A Distributed Search System Based on Markov Decision Processes

Yipeng Shen, Dik Lun Lee, and Lian Wen Zhang

Department of Computer Science
The Hong Kong University of Science and Technology
{yipeng, dlee, lzhang}@cs.ust.hk

Abstract. This paper discusses a flat distributed search system in which selecting the relevant servers to a given query becomes difficult due to the lack of global knowledge at each server. To ensure the quality of the query results, Markov Decision Processes (MDPs) are set up for the query terms among all the autonomous search servers in the system. A new server ranking method is also designed for this MDP based system. The simulation reveals that query results with satisfactory quality can be achieved since the global optimal servers are obtained by the precomputed MDP policies.

1 Introduction

The ever increasing amount of information available on the World Wide Web (WWW) has made searching and indexing information a difficult task on the Internet. Various search engines such as AltaVista, InfoSeek, Yahoo, etc., have been developed and used nowadays.

Normally, a query submitted to the Internet search engine is composed of terms or keywords. The relevance of a document to a given query is calculated according to some formulae such as **TFxIDF** [4]. In comparison to the queries used in traditional information retrieval systems, the queries posed to the search engines on the Internet consist of only about 2 query terms on average [5].

The Web is believed to contain several hundred million pages, but the coverage of a single search engine is only a small part of the Web. Distributed search systems containing a large number of search engines can address this problem. The search engines in a distributed search system are connected as a graph, each search engine is autonomous and maintains its own index and document collection. Queries are answered via the cooperation among search engines so that the most relevant documents in the system are retrieved.

The problem of distributed searching has been investigated by many people. $Whois++$ [10] and *Discover* [8] are two typical systems with hierarchical architectures containing multiple layers. A probabilistic model for the hierarchical architecture is described in [1]. However, the autonomy of each server is weakened in the hierarchical architecture since each search engine is managed by its parent in the upper layer. The hierarchical architecture is not likely to scale up for the Internet.

An inference network model is proposed in [2]. In this model, a special global server is used to store the inference network for selecting the relevant servers when a query is posed to the distributed system. Similarly, a server containing the global knowledge which is able to select the most relevant servers from the system is assumed in [3], [6], [11] despite the different server ranking algorithms used by them. Moreover, each server in [9] should store part of the information of every other server in the system in order to pick up the relevant servers. Of course, this is not feasible on the WWW.

The motivation of this paper is to set up a system with a fully distributed architecture. The servers form a flat server architecture (network) in which every server only knows the knowledge about its neighbours. Given a query at a server in the flat distributed system, it is difficult to identify the relevant servers since the server receiving this query only has local knowledge. In order to address this problem, Markov Decision Processes (MDPs) are constructed among the server network so that the relevant servers can be identified efficiently.

Section 2 describes the distributed search system based on MDP. Section 3 discusses the server ranking method adopted by the MDP model. Section 4 presents the experimental results. In Section 5, we draw our conclusion.

2 Distributed Search System Based on MDP Model

In this section, we show how search in a distributed system can be modelled as an MDP. The main issues in our MDP model are discussed.

2.1 MDP and Distributed Searching

The MDP deals with the sequential decision process in a dynamic system. At any decision time, the decision maker chooses a decision or action from a number of alternatives based on the current state of the system. After the chosen action is executed, the decision maker receives an immediate reward returned by this action and the system may transfer to another state. The objective is to obtain the greatest reward after a series of actions. Formally, an MDP consists of four components: a set of all possible states, each state's set of possible actions, transition probability and immediate rewards obtained by the decision maker.

Given a set of servers $S = \{s_1, s_2, ..., s_N\}$, let these servers form a flat system architecture (network). If we pose a query q at server s, $s \in S$, the task is to locate the most relevant servers in S. We start at server s and determine whether to answer the query at this server or to pass the query to a neighbour in the server network. If the latter is chosen, the decision will be further processed at the neighbour who receives the query. Thus, the server where query q currently resides can be regarded as the present state of the query and S is the set of all possible states. The present state of the query changes when the query is passed to a neighbour. This is a sequential decision process and it can be naturally modelled as an MDP.

Let $A_s = \{a_1, a_2, ...\}$ be the set of all possible actions at server s. According to these actions, query q may be answered at s or passed to a neighbour of s. If the action a at s is to return itself, we obtain an *immediate reward* $r_q(s, a = s)$, which represents the relevance or the goodness score of server s with respect to q. On the other hand, if the action a is to pass the query to one of its neighbour s', the immediate reward $r_q(s, a = s')$ is 0 and the action takes us to server s'.

