SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks

Brian F. Cooper and Hector Garcia-Molina

Department of Computer Science
Stanford University
Stanford, CA 94305 USA
{cooperb,hector}@db.Stanford.EDU

Abstract. The popularity of peer-to-peer search networks continues to grow, even as the limitations to the scalability of existing systems become apparent. We propose a simple model for search networks, called the *Search/Index Links* (SIL) model. The SIL model describes existing networks while also yielding organizations not previously studied. Using analytical and simulation results, we argue that one new organization, *parallel search clusters*, is superior to existing supernode networks in many cases.

1 Introduction

Peer-to-peer search networks have become very popular as a way to effectively search huge, distributed data repositories. On a typical day, systems such as Kazaa support several million simultaneous users, allowing them to search hundreds of millions of digital objects totaling multiple petabytes of data. These search networks take advantage of the large aggregate processing power of many hosts, while leveraging the distributed nature of the system to enhance robustness. Despite the popularity of peer-to-peer search networks, they still suffer from many problems: nodes quickly become overloaded as the network grows, and users can become frustrated with long search latencies or service degradation due to node failures. These issues limit the usefulness of existing peer-to-peer networks for new data management applications beyond traditional multimedia file sharing.

We wish to develop techniques for improving the efficiency and fault tolerance of search in networks of autonomous data repositories. Our approach is to study how we can place indexes in a peer-to-peer network to reduce system load by avoiding the need to query all nodes. The scale and dynamism of the system, as large numbers of nodes constantly join and leave, requires us to re-examine index replication and query forwarding techniques.

However, the space of options to consider is complex and difficult to analyze, given the bewildering array of options for search network topologies, query routing and processing techniques, index and content replication, and so on. In order to make our exploration more manageable, we separate the process into two phases. In the first phase, we construct a coarse-grained *architectural model* that describes the topology of the connections between distributed nodes, and models the basic query flow properties and index placement strategies within this topology. In the second phase, we use the insights gained from the architectural model to develop a finer-grained *operational model*, which describes at a lower level the actual processing in the system. The operational model allows us to study alternatives for building and maintaining the topology as nodes join and leave, directing queries to nodes (for example, using flooding, random walks or routing indices), parallel versus sequential query submission to different parts of the network, and so on.

Our focus in this paper is on the first phase architectural model. We have developed the Search/Index Link (SIL) model for representing and visualizing peer-to-peer search networks at the architectural level. The SIL model helps us to understand the inherent properties of many existing network architectures, and to design and evaluate novel architectures that are more robust and efficient. Once we understand which architectures are promising, ongoing work can examine operational issues. For example, in [6], we examine the operational question of how the architectures described here might be constructed. In this paper, we first present and analyze the SIL model, and show how it can lead to new search network architectures. Then, using analytical and simulation results, we show that our new organizations can be superior to existing P2P networks in several important cases, in terms of both efficiency and fault tolerance.

2 The Search/Index Link model

A *peer-to-peer search network* is a set of peers that store, search for, and transfer digital documents. We consider here content-based searches, such as keyword searches, metadata searches, and so on. This distinguishes a peer-to-peer search network from a distributed hash table [21, 18], where queries are to locate a specific document with a specific identifier (see Section 7 for more discussion about SIL versus DHTs). Each peer in the network maintains an index over its content (such as an inverted list of the words in each document) to assist in processing searches. We assume that the index is sufficient to answer searches, even though it does not contain the whole content of the indexed documents.

The search network forms an *overlay* on top of a fully-connected underlying network infrastructure. The topology of the overlay determines where indexes are placed in the network, and how queries reach either a data repository or an index over that repository's content. Peers that are neighbors in the overlay are connected by network links that are logically persistent, although they may be implemented in a connection-oriented or connectionless way.

The Search/Index Link (SIL) model allows us to describe and visualize the overlay topology. In the SIL model, there are four kinds of network links, distinguished by the types of messages that are sent, and whether a peer receiving a message forwards the message after processing it:

- A *non-forwarding search link* (NSL) carries search messages a single hop in the overlay from their origin. For example, a search generated at one peer A will be sent to another peer B, but not forwarded beyond B. Peer B processes each search message and returns results to A.

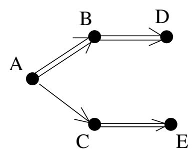


Fig. 1. A network with search links.

