Distributed Search[*]

Benjamin W. Wah
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
E-mail: b-wah@uiuc.edu
URL: http://manip.crhc.uiuc.edu

Searching in a distributed system involves a thorough examination of the system in order to find something of interest. It involves a partitioning of the state space and its associated operator and control regime so that local searches can be performed in different parts of the distributed system [1]. A local search in this case entails the execution of a conventional search algorithm in a sequential processor.

There are numerous applications involving distributed search. In the simplest case, finding a route from a source computer to a destination computer in the Internet involves sending queries to various domain name servers in order to translate the destination machine name into an IP address, and looking up the local routing table to determine the next computer to forward the message to. Such a search can be done efficiently because up-to-date information on network connectivity and routing is periodically distributed in the system. As another example, a user wishing to look up the topic "distributed search" may call a search engine to look up information in various digital libraries and the World Wide Web. This information can be found in a short amount of time when the collection of information in the system has been indexed previously.

An application involving a distributed search may have one or more well-defined objectives, as in finding the shortest path to a destination, or have a set of ill-defined objectives, such as finding all the articles on the World Wide Web related to "distributed search." When the objectives are ill-defined or undefined, the search may start with some initial objectives, and users may be asked to evaluate the quality of the results returned in order to refine the objectives and the scope of the search. The results returned by a search may not always be the best because the information sought in a distributed system may be time varying and of large quantity, and there are finite delays in propagating information from one point to another. This happens in finding the shortest path to a destination in which the shortest path may be based on possibly

[*]Research supported by National Science Foundation Grant MIP 96-32316 and National Aeronautics and Space Administration Grant NAG 1-613

out-dated topology and congestion information in the network. The searcher must, therefore, operate based on the local state of the distributed system and information that can be obtained, and be able to extrapolate the information in time when the information obtained is out-dated.

There are four components of a distributed search:

Search algorithms. Extended from sequential search algorithms, these include depth-first, breadth-first, best-first, and heuristic searches. In each case, the algorithm maintains a queue of nodes to be searched, fetches a node from the queue, performs the search, adds any new nodes found to the queue, and sorts the queue of nodes according to some measure.

The performance of a search algorithm in a distributed system depends on the order that nodes are traversed. A depth-first search that focuses the search on one node and traverses all subtrees rooted at this node before traversing others may overload the node. In contrast, a breadth-first search that traverses all nodes in the queue before going deeper into the subtrees may result in a large queue of pending nodes. To avoid space from growing unbounded, a heuristic function can be used to determine the next best node to search. This function can be designed to optimize a combination of delays, amount of information obtained, workload, number of nodes traversed, and queue size.

Search algorithms are employed in Internet search services, such as Lycos [2], WebCrawler [3] that access Web pages and compile index information to be used later. These algorithms are generally based on heuristic searches that employs some popularity heuristics which measure, for example, the number of external Web servers with at least one link to a Web page. Such heuristics are used to indicate the usefulness of a Web page and, therefore, the demand from others to access the information.

Search algorithms can also be combined to form meta-search algorithms. The idea is that different search algorithms may return different information. Hence, a meta-search algorithm that calls multiple search algorithms in parallel and integrate the responses received by ranking it may provide better results. For example, MetaCrawler [4] is an Internet meta-search service that calls multiple search services and aggregates the information before returning it to users. The key services provided in meta-search is, therefore, the assimilation of information returned rather than the search itself.

The order that a search is performed may also be driven by factors other than the information desired. These factors may include network load, network connectivity, firewalls in networks, protocol compatibility, and language compatibility. The search algorithm may need to utilize dynamic information collected at run time in order to determine the next node to search.

Search agents. These are specialized fine-grained software modules that logically traverse a distributed system and looks for desired information [1]. Physically, they can be implemented as clients that communicate with remote servers using a standard networking protocol to download the necessary information. The information received is then parsed, filtered, and indexed before being passed to a search engine.

On the World Wide Web, the search agents are called robots that communicate with WWW servers using the Hypertext Transfer Protocol (HTTP). A robot will not access a site if it or a portion of it is excluded from public access. It will also avoid retrying repeatedly on failed accesses. It must be intelligent enough to detect loops in circular symbolic links leading to loops in accesses of the same set of sites.