The optimal value function $V_q(s)$ is the expected total reward we can obtain by acting optimally from server s. Bellman's optimality principle tells us that

$$V_q(s) = max_{a \in A_s}[r_q(s, a) + \lambda V_q(T(s, a))] \tag{1}$$

$T(s, a)$ stands for the next state (server) if action a is chosen. The transition $T(s, a)$ caused by action a is deterministic in our model. That is, the state of the query execution switches to another server with probability 1 if action a is chosen. The first term in the equation is the local reward received right away, which is zero when the action a is to proceed to another server. The second term is the reward that can be received in the future, which is zero if the action is to return the current server.

Parameter λ is a discount factor which is between 0 and 1. It captures the communication cost between adjacent servers. Due to parameter λ, the local immediate reward $r_q(s, s)$ of server s is discounted step by step in the server network when it is propagated to the other servers. The discounted reward, which is propagated to a remote server s', might be much smaller than its original value. Despite the greater initial local immediate reward of server s, the nearby servers of the remote server s' might propagate greater rewards to s' since their rewards are less discounted. Thus, the influence of server s on server s' is reduced by the parameter λ. The less the λ, the greater the reduction. We can confine the influence of a single server to an appropriate range by the parameter λ. The communication and computation cost is then reduced without losing much result precision.

The optimal policy (i.e., the optimal action $d_q(s)$ with respect to q at every server s) can be obtained as follows.

$$d_q(s) = arg\ max_{a \in A_s}[r_q(s, a) + \lambda V_q(T(s, a))] \tag{2}$$

This optimal policy may be computed by means of *value iteration* [7]. Given a query q, value iteration starts with 0-value functions $V_{q,0}(s)$ at iteration step 0 and iteratively computes a series of value functions $V_{q,i}(s)$ $(i = 1, 2, ...)$. In our MDP model, the value iteration can be simplified according to its specific Bellman equation where the transition is deterministic. That is, we iteratively compute $V_{q,i}(s)$ $(i = 1, 2, ...)$ with respect to the following formula at every server s until the sequence of value functions $V_{q,i}(s)$ $(i = 1, 2, ...)$ converges to the optimal value function $V_q(s)$.

$$V_{q,i}(s) = max_{a \in A_s}[r_q(s, a) + \lambda V_{q,i-1}(T(s, a))] \tag{3}$$

2.2 Multiple Access Policy

Suppose the communication between servers is strictly confined to the server network. For a query q posted at s_{i1}, according to the optimal action a_{i1} obtained by the value iteration, q may be passed to another server s_{i2} and the delivery of q changes the current state to s_{i2} and so on. Once the query reaches a server s_{im} where the action is to stick to s_{im}, the delivery of query q stops and the query is answered at s_{im} (i.e., s_{im} is the optimal server). Later on, s_{im} is also called the *origin* of the optimal reward $V_q(s_{i1})$ at server s_{i1}.

The above discussion only copes with single cast, namely, selecting a single server from the system. For multicasting where a query may have several relevant servers, the value iteration algorithm must be revised.

The value iteration can be extended by not only keeping the greatest expected reward $V_q(s)$ and the corresponding optimal action but also memorizing the top k best expected rewards, the corresponding optimal actions and their corresponding origins. Every origin should be distinct since k different optimal servers are to be selected. In other words, a top k list is reserved in the value iteration at every server for query q. The *multiple access policy*, which provides k optimal servers with respect to a query posed at any server in the system, consists of the top k lists at all servers.

2.3 Queries with Multiple Terms

Obviously, it is not possible to set up the MDP value iteration for every possible query because of the high computation cost. However, the number of distinct single query terms is limited, especially when we focus on the frequently used terms. Thus, it is feasible to set up MDPs for all single query terms. A query with multiple terms can then be answered according to the MDP policies of the query terms contained in the query.

Suppose query $q =< t_1, t_2 >$, which has two query terms, is submitted to server s. $L_{t1}(s)$, $L_{t2}(s)$ are the two lists of top k optimal policies at server s with respect to query term t_1 and t_2 respectively. The total relevance of server s' to query q is the sum of the relevance scores of all query terms in query q.

If k is appropriately large, query q can be answered in the following manner. List $L_{t1}(s)$ and $L_{t2}(s)$ are merged into a new list $L_q(s)$ sorted by the adjusted server scores. That is, if s' appears in both $L_{t1}(s)$ and $L_{t2}(s)$, the adjusted relevance score of s' in $L_q(s)$ is the sum of its two expected rewards in $L_{t1}(s)$ and $L_{t2}(s)$. Finally, the new top k servers are picked up. Similarly, queries with more than two query terms can be processed in the same manner.

