ingemar Cox[1], Jianhan Zhu[1], Ruoxun Fu[1], and Lars Kai Hansen[2]

[1] University College London
[2] Technical University of Denmark
{i.cox, j.zhu, r.fu}@cs.ucl.ac.uk, lkh@imm.dtu.dk

Abstract. A non-deterministic architecture for information retrieval, known as probably approximately correct (PAC) search, has recently been proposed. However, for equivalent storage and computational resources, the performance of PAC is only 63% of a deterministic system. We propose a modification to the PAC architecture, introducing a centralized query coordination node. To respond to a query, random sampling of computers is replaced with pseudo-random sampling using the query as a seed. Then, for queries that occur frequently, this pseudo-random sample is iteratively refined so that performance improves with each iteration. A theoretical analysis is presented that provides an upper bound on the performance of any iterative algorithm. Two heuristic algorithms are then proposed to iteratively improve the performance of PAC search. Experiments on the TREC-8 dataset demonstrate that performance can improve from 67% to 96% in just 10 iterations, and continues to improve with each iteration. Thus, for queries that occur 10 or more times, the performance of a non-deterministic PAC architecture can closely match that of a deterministic system.

1 Introduction

High query rates together with a very large collection size combine to make web search computationally challenging. To meet this challenge commercial search engines use a centralized distributed architecture in which the index is disjointly partitioned across a number of clusters [2]. Within each cluster, the partial index is then replicated across all machines in the cluster. When a query is received, the query is forwarded to a single machine in each partition/cluster, and the results are then consolidated before transmitting the retrieved results to the user.

This centralized distributed architecture guarantees that the entire index is searched. Moreover, for a fixed number of machines (fixed computational and storage budget), the number of partitions and the number of computers per partition (replication) can be altered in order to ensure that queries are responded to with low latency. This distributed architecture works well when the data set and the query rate do not change frequently. Nevertheless, repartitioning and replication cannot be avoided, and the procedure can be time consuming and expensive. For example, in [9], it is reported that Google must take half of their machines offline during this process and that terabytes of data must be copied between machines.

A key characteristic of a centralized distributed architecture is that it is deterministic, i.e. every query is compared to all documents in the collection and multiple instances of the same query generate the same result set. Recently a non-deterministic architecture has been proposed [4], called probably approximately correct (PAC) search. In the PAC search architecture, it is assumed that (i) the computers are independent, i.e. there is no communication between computers, (ii) each computer locally stores a random subset or partition of the index, (iii) the partitions are *not* disjoint, i.e. documents indexed on one machine may also be indexed on other machines, but there is no replication of documents within a single machine, (iv) a query is sent to a random subset of computers and the results from each machine are then consolidated before being displayed to the user.

A PAC search architecture is non-deterministic because (i) only a random subset of the index is searched in response to a query, and (ii) multiple instances of the same query may generate different result sets. The correctness of a PAC search is defined with respect to the result set achieved using a deterministic search. That is, it is assumed that both architectures implement the same retrieval model, and that, for each query, it is desired to approximate the result set that would be returned with a deterministic implementation of this retrieval model. Given the random nature of PAC search, the results are approximate and probabilistic, hence probably approximately correct search. Note that this definition of correctness does *not* incorporate traditional measure of precision and recall. Precision and recall are considered to be *only* a function of the retrieval model. The purpose of the PAC search is to approximate the results of this retrieval model as closely as possible using a non-deterministic search architecture. The correctness of a PAC search is measured by retrieval accuracy, which is defined as the ratio of the results returned by a non-deterministic system to the results returned by a deterministic system.

The PAC architecture has several advantages. For example, it does not need to be repartitioned. Rather, should the rate of queries increase to a point where the latency becomes too high, the PAC architecture can gracefully degrade performance in order to maintain responsiveness. In addition, it is designed so that there is very little communication overhead within the system. It is also inherently scalable and fault tolerant. However, for the same computational and storage budgets, the expected performance of a PAC system is less than that for a deterministic system.

In [4] 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. 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 computational resources expended for an equivalent deterministic search, and is therefore not economically feasible. In this paper, we examine a number of ways in which PAC search can be modified to improve its accuracy, while utilizing computational resources comparable to a deterministic search.

In Section 3 we briefly review some basics of a PAC search system. In Section 4 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 accuracy quickly increases to close to 100%. Section 5 presents both simulation results, as well as results based on a TREC data set. Finally, we conclude and discuss future works in Section 6.

