# A distributed approach to high-performance information retrieval

Kenneth Baclawski* and J. Elliott Smith
Northeastern University
College of Computer Science
Boston, Massachusetts 02115
(617) 373-4631
FAX: (617) 373-5121
{kenb,esmith}@ccs.neu.edu

April 3, 2006

**Abstract**

A distributed architecture and indexing algorithm for high-performance information retrieval has been developed. A prototype system has been built that achieves a throughput of 500 queries per second with a response time of less than one second on an 8-node network of workstations. The algorithm is robust in the presence of lost and duplicated messages as well as failures of nodes of the network. The architecture can be scaled up to larger networks and higher levels of service. The retrieval model allows for semantically rich queries and information objects.

# 1 Introduction and Architecture

In this article, we discuss a distributed approach to high-performance information retrieval, called `KEYNET`. More details about the model and its architecture have been presented elsewhere [BS94b, BS94c, BS94d]. In this note, we sketch some background material about `KEYNET`, and then discuss those aspects of the distributed algorithm that have not been presented elsewhere.

The `KEYNET` system is designed for information retrieval (IR) from a corpus of information objects in a single subject area. It is especially well suited for non-textual information objects, for example, scientific data files, satellite images and videotapes, although some kinds

*This material is based upon work supported by the National Science Foundation under Grant No. IRI-9117030.

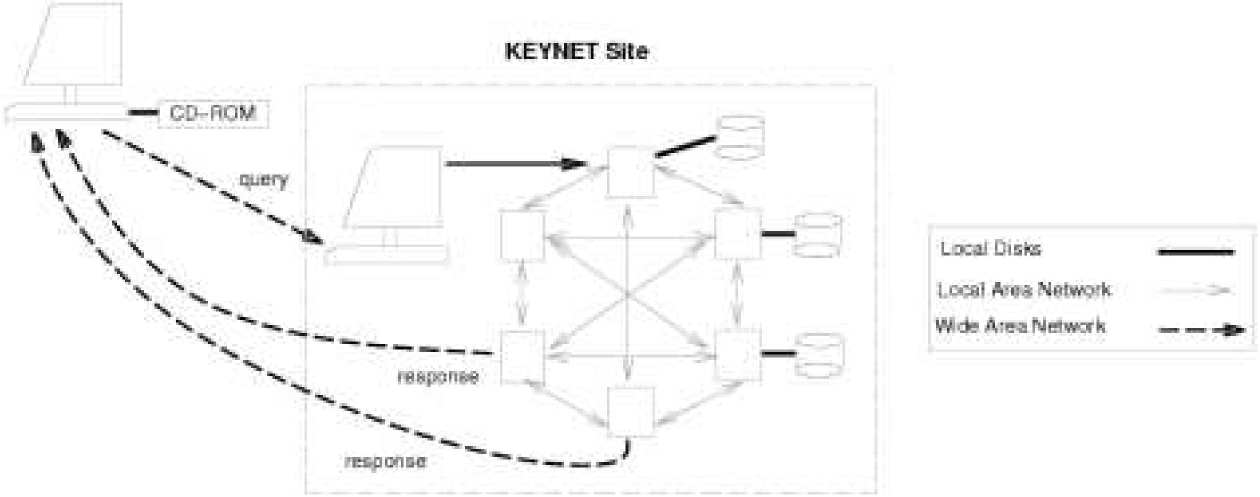

Figure 1: Architecture of KEYNET Search Engine

of textual document, such as research papers in a single discipline, can also be supported. The information objects may be physically located anywhere in the network. Retrieval is accomplished by means of a *content label* for each information object. These content labels are stored in a repository at the KEYNET site. The structure of the content labels is specified by an information model or *ontology*. The ontology can be as simple as a collection of document attributes such as "Author," "Title," "Publication Date," and so on, or it can be as complex as a general semantic network as used in artificial intelligence[Leh92]. The content labels are indexed by means of a distributed hash table stored in the main memories of a collection of processors at the KEYNET site. These processors form the *search engine*.

To see more precisely where all of these components reside, and how they are connected to one another, refer to Figure 1. The user's computer is in the upper left. A copy of the ontology is kept locally at the user site. As this will require from several hundred megabytes to a few gigabytes of memory, it would generally be stored on a CD-ROM. The ontology is also the basis for the user interface to the search engine. Queries must conform to the format specified by the ontology, and are sent over the network to a front-end processor at the KEYNET site. Responses are sent back over the network to the user's site, where they are presented to the user using the ontology. The prototype system uses a connectionless communication protocol so that no connection is required for making a query, and also so that the responses need not be sent back from the same computer that originally received the query.