From now on, the MDP with top k access policy is constructed according to a single term t instead of a query q.

2.4 MDP Server

Value iteration assumes knowledge about the topology of the entire server network. This assumption is not true in a flat distributed system where every server is fully autonomous and only has the knowledge about its neighbours. Fortunately, there is a variation of value iteration that does not require this assumption. The variant is called *asynchronous value iteration* [7], which is adopted in our simulation experiments.

In terms of asynchronous value iteration, every server carries out its own value iteration steps whenever it is free. Not concerning about the value iteration steps of its neighbours, each server only renews its expected total reward based on the current expected rewards of its neighbours during every value iteration step. Compared to the synchronous value iteration, each server may carry out different number of iteration steps in the asynchronous value iteration.

Since every server in our system has to fulfill both the query processing task and the asynchronous MDP value iteration task, a search engine (server) is extended with some components related to the MDP policy computation and the query processing based on the calculated MDP policies. These servers are referred to as MDP servers.

3 Server Weight

In this section, we concentrate on how the relevance of servers with respect to the given query is evaluated. In the MDP, the server weight standing for the relevance of a server to a query is intended as the immediate reward obtained at that server (i.e., $r_q(s, s)$ in Formula 1).

All traditional server ranking methods depend on some global information about the other servers [12]. In a flat distributed environment, the goodness score of a server should be evaluated only upon that server's local information according to the MDP model set up so far. That is, all the servers ought to be ranked by a uniform ranking criterion with respect to their own information. The comparison or interaction between servers is accomplished during the value iteration of the MDP. The advent of the following server ranking method satisfies this requirement in the MDP model and is referred to as the $Uniform\ Ranking\ Criterion$ (URC).

The idea of URC comes from the term ranking method used in the query refinement of $Discover$ [8] where the moderately correlated terms are given the highest ranking values. In the following formula, suppose $\bar{p}$ is a moderate value between 0 and 1, the greatest value of $\mathcal{M}_{\bar{p}}$ is obtained if $p_i = \bar{p}$.

$$\mathcal{M}_{\bar{p}} = 1 - |p_i - \bar{p}|$$

Now let's derive the URC server ranking method from the previous term ranking method used in $Discover$. Given a term t, despite the variation of server ranking methods, the relevance is basically calculated based on the *term frequency* (TF) and the *document frequency* (DF) of term t. In particular, TF is the crucial parameter for determining relevance. Typically, TF is normalized to a value between 0.5 and 1, which is referred to as the *normalized term frequency* (tf). Given a document i in a collection and term t, $tf_{i,t}$ is:

$$tf_{i,t} = 0.5 + 0.5 \times TF_{i,t} / \max_{k \in i} (TF_{i,k})$$

Given term t, server s_i and its own local document collection, $stf_{i,t}$, which is the sum of the tf of all the documents containing t in s_i, is defined as follows.

Definition 1 *Total normalized term frequency for term t at server s_i is:*

$$stf_{i,t} = \sum_{doc_k \in s_i} tf_{k,t}.$$

Furthermore, $stf_{i,t}$ is obtained from $DF_{i,t}$ documents ($DF_{i,t} > 0$ if $stf_{i,t} > 0$) at server s_i. Since $tf_{k,t}$ is between 0.5 and 1 for all $DF_{i,t}$ documents, $stf_{i,t}$ is a number between $0.5 \times DF_{i,t}$ and $DF_{i,t}$. Let $atf_{i,t}$ be the *average* tf at s_i.

Definition 2 *Average normalized term frequency for term t at server s_i is:*

$$atf_{i,t} = stf_{i,t}/DF_{i,t}$$

$atf_{i,t}$ is also in $[0.5, 1]$. It stands for the average relevance of all the documents containing t. Suppose we have $stf_{i,t} = stf_{j,t}$, $i \neq j$, their corresponding atf are $atf_{i,t}$ and $atf_{j,t}$ respectively. Normally the greater atf results in better retrieval precision, while on the contrary, the recall is ensured if the server with lower atf is picked up.

According to the above discussion, the relevance of a server is generally determined by its stf value. But the user might opt for different servers depending on their different recall/precision requirements. The URC server weight is then elicited.

$$\begin{aligned} &G_{i,t} = stf_{i,t} \cdot [\alpha + \beta \cdot (1 - |atf_{i,t} - \bar{p}|)] \\ &\alpha + \beta = 1, \quad 0 < \alpha < 1. \end{aligned} \tag{4}$$

α in Formula 4 is called the importance weight of stf. The greater the α, the more important the stf in calculating the server weight. The latter term $\beta \cdot (1 - |atf_{i,t} - \bar{p}|)$ is adjustable so that the user can decide which atf is the most preferable (i.e., $\bar{p}$ in Formula 4). Besides, according to Formula 4, the importance of atf is further scaled up by stf so that it is proportional to the total term frequency at the server.