2 Related Work

Two broad classes of distributed computer architectures have been proposed for information retrieval. These are centralized distributed architectures, used by commercial search engines such as Google, and decentralized distributed architectures used by a variety of peer-to-peer systems.

In the centralized architecture, the index is distributed over a number of disjoint partitions [2]. 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 [9]. Changes to the collection or to the query distribution may necessitate that the index be repartitioned, a process than can be complex and time consuming. Note that while the index is distributed across machines, the machines themselves are typically housed within a central server facility.

The problem of repartitioning was addressed in, ROAR (Rendezvous On a Ring) [9]. However, the proposed system is still deterministic.

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.* [8] 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. 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.

Terpstra *et al.* proposed a non-deterministic architecture called Bubblestorm [16]. They viewed the problem from a different perspective than PAC, assuming queries and documents are randomly replicated to machines and estimating the chance of a query meeting a document on a same machine. Bubblestorm faces the same problem as PAC search: under fixed communication, computation and storage budgets, the low storage capacity of individual machines results in unacceptable retrieval performance.

A variety of peer-to-peer decentralized architectures [14, 15, 6, 10, 18, 17, 13] have also been proposed and deployed previously, with a variety of search capabilities. They all have deterministic indexing and retrieval processes, and can be classified as deterministic systems following the definition in [4]. A problem with these peer-to-peer decentralized architectures is that their search depends on the network structure. Thus, they are relatively fragile to network structure changes, i.e. nodes entering, nodes leaving and nodes failing.

In this paper, we focus our attention on queries that occur more than once, our goal being to improve our performance with each new instance of the query. There has been considerable work investigating the distribution of queries. Analysis of a search engine query log [12] shows that query occurrences roughly follow a power law distribution, where 63.7% of unique queries only occur once (forming the so-called "long tail"). An hourly analysis of a query log [3] highlighted the temporal behavior of query traffic during a day. The temporal and spatial features of queries have been taken into account in Web caching [5].

3 PAC search

Here we review the concept of PAC search and provide some results derived in [4] that will be needed subsequently. Figure 1 illustrates the basic elements of the PAC architecture. It is assumed that there are N unique documents in the collection to be indexed. For Web search, N is the unique number of web pages,

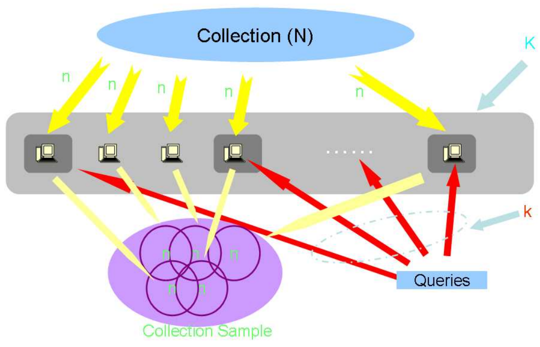


Fig. 1. Basic elements of the PAC architecture.

currently estimated to be of the order of 65 billion documents [1]. It is further assumed that K computers are available and each computer indexes a random sample of n documents. It is reported in [4] that Google utilizes 300,000 computers and we therefore set $K = 300,000$. The n documents on each computer are assumed to be unique, i.e. no document appears more than once. However, each computer's sample of n documents is *not* disjoint with other computers, so a document may, and very likely will, be indexed by more than one computer. In [9] it was reported that the fraction of the collection indexed by each machine in the Google search architecture, $\frac{n}{N}$ is 0.001, and we therefore use this ratio in our subsequent analysis.

The union of the K 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, $|C_s| = Kn$. Note that in a deterministic centralized distributed architecture the same storage capacity is need. However, in this case, each document is replicated on each machine within a partition.

It was shown in [4] that if each computer randomly samples the Web to acquire its n documents, then the expected coverage, E(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, N, is

$$E(\text{coverage}) = \hat{N}/N = 1 - \epsilon, \tag{1}$$

where ϵ is given by

$$\epsilon = (1 - n/N)^K \tag{2}$$

Since the value inside the parentheses is less than one and $K = 300,000$, then ϵ is effectively zero and our expected coverage E(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, k, of computers are queried, where $k = 1,000$ based on the number of disjoint partitions attributed to Google [9]. The union of n documents on each of the k computers is referred to as the sample index. Since k is much smaller than the total number of computers, K, 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

$$P(d) = E(\text{coverage}) = 1 - (1 - n/N)^k \tag{3}$$

is 0.63 when $k = 1,000$ and $\frac{n}{N} = 0.001$. 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.