At the KEYNET site, the front-end computer is responsible for relaying query requests to one of the search engine computers. The reason for having a front-end computer is mainly for distributing the workload but it also helps to simplify the protocol for making queries. The search engine itself is a collection of processors (or more precisely server processes) joined by a high-speed local area network. The search engine processors will be called *nodes*. The repository of content labels is distributed on disks attached to some of the nodes. The index to the content labels is distributed among the main memories of the nodes.

Since a connectionless communication protocol is unreliable, it is necessary for the user computer to resend the query if there is no response after a timeout period. The keynet protocol is stateless and idempotent, and so it works well with a connectionless communication service.

## 2 Distributed Algorithm

The basic indexing strategy is to match probes (fragments of queries) with index terms (fragments of content labels). We now discuss the details of the distributed algorithm that accomplishes this matching. This algorithm can be characterized as a "scatter-gather" technique. Queries are sent to a front-end processor in the form of datagrams. The front-end processor assigns a query id, acknowledges receipt of the query and forwards the query to a randomly chosen node of the search engine. This is the first scattering step. The node that is assigned the query is called the *home node* of the query.

At the home node, the query is broken apart into probe fragments. For the details of the fragmentation algorithm, see [BS94a]. For the purposes of this article, one can regard the fragments as small pieces of the original query. These fragments overlap one another, especially in semantically complex queries.

Each probe is then hashed using a standard hashing algorithm. The hash value is in two parts. One part is a node number and the other part is the local hash value used at that node. The local hash value and the query identifier are then sent to the node that was selected by the hash value. This forms the second scatter step of the algorithm. The result of hashing is to scatter the probes uniformly to all of the nodes of the search engine.

Upon receiving the local hash value of a probe, the node looks it up in its local hash table. An index term in the hash table that matches a probe is called a *hit*. The hits are sent back to the home node of the query. This is the "gather" step of the algorithm. Special trailer messages are used for determining when all the hits of all the probes of a query have been collected. The home node then computes the similarity measure (using an IR measure of similarity called the cosine measure) of each object in the collection, and the objects are ordered by the degree of similarity to the query. The object identifiers of the most relevant objects are then sent back to the user.

The insertion of a new content label in the index is done in a manner very similar to the query algorithm. Since content labels and queries are both keynets conforming to the same keynet ontology, they use exactly the same data structure. The same fragmentation, hashing and scattering algorithms are used for content labels as for queries. The only difference is that instead of matching entries in the hash table, index terms are inserted into the table. Note that index terms are not explicitly stored in the index, just their hash values are stored. The number of bits in the hash value is chosen to be so large that it is very unlikely that two probes would have the same hash value. As a result it suffices to store only a hash value and not the index term itself. Since an index term is nearly always much larger than a local hash value, this results in a significant savings of space with only a slight reduction in retrieval