If the user needs to choose documents with higher relevance scores, a larger value is assigned to $\bar{p}$, whereas $\bar{p}$ is smaller if the user tends to retrieve more relevant documents regardless of their relevance scores. Nonetheless, α is generally larger than β since stf is the dominant factor in evaluating the relevance of a server to a term. The value of server weight $G_{i,t}$ is in $[0, stf]$.

The discrimination between different α parameters is demonstrated in Fig. 1. With a greater α parameter($\alpha = 0.8$), the stf takes an important role in the server weight. As long as the server has large stf, high server weight is always obtained. Whereas, a smaller α parameter causes the server weight to decrease sharply if the atf deviates from the peak point $\bar{p}$.

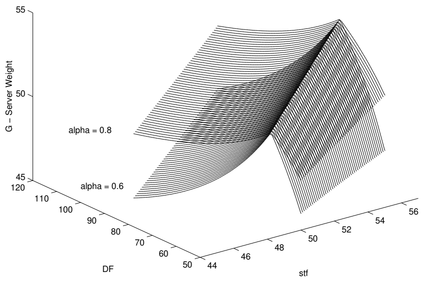

Fig. 1. URC Server Weight ($\bar{p} = 0.8$)

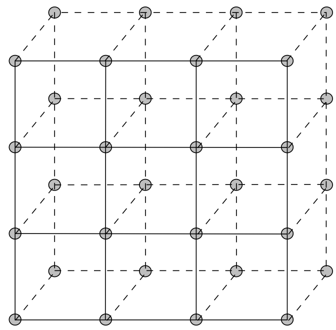

Fig. 2. Folding Mesh

4 Experimental Results and Analyses

To verify the performance of adopting the MDP model in the distributed search system, a simulator is constructed and experiments have been conducted based on some standard information retrieval document collections.

4.1 Environment Construction

Our simulator is running upon a pseudo distributed search environment. To set up this environment, the document collections, the server connections and the test queries should be constructed in advance.

Our experiments include 32 pseudo servers which form a regular flat architecture (Fig. 2). It is referred to as the *Folding Mesh*. The average node degree in the network is 4, the minimum and maximum node degrees are 3 and 5 respectively.

Four standard document collections used in performance analyses of many information retrieval systems are selected as the test data in our experiments (i.e., CACM, CISI, CRAN and MED). These document collections are available from the Smart System at Cornell University (*ftp://ftp.cs.cornell.edu/pub/smart/*). These four original sample collections are divided and distributed to 32 servers.

Although the four sample document collections provide their collateral sample queries, these queries are descriptions or excerpts rather than queries with few terms. In accordance with the queries posed on the WWW, we extract the short queries with few terms from these sample queries. Then these extracted queries are imposed on the distributed search system and the returned query results are evaluated.

4.2 Performance Criterion

In terms of the quality of query results, a criterion called the *Effectiveness Ratio* is adopted. Suppose we combine all the local document collections together and treat the combination as one huge global document collection, then the classic text retrieval method **TFxIDF** can be applied to this combined collection. Given a query q, the performance is measured by the *Effectiveness Ratio*.

Definition 3 *The Effectiveness Ratio is the ratio between* v_d *and* v_h *(i.e.,* v_d/v_h*), where* v_h *is the sum of the* **TFxIDF** *scores in the global document collection of all the documents retrieved directly from the global document collection;* v_d *is the sum of the* **TFxIDF** *scores in the global document collection of all the documents obtained by the selected MDP servers.*

4.3 Results

Cast Number Given a query, more than one of the 32 servers may be relevant. The more servers the query accesses, the higher result quality we achieve; on the other hand, visiting more servers increases the system's overload. Ideally we hope that a higher quality can be obtained by covering only a small portion of the distributed system.

In the following experiments, the server ranking method is the URC defined in Section 3, $\alpha = 0.7$ and $\bar{p} = 0.9$. The ranking method with these α, $\bar{p}$ values tends to select servers with great stf and atf.

The experimental result is depicted in Fig. 3. Observing the gradient of the curves, we know that the Average Effectiveness Ratio increases at a high rate at the beginning. As the number of visited servers becomes larger, the curves tend to be flat. This means the servers containing most of the relevant documents with high relevance are selected at first, the servers with less relevance are unlikely to be visited initially. Besides, the smallest Average Effectiveness Ratio is near 0.6 when eight servers among thirty two are accessed.