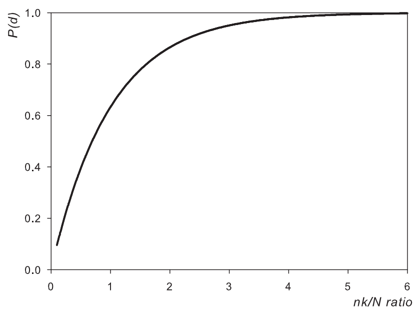


Fig. 2. The probability of retrieving any given relevant document of PAC search for different ratios of the sample index to collection size, $\frac{nk}{N}$.

If r represents the set of documents retrieved by a deterministic system, and r' represents the number of documents retrieved by PAC search that are contained in r, i.e. $|r'| \leq |r|$, then in [4] it was shown that the probability of retrieving $|r'|$ documents, $P(|r'|)$ is

$$P(|r'|) = \binom{r}{r'} P(d)^{|r'|}(1 - P(d))^{|r|-|r'|}, \tag{4}$$

For the specific case where $\frac{n}{N} = 0.001$, $k = 1000$, and r=10, the probability of retrieving 5, 6, 7, 8, 9, and 10 documents in r 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%.

The expectation of $|r'|$ is $E(|r'|) = |r|P(d)$. Therefore, Equation (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, kn in comparison with the collection size, N. Following the same setting as Google where $n/N = 0.001$, Figure 2 shows the accuracy of PAC search for various ratios of $\frac{nk}{N}$. 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%.

4 Centralized PAC search

The previous discussion of PAC search assumed a decentralized architecture in which a client device randomly selects k computers to issue a query to. As discussed in [4] 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 [4], it was proposed to ameliorate this problem, by issuing the query to a set of k 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 will always see the same result set, since the set of pseudo-random computers will always 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 computer or computers receives queries from users, performs the pseudo-random selection centrally, and then forwards the query to the chosen set of k machines. This is illustrated in Figure 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.

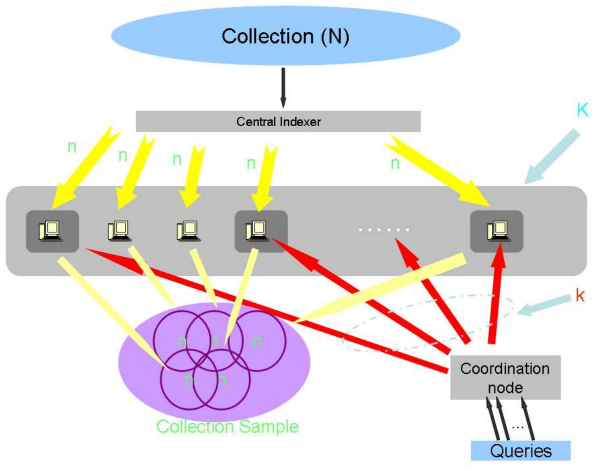


Fig. 3. A centralized PAC search architecture.

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.

4.1 Adaptive pseudo-random selection of the sample index

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 [3, 5]. For frequently issued queries, we consider whether it is possible to improve upon our initial selection of k 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 k pseudo-random computers. These computers form the sample index for this query. After receiving the results from the k machines, consolidating the results, and transmitting the retrieval results back to the user, we record the subset of computers that provided the highest ranked documents. The choice and size, k_0, of this subset will be discussed shortly. When we receive the same query a second time, the query is issued to this subset k_0 together with a new set, $k - k_0$ of pseudo-randomly chosen machines. Once again, after receiving the results from the k machines, consolidating the results, and transmitting the retrieval results back to the user, we record the subset, k_1, of computers that provided the highest ranked documents. And the process repeats. At each iteration, i, the identifiers of the retained computers, k_{i-1}, are cached. We refer to this as node caching. Note that nodes, rather than the documents retrieved from these nodes, are cached i.e. we are not caching queries in the traditional sense [5].

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 paper.

After responding to the initial query, we have examined k computers, and the expected accuracy is given by Equation (3). After responding to the query a second time, we have examined $k+k-k_0$ computers. From Equation (3) we know that if we had looked at all $2k-k_0$ computers simultaneously, then our expected accuracy would increase. This provides an upper limit on our performance. In practice, we did *not* examine all $2k - k_0$ computers simultaneously. Rather, at each iteration we examined k computers, this being the computation 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 (3)?