To allow agents to cope with new conditions not anticipated at design time, they may have meta-knowledge that allows them to be reusable and adapts to unforeseen conditions [1]. Such meta-knowledge allows the agents to share metalevel search information at run time, allowing them to run more efficiently, adaptable, and maintainable. In this mode of distributed problem solving, agents preserve their own local view of the world, bounded by their local knowledge. Since agents with only local information may generate conflicting solutions, the solutions found must be integrated into a coherent form before delivered to users.

An alternative to using search agents is to have servers periodically push the necessary information to the searchers so that it can stored for later lookup. This mode of operation is useful in maintaining time-varying control information in a distributed system. For instance, the best route to a destination is obtained by searching a routing table that contains information pushed from other sites in the network, and does not involve traversing the network at run time to find the most efficient route.

Indexing schemes. These involve abstracting information found and storing it in a database to facilitate future lookups. This is necessary when the overhead of sending queries in a distributed system in real-time is large, and users are satisfied by retrieving information stored in a database in a timely fashion rather than actually performing the search in real time.

When the amount of information involved is small and precise, it can be kept efficiently in local memory. For instance, network connectivity can be maintained in a routing table in local memory and accessed when queried.

On the other hand, when the information involved is large and possibly imprecise, such as the information returned by a search robot in the World Wide Web, then the searcher has to decide what information to keep, how to keep the information to facilitate efficient insertion of new information and random read accesses to any particular record, and how to manage the information efficiently in terms of disk space. For instance, some Web searchers keep the whole text of pages, while others rely on author-generated descriptions of documents. These schemes may result in large disk usage and possibly copyright violations. Lycos [2] uses automated abstracts that combine the 100 keywords most related to the document being indexed with the titles, header text, and an excerpt of the first 20 lines, or 10% of the document, to result in an abstract about one quarter the size of the original document. The keywords and abstract form an index to be used when queried. In contrast, Yahoo predefines a tree-structured hierarchy of descriptions as an index structure, and documents found are matched against the hierarchy in order to find a node to be associated with.

Querying processing with interactive feedback. Query processing is important when the information searched may not be precise and may be contained in a large number of documents, as in the case of the World Wide Web or a digital library. Queries supplied by users may be incomplete, or imprecise, or too general, leading to a huge amount of information found by the searcher. In this case, the queries will have to be refined interactively until the required information is identified.

For a given query, there are four general schemes to match a query with the information stored or searched: Boolean keyword query, regular expression matching, vector-space retrieval, and inverted file indexing. In Boolean keyword query, the query is formulated as Boolean expressions of keywords, and the searcher finds documents that satisfy the expression. In regular expression matching, the entire database is searched to find documents that match the regular expression. Besides being very expensive, both of these schemes retrieve information in database order, not in relevance order. In vector-space retrieval [5], the similarity between a document and a query is measured by the cosine of the angle between their vector representations in a multi-dimensional space. One way to compute this similarity value is to compute the sum of the weights of the query terms that appear in a document, normalized by the Euclidean vector length of the document. Here, the weight of a term depends on the term's occurrence frequency in the document and the number of documents containing the term in the document. Finally, inverted file indexing is used in some commercial Internet search engines, including WebCrawler [3], Lycos [2], and RBSE Spider. In this approach, the searcher keeps a list of all occurrences of keywords in an inverted file, each of which points to a list of documents and a list of positions in the documents. The inverted file allows a searcher to determine the proximity of the query terms to the terms in a document, the relative positions of the query terms, the frequency of occurrence of the query terms, and the number of query terms. This scheme has been found to be relatively inexpensive to compute for a large database.

References

[1] S. E. Lander and V. R. Lesser, *Sharing Metainformation to Guide Cooperative Search among Heterogeneous Reusable Agents*, IEEE Transactions on Knowledge and Data Engineering, vol. 9, no. 2, March-April 1997, pp. 193-208.

[2] M. L. Mauldin, *Lycos: Design Choices in an Internet Search Service*, IEEE Expert, vol. 12, no. 1, January-February 1997, pp. 1-8.

[3] B. Pinkerton, *Finding What People Want: Experiences with the WebCrawler*, Proc. Second Int'l World Wide Web Conference, Elsevier Science, 1994.

[4] E. Selberg and O. Etzioni, *The MetaCrawler Architecture for Resource Aggregation on the Web*, IEEE Expert, vol. 12, no. 1, January-February 1997, pp. 8-14.

[5] G. Salton and C. Buckley, *Term Weighting Approaches in Automatic Text Retrieval*, Information Processing and Management, vol. 24, no. 5, pp. 513-523, 1988.