- A *forwarding search link* (FSL) carries search messages from A to B. Peer B will process each search message, return search results to A, and forward the message along any other forwarding search links originating at B. If A is not the originator of the query, it should forward any search results received from B (and any other nodes) along the FSL on which A received the query. Each search message should have a unique identifier that is retained as the message is forwarded. When a peer receives a search message with an id it has previously seen, the peer should discard the message without processing or forwarding it. This will prevent messages from circulating forever in the network if there is a cycle of FSLs.
- A *non-forwarding index link* (NIL) carries index update messages one hop in the overlay from their origin. That is, updates occurring at A will be sent to B, but not forwarded. Peer B adds A's index entries to its own index, and then effectively has a copy of A's index. Peer B need not have a full copy of A's content.
- A *forwarding index link* (FIL) carries index update messages from A to B, as with non-forwarding index links, but then B forwards the update message along any other forwarding index links originating at B. As with FSLs, update messages should have unique ids, and a peer should discard duplicate update messages without processing or forwarding them.

Network links are directed communications channels. A link from peer A to peer B indicates that A sends messages to B, but B only sends messages to A if there is also a separate link from B to A. Modeling links as directed channels makes the model more general. An undirected channel can of course be modeled as a pair of directed links going in opposite directions. For example, the links in Gnutella can be modeled as a pair of forwarding search links, one in each direction. Although forwarding links may at first glance seem more useful, we will see later how non-forwarding links can be used (Section 3).

Figure 1 shows an example network containing search links. Non-forwarding search links are represented as single arrows ($\longrightarrow$) while forwarding search links are represented as double arrows ($\Longrightarrow$). Imagine that a user submits a query to peer A. Peer A will first process the query and return any search results it finds to the user. Node A will then send this query to both B and C, who will also process the query. Node B will forward the query to D. Node C will not forward the query, since it received the query along an NSL. The user's query will not reach E at all, and E's content will not be searched for this query.

A peer uses an *index link* to send copies of index entries to its neighbors. These index entries allow the content to be searched by the neighbors without the neighbors

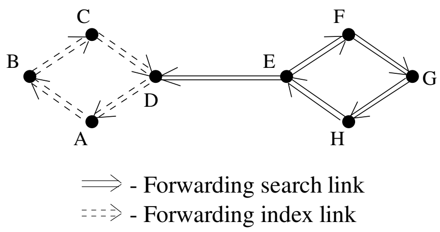


Fig. 2. A network with search and index links.

having to store the peer's actual content. For example, consider a peer A that has an index link to a peer B. When B processes a query, it will return search results both for its own content as well as for the content stored at A. Peer A need not process the query at all. We say that B is *searched directly* in this case, while A is *searched indirectly*.

Whenever a peer creates a new index entry or modifies an existing entry, it should send a message indicating the change along all of its outgoing index links. A peer might create an index over all of its locally stored documents when it first starts up, and should send all of the index entries to each of its index link neighbors. Similarly, if a node deletes a document, it would remove the corresponding entries from its own index as well as notifying its index link neighbors to do the same.

Figure 2 shows a network that contains both search and index links. ndex links are represented as dashed lines, single (- - ->) for non-forwarding index links and double (= = =>) for forwarding index links. (Note that Figure 2 contains only FILs.) Nodes A, B, C and D are connected by a "ring" of FILs. An index update occurring at peer A will thus be forwarded to B, C, D and back to A (A will not forward the update again). In fact, all four of the nodes $A...D$ will have complete copies of the indexes at the other three nodes in the index "ring". Nodes E, F, G and H are connected by FSLs, and a search originating at any peer $E...H$ will reach, and be processed by, the three other nodes on the search "ring." Notice that there is also an FSL between E and D. Any query that is processed by E will be forwarded to D, who will also process the query. Since D has a copy of the indexes from $A...C$, this means that any query generated at E, F, G and H will effectively search the content of all eight nodes in the network. In contrast, a query generated at nodes $A...D$ will be processed at the node generating the query, and will only search the indexes of the nodes $A...D$.

For the rest of our discussion, it is useful to define the concept of a *search path*:

Definition 1. *A* search path *from peer X to peer Y is*

- *a (possibly empty) sequence of FSLs $f_1, f_2, ...f_n$ such that f_1 originates at X, f_n terminates at Y, and f_i terminates at the same node at which f_{i+1} originates, or*
- *an NSL from X to Y*

A search path from X to Y indicates that queries submitted to X will eventually be forwarded to Y. For example, in Figure 2 there is a search path from F to D but not from D to F. Note also that there is (trivially) a search path from a node to itself.

Similarly, an *index path* from X to Y is a sequence of FILs from X to Y, or one NIL from X to Y. In this case, X's index updates will be sent to Y, and Y will have a copy of X's index.

2.1 "Good" networks

The network links we have discussed above are not by themselves new. Forwarding search links are present in Gnutella, forwarding index links are used in publish/subscribe systems, non-forwarding index links are used in supernode networks, and so on. However, different link types tend to be used in isolation or for narrowly specific purposes, and are rarely combined into a single, general model. Our graphical representation allows us to consider new combinations. In fact, the number of search networks of n nodes that can be constructed under the SIL model is exponential in n^2. Only a small fraction of these networks will allow users to search the content of most or all the peers in the network, and an even smaller fraction will also have desirable scalability, efficiency or fault tolerance properties. We want to use the SIL model to find and study "good" networks, and this of course requires defining what we mean by "good."