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

For a specific set of k computers, the expected number of unique documents in the sample index, $\hat{N}$, is, from Equation (1)

$$\hat{N} = (1-\epsilon)N = \left(1 - (1 - n/N)^k\right) N \tag{5}$$

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

$$P(|r|) = \prod_{\delta=0}^{r-1} \left(\frac{\hat{N} - \delta}{N - \delta} \right) \tag{6}$$

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

$$P(|\bar{r}|) = \left[1 - \prod_{\delta=0}^{r-1} \left(\frac{\hat{N} - \delta}{N - \delta} \right) \right] \tag{7}$$

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

$$P_{all}(|\bar{r}|) = \left[1 - \prod_{\delta=0}^{r-1} \left(\frac{\hat{N} - \delta}{N - \delta} \right) \right]^{\binom{K}{k}} \tag{8}$$

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

$$P = 1 - \left[1 - \prod_{\delta=0}^{r-1} \left(\frac{\hat{N} - \delta}{N - \delta} \right) \right]^{\binom{K}{k}} \tag{9}$$

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

Now, let us consider the number of relevant documents we expect to see on a single computer. For a given query, q, let r denote the number of relevant documents in the collection (of size N). remembering that each computer samples n documents, the probability that a sampled document will be relevant is simply $\frac{r}{N}$. And the probability, $P(r')$ of sampling $r' \leq r$ documents is given by

$$P(r') = \binom{n}{r'} \left(\frac{r}{N} \right)^{r'} \left(1 - \frac{r}{N} \right)^{n-r'} \tag{10}$$

This is a standard binomial distribution, so the expected number of relevant documents to be found on a single machine, $E(r')$ is

$$E(r') = nr/N \tag{11}$$

The preceding analysis indicates that (i) there are configurations of k computers on which all relevant documents will be present in the sample index, and (ii) for a known number of relevant documents, Equation (11) provides the number of relevant documents expected on each computer. This number can be used to guide an heuristic search to find a k-configuration that includes all relevant documents for a specific query. We begin with a description of an algorithm that assumes knowledge of the number of relevant documents, see Section 4.2. Of course, in practice this is not the case, and the algorithm is modified in Section 4.3 to account for this.

4.2 Known number of relevant documents

Before we consider the practical problem in which the number of relevant documents is unknown, we first consider the ideal case in which, for a given query, the number of relevant documents is known. The purpose of this exercise is to investigate the performance of an algorithm under idealized conditions. By so doing, we are better able to understand the upper limit on the performance of our algorithm prior to introducing heuristic assumptions. Experimental results are provided based on a simulation described in Section (5.1).

At iteration, i, each of our k computers has a retrieved r_j relevant documents, where $1 \leq j \leq k$. We order the computers based on the number of relevant documents retrieved, r_j. For simplicity, and without loss of generality, we assume that $r_1 \geq r_2 \cdots \geq r_k$.

At each iteration, we need to decide which computers to keep, and which to discard. Intuitively, we expect to retain more computers with each iteration, as we converge to a "optimum" configuration of k computers for a specific query. Therefore, we initially retain x% of computers (i.e. the x% that retrieve that largest number of relevant documents), and for each subsequent iteration, i, retain $(x + (i-1)y)$% of computers. We refer to this percentage as the "keep" ratio.

Furthermore, at each iteration, any computers in the keep ratio that retrieved less than the expected number of documents given by Equation (11) are discarded.

4.3 Unknown number of relevant documents

In practice, the number of relevant documents for a particular query is unknown. The assumption of the iterative method proposed in Section 4.2 is that the set of relevant documents is known beforehand. However, this is not realistic in a real retrieval environment. Instead, we assume that the top ranked r documents, based on a retrieval model such as the BM25 model [11], are relevant to the query. Indeed, in a deterministic system such as Google, the top ranked r documents are presented to users.

At iteration, i, our k computers produce a merged ranked list of documents, where each of our k computers has retrieved r_j documents, $1 \leq j \leq k$, in the top r rank positions. Intuitively, we favor a computer with many documents ranked highly in the top r positions.