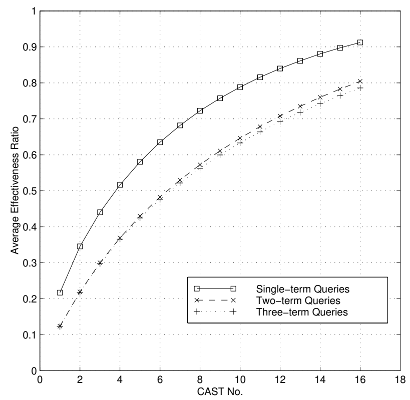

Fig. 3. Average Effectiveness Ratio and Cast Number

Reward Discount In the MDP model, the λ parameter is the discount factor of the reward that can be obtained in the future. It represents the communication delay between MDP servers.

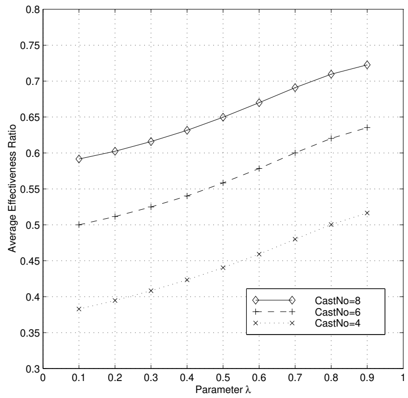

Fig. 4. Average Effectiveness Ratio and Parameter λ (Single Term Queries)

Fig. 4 is the experimental result of single term queries. Generally, the quality of the query result is proportional to the λ parameter. As λ varies from zero to one, the quality of the query result increases slowly when λ is near the two ends of the $[0, 1]$ interval. Namely, if λ is an extreme value, the variation in the result quality is small. While λ has a moderate value in $[0, 1]$, the result quality almost increases linearly. Meanwhile, it seems that visiting six servers can cover most of the relevant contents in our case.

5 Conclusion

The simulation results present the general prospect of the MDP model applied to the distributed information retrieval system on the WWW. Although each server in a flat architecture only has local knowledge at the beginning, the global optimal servers are obtained after the MDP policies are computed. As a result, adopting MDP model helps the distributed search system to improve the quality of the query result since the precomputed MDP policies can efficiently locate the most relevant servers.

Because every server in our model takes a same role, flat distributed systems are more flexible and extensible than hierarchical systems. This conforms to the need of the dynamic WWW environment.

Since the sample document collections adopted in our experiments are not large enough, our future task is to adopt larger test collections in the experiments and further refine our MDP model.

References

1. Baumgarten, C. A probabilistic model for distributed information retrieval. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Philadelphia PA, USA, 1997), pp. 258–266.
2. Callan, J., Lu, Z., and Croft, W. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1995), pp. 21–28.
3. French, J. C., Powell, A. L., Viles, C. L., Emmitt, T., and Prey, K. J. Evaluating database selection techniques: A testbed and experiment. In *Proceedings of the 21th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Melbourne, Australia, 1998), pp. 121–129.
4. Harman, D. Ranking algorithms. In *Information Retrieval: Data Structures and Algorithms* (Englewood Cliffs, NJ, 1992), Prentice-Hall, pp. 293–362.
5. Jansen, B. J., Spink, A., Bateman, J., and Saracevic, T. Real life information retrieval: A study of user queries on the Web. *SIGIR Forum 32*, 1 (1998), 5–17.
6. Meng, W., Liu, K.-L., Yu, C., Wang, X., Chang, Y., and Rishe, N. Determing text databases to search in the internet. In *Proceedings of the 24th VLDB Conference* (New York, USA, 1998), pp. 14–25.
7. Puterman, M. L. *Markov decision processes : discrete stochastic dynamic programming.* John Wiley & Sons, New York, 1994.
8. Sheldon, M. A., Duda, A., and Gifford, D. K. Discover: A resource discovery system based on content routing. *Computer Networks and ISDN Systems 27*, 6 (1995), 953–972.
9. Viles, C., and French, J. Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1995), pp. 12–20.
10. Weider, C., and Spero, S. Architecture for the WHOIS++ index service. Internet Working Draft, 1993.
11. Xu, J., and Callan, J. Effective retrieval with distributed collections. In *Proceedings of the 21th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Melbourne, Australia, 1998), pp. 112–120.
12. Yuwono, B., and Lee, D. L. Server ranking for distributed text retrieval systems on the Internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications* (Melbourne, Australia, April, 1997).