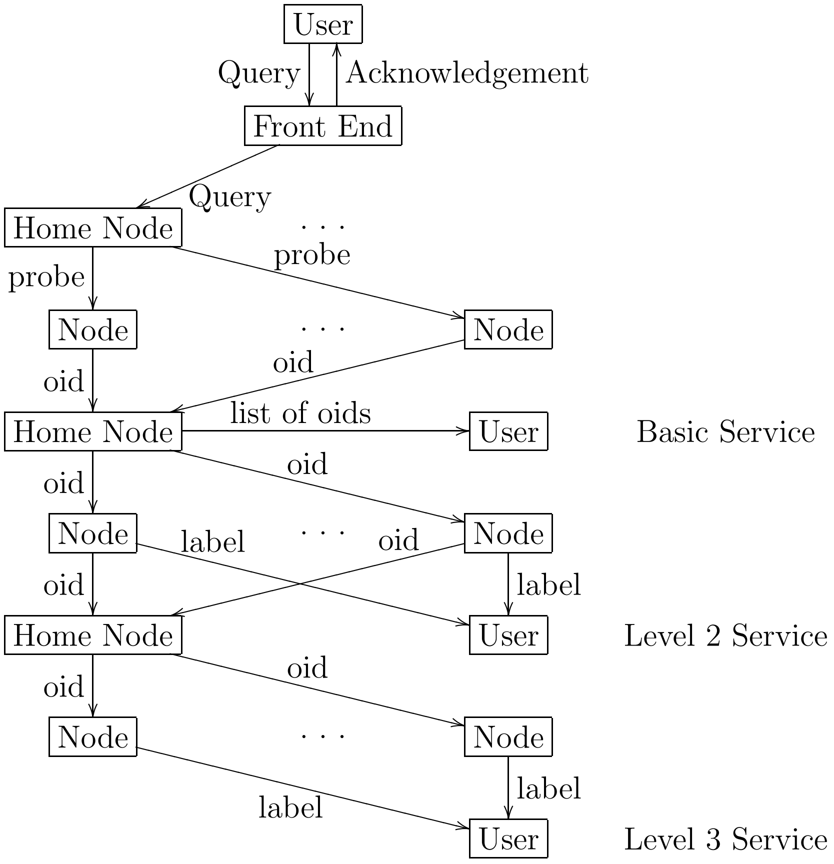

Figure 2: The KEYNET Distributed IR Algorithm

effectiveness.

The query algorithm presented so far represents the *basic* level of service. Higher levels of service are provided by using additional scatter-gather operations. The overall structure of the various levels of service is shown in Figure 2. The second level of service uses two scatter-gather operations. After completing the collection phase of the basic level of service, the home node sends each object identifier to the node where its content label is stored, resulting in yet another scatter step. The content label is then retrieved and sent back to the user's computer where the content labels are gathered, arranged and displayed using the keynet ontology.

The highest level of service is level 3. Instead of sending the content labels back to the user as in level 2, the nodes perform a structural analysis of the content labels each of which is compared with the original query using subgraph isomorphism techniques. The result of this analysis is a new estimate of the degree of relevance of each information object with the original query. These new estimates are sent back to the home node which gathers them and constructs a new ranking of the object identifiers. In the process the least relevant objects will be dropped from the list. The object identifiers are then sent back to the nodes where the content labels reside so that the content labels can be sent to the user with their final ranks.

The KEYNET algorithm is a “shared nothing” algorithm: each processing node has responsibility for its own local memory (both main memory and disk storage) and there is no shared memory. This is in contrast with shared memory paradigms for parallel and distributed computation, such as the Linda model [CG89]. Nevertheless, the KEYNET does share some aspects with the Linda model. For example, communication is one-way and messages can be processed in a different order than they were sent. Furthermore, the messages of KEYNET are tuples, and while it is important that a message be processed at a particular node, it is not important which thread processes it at that node.

# 3 Reliability

An important characteristic of the KEYNET system is its reliability. Unlike many distributed algorithms which are sensitive to the possibility that information might be lost, the KEYNET algorithm can tolerate the loss and duplication of packets as well as the failure of one of the nodes of the search engine. Such occurrences may cause some degradation in retrieval effectiveness, but they do not stop query processing. The advantage of being able to tolerate communication failures is that a connectionless communication protocol can be used. Such a protocol is unreliable but has better performance than a connection-oriented protocol.

We give some examples to illustrate how the algorithm can tolerate communication and node failures. Suppose the user’s original query packet were lost. The user’s computer would fail to receive the acknowledgement, and it would time out and send the query again. If the acknowledgement were lost, then the user’s computer would send the query packet again. The effect is exactly the same as the duplication of the query packet. Since KEYNET is a stateless, idempotent server, there is no harm in requesting the same query twice, Since responses will be labeled with the query identifier, the user’s computer would be able to detect and to discard the spurious responses.

Within the KEYNET search engine, loss or duplication of packets is rare since communication is over a local area network, but they can still occur. Loss of a probe effectively deletes the part of the query that is represented by the probe. Because the fragments of a query overlap one another, the probes are redundant. The role played by the lost or duplicated probe is therefore likely to be compensated by probes that overlap it. In a similar way, the loss of a hit message may change the overall ordering of the information objects that are retrieved but since the retrieval of a document generally requires several hits for it to be considered sufficiently relevant, one can tolerate the loss of one hit.

The loss of a node is a more serious failure than just the loss of a message. It results in the loss of a large number of probes and hits. However, if there are many nodes, then the effect on any one query is relatively small on the average. So even this kind of failure can be tolerated in much the same way that a simple loss of a message can be tolerated.