As before, we can simply rank the k computers based on the number of documents each computer has in the top r positions, r_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 [7] for judging how well each computer is responding to a query. For computer, j, its score, s_j is defined as

$$s_j = \sum_{m=1,\ldots,r} \frac{\delta_m}{log_2(1+m)} \tag{12}$$

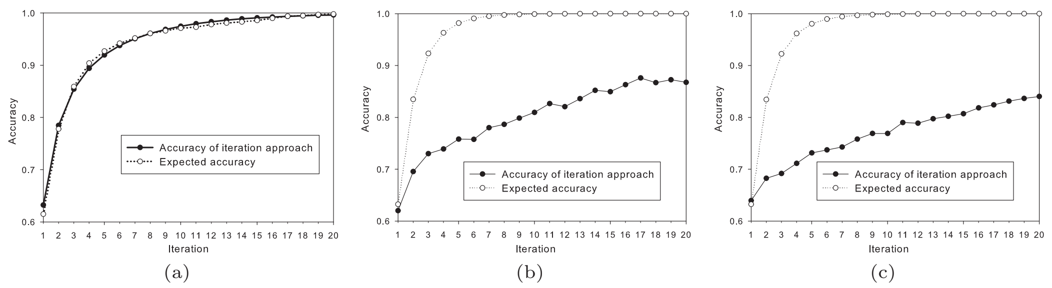


Fig. 4. Accuracy for our iteration approach and comparison with expected accuracy. The data points are based on the performance at each step. Under different number of relevant documents, r = (a) 1,000 (b) 2,000 (c) 4,000

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

We set r=1,000 in our TREC data based experiments in Section 5.2, 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 $x\%$ of the ordered list of computers in the first iteration. And in iteration, i, we retain the top $(x + (i - 1) * y)\%$ of the order list.

5 Experimental Results

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

For both simulations and TREC experiments, we used the same computation and storage requirements as assumed of Google. There are K=300,000 computers which independently sample N documents. We kept the ratio between the number of documents indexed by each computer, n, to N as 0.001, and fix the number of computers queried at each step $k = 1,000$.

We fixed the initial keep ratio at $x = 20\%$ in the first iteration. This is incremented by $y = 3\%$ in each subsequent iteration. A less heuristic pruning strategy is a topic of future work.

5.1 Simulation

Due to computational cost, we set N=100,000 and $n = 100$. Note that the expected accuracy is only affected by the *ratio*, $\frac{n}{N}$. Therefore, our results generalize to much larger N and n, provided the ratio remains fixed (0.001). We randomly sample r documents from the entire N and treat these as the relevant set to a query. The measured accuracy is the ratio of relevant documents, r', found on k computers, to the total number of relevant documents r.

We consider the cases where the number of relevant documents are r=1,000, 2,000, and 4,000. The expected number of relevant documents on a computer is $\frac{nr}{N}$ = 1, 2, and 4, respectively. For r=1,000, 2,000, and 4,000, we ran ten trials of our simulations, and average the accuracy scores at each iteration step over the ten trials. The results are shown in Fig. (4). Note that the variances of the accuracy scores at each step are small, and do not affect the overall trend. Fig. (4) shows that when the number of relevant documents $r = 1,000$, the iterative algorithm closely follows the upper bound on performance, and an accuracy of 99% is reached in 15 iterations. As the number of relevant documents increases ($r = 2,000$ and $r = 4,000$), the iterative algorithm performs less well. The reason for this degradation in performance as r increases is unclear, and a topic of ongoing research.

Next, we study how the number of sampled computers, k, affects the performance of the algorithm when r is large. We fix the number of relevant documents to $r = 4,000$, and considered two values of k, namely 1,000 and 2,000. The results are shown in Fig. (5), where we observe that for the larger k the simulation

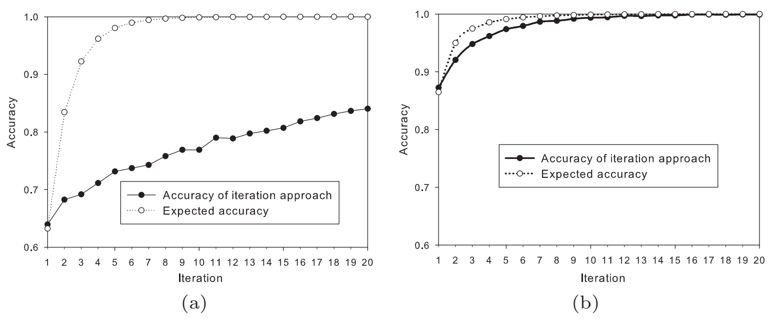


Fig. 5. When the number of relevant document r is 4,000, when we adjust the number of sampled computers k, accuracy for our iteration approach and comparison with expected accuracy. The data points are based on the performance at each step. k= (a) 1000 (b) 2000

results much more closely track the upper bound on performance. By querying 2000 rather than 1000 machines, our initial accuracy increase from 63% to 86% and this appears to allow us to perform a much better pruning at subsequent iterations. Conversely, it appears that when we query $k = 1000$ machines and our initial accuracy is 63%, our hueristic pruning strategy is retaining poor machines. The reason for this is unknown and the subject of ongoing work.

5.2 TREC Experimental Results

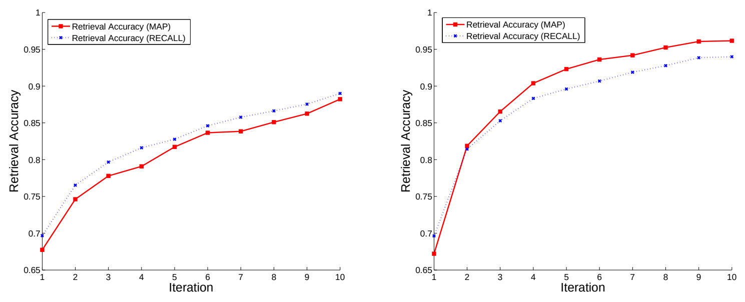
In Section 5.1, we investigated PAC performance in a simulated retrieval environment in which we assumed that all the relevant documents for a query are known. However, in a real retrieval environment, the number of relevant documents is unlikely to be known. Thus we need to evaluate our approach based on document retrieval models. Our proposed approach in Section 4.3 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., K=300,000, and $\frac{n}{N}$=0.001. We used the BM25 retrieval model [11] 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 4.3. 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 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:

$$RetrievalAccuracy(Metric) = \frac{Metric_{PAC}}{Metric_{det}}, \tag{13}$$

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 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.



(a) (b)

Fig. 6. Retrieval accuracy of our iterative approach on MAP and Recall-1000. Results are based on the TREC8 50 queries. (a) when nodes are pruned based on the number of documents they retrieve in the ranked list, and (b) when nodes are pruned based on the NDCG-like metric.

Figure 6 (b), also shows that both MAP and Recall-1000 follow a similar trend, i.e., performance 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.

6 Conclusions 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 paper, 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 [4] 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 k computers in order to improve the performance.

A theoretical analysis provided a proof that there exists (with near certainty) a configuration of k 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 $r = 1000$. 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 $r = 4000$, the case where twice as many machines ($k = 2000$) 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 $k = 1000$, the iterative algorithm is retaining a sub-optimal choice of machines. Analysis to the relationship among the number of relevant documents, the number of queried machines and the retrieval performance 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 10 times). For less frequently occurring queries, alternative approaches must be developed, which are the subject of future work.

References

1. http://www.worldwidewebsize.com, 2009.
2. L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.
3. S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. A. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR*, pages 321–328, 2004.
4. I. Cox, R. Fu, and Larsen. Probably approximately correct search. In *Proc. of the Internationla Conference on Theoretical Information Retrieval (ICTIR)*, 2009.
5. 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 Trans. Inf. Syst.*, 24(1):51–78, 2006.
6. 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.
7. K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
8. 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.
9. C. Raiciu, F. Huici, M. Handley, and D. S. Rosenblum. Roar: increasing the flexibility and performance of distributed search. *SIGCOMM Comput. Commun. Rev.*, 39(4):291–302, 2009.
10. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of the International Middleware Conference*, 2003.
11. S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Proc. of the Third Text REtrieval Conference (TREC 1994)*, pages 109–126, 1996.
12. C. Silverstein, M. R. Henzinger, H. Marais, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.
13. 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.
14. 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.
15. C. Tang, Z. Xu, and M. Mahalingam. psearch: Information retrieval in structured overlays. In *HotNets-I*, 2002.
16. W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM*, pages 49–60, 2007.
17. K.-H. Yang and J.-M. Ho. Proof: A dht-based peer-to-peer search engine. In *Conference on Web Intelligence*, pages 702–708, 2006.

18. Y. Yang, R. Dunlap, M. Rexroad, and B. F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. In *INFOCOM*, 2006.