# 4 Performance

We have developed a prototype KEYNET system that runs on a network of sparcstations connected by a local area network. The sparcstations are part of the network of Unix workstations maintained by the College for a large community of faculty, students, staff and guests. We ran our tests late at night on relatively unused workstations. There was less activity on the network at these times, but there was some activity even at 3:00 AM, so our results exhibit a variance that reflects this.

The largest search engine we have used consisted of an 8-node network. On each node we index the content labels of 20,000 information objects. The individual nodes are implemented as servers. Specifically, they are implemented as connectionless, multi-threaded, interrupt-driven, stateless servers. Each server is responsible for a fixed amount of memory, 16 Mbytes, which is small enough for page faulting to be rare so that, to a first approximation, the 16 Mbytes can be regarded as physical memory.

Messages between nodes are buffered and sent in groups to improve throughput at the expense of some response time. The amount of buffering can be adjusted so as to maximize throughput for a given response time requirement. In the test runs, the buffer size was adjusted for each configuration so that the frequency with which packets are being sent is approximately the same. Otherwise, when one varies the number of nodes, all one is measuring is the effect of the buffer size. We have a mechanism for flushing buffers when this is deemed to be appropriate. Load balancing is done by measuring the relative performance of the nodes at the beginning of each run, and then allocating tasks to the nodes using this measurement.

In figures 3 and 4, we show the median, $90^{th}$ and $95^{th}$ percentile response times versus throughput for 4- and 8-node search engines, respectively. The 4-node search engine has a better response-time than the 8-node search engine for lower values of the throughput, but for higher values of the throughput, the 4-node response-time gets so large that it goes off the scale. The 8-node engine can can sustain a much higher throughput before the response-time goes off the scale.

# 5 Summary

A distributed algorithm has been developed for high-performance information retrieval from a subject-specific corpus of information objects. A prototype system has been developed to measure the performance characteristics of the algorithm, and the prototype has achieved a throughput of 500 queries per second. The algorithm uses a local memory model and connectionless communication both of which are very useful for scaling up to larger corpora. The algorithm supports several levels of service and can tolerate a small number of communication and processor failures.

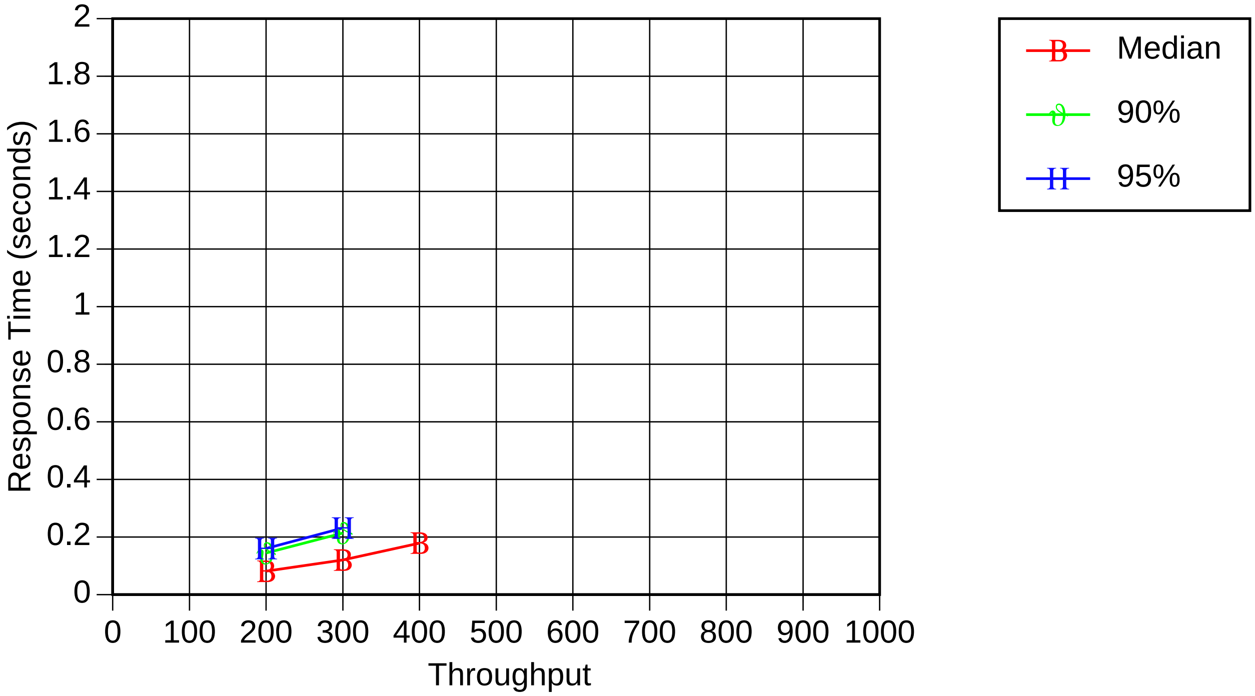

Figure 3: Response-Time versus Throughput on a Four-Node Engine

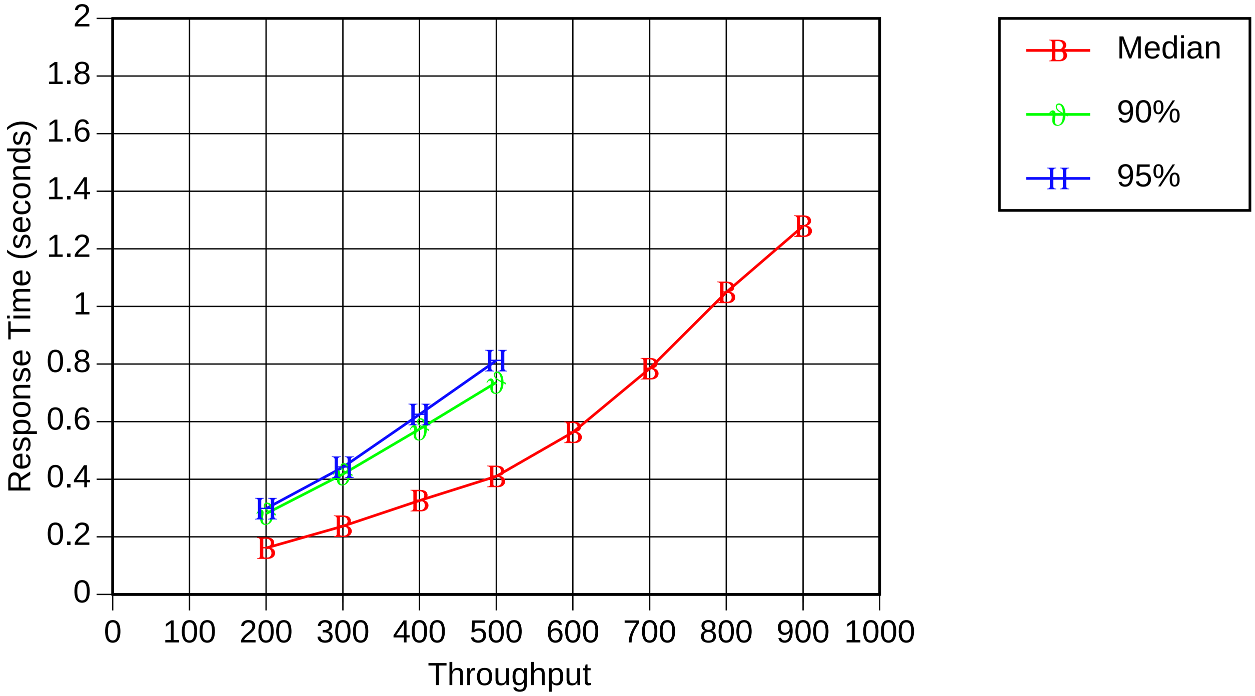

Figure 4: Response-Time versus Throughput on an Eight-Node Engine

# References

[BS94a] K. Baclawski and D. Simovici. An abstract model for semantically rich information retrieval. *Information Systems*, 1994. to be submitted.

[BS94b] K. Baclawski and J. E. Smith. High-performance, distributed information retrieval. Technical Report NU-CCS-94-05, Northeastern University, College of Computer Science, 1994.

[BS94c] K. Baclawski and J. E. Smith. `KEYNET`: Fast indexing for semantically rich information retrieval. Technical Report NU-CCS-94-06, Northeastern University, College of Computer Science, 1994.

[BS94d] K. Baclawski and J. E. Smith. A unified approach to high-performance, vector-based information retrieval. In *RIAO'94*, 1994. submitted.

[CG89] N. Carriero and D. Gelernter. Linda in context. *Comm. ACM*, 32:444–458, 1989.

[Leh92] Fritz Lehmann. Semantic networks in artificial intelligence, parts I and II. *Computers and Mathematics with Applications*, 23(2-9), 1992.