First, we observe that a search network only meets users' needs if it allows them to find content. Since content may be located anywhere in the network, a user must be able to effectively search as many content repositories as possible, either directly or indirectly. We can quantify this goal by defining the concept of *coverage*.

Definition 2. *The* coverage *of peer p in a network N is the fraction of the peers in N that can be searched, either directly or indirectly, by a query generated by p.*

Ideal networks would have *full coverage*:

Definition 3. *A network N has* full coverage *if every peer p in N has* coverage $= 1$.

A full coverage network is ideal in the sense that if content exists anywhere in the network, users can find it. It may be necessary to reduce coverage in order to improve network efficiency.

Even a network that has full coverage may not necessarily be "good." Good networks should also be efficient, in the sense that peers are not overloaded with work answering queries. One important way to improve the efficiency of a network is to reduce or eliminate redundant work. If peers are duplicating each other's processing, then they are doing unnecessary work.

Definition 4. *A search network N has* redundancy *if there exists a network link in N that can be removed without reducing the coverage for any peer.*

Intuitively, redundancy results in messages being sent to and processed by peers, even when such processing does not add to the network's ability to answer queries.

Redundancy can manifest in search networks in four ways:

- *Search/search redundancy* occurs when the same peer P processes the same query from the same user multiple times.
- *Update/update redundancy* occurs when the same peer P processes the same update multiple times.

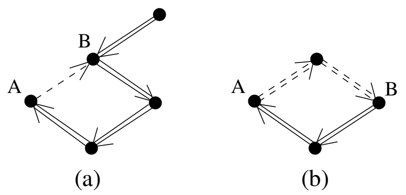


Fig. 3. Networks with cycles: a. with search/index redundancy, and b. no search/index redundancy.

- *Search/index redundancy* means a peer A processes a query even though another peer B has a copy of A's index and processes the same query.
- *Index/index redundancy* is where two different peers B and C both process a search over a copy of a third peer A's index.

In each of these cases, a node is doing work that is unnecessary to achieve high or full coverage.

Note that redundancy may actually be useful to improve the fault tolerance of the system, since if one node fails another can perform its processing. Moreover, redundancy may be useful to reduce the time a user must wait for search results, if a node near the user can process the user's search even when this processing is redundant. However, fault tolerance and search latency tradeoff with efficiency, since redundancy results in extra work for peers.

2.2 Topological features of networks with redundancy

The concept of "redundancy" and even the subconcepts like search/index redundancy are quite general. Rather than avoiding generalized redundancy when designing a peer-to-peer search network, it is easier to identify specific features of network topologies that lead to redundancy, and avoid those features.

One feature that causes search/index redundancy is a specific type of cycle called a *one-cycle*. One version of a one-cycle is a *one-index-cycle*: a node A has an index link to another node B, and B has a search path to A. An example is shown in Figure 3a. This construct leads to redundant processing, since B will answer queries over A's index, and yet these queries will be forwarded to A who will also answer them over A's index. More formally, a one-index-cycle fits our definition of *redundancy* because at least one link in the cycle can be removed without affecting coverage: the index link from A to B. Another version of a one-cycle is a *one-search-cycle*, which is when a node A has an search link to another node B, and B has an index path to A. While *one-cycles* (one-index-cycles and one-search-cycles) cause redundancy, not all cycles do. Consider the cycle in Figure 3b. This cycle may seem to introduce redundancy in the same way as a one-cycle, except that none of the links can be removed without reducing coverage for some node.

Another feature that causes search/index redundancy is a *fork*. A *search-fork* is when a node C has a search link to A and a search path to B that does not include A, and there is an index path from A to B. An example is shown in Figure 4a. Again, A will

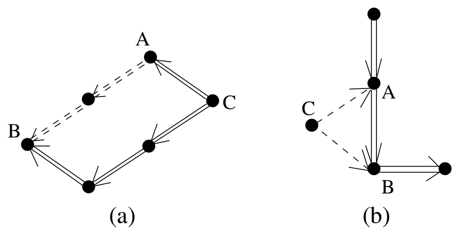


Fig. 4. Networks with forks: a. a search-fork, and b. an index-fork.

process any searches from C unnecessarily, since B can process the queries for A. The redundant link in this example is the link $C \Longrightarrow A$. We specify that there is a search path from C to B that does not include A because if the only path from C to B included A there would be no link that could be removed without reducing coverage. The analog of a search-fork is an *index-fork*: a node C has an index link to A and an index path to B that does not include A, and there is a search path from A to B. An example is shown in Figure 4b.

The third feature that causes redundancy is a *loop*:

- A *search-loop* is when a node A has an search link l_s to another node B, and also another search path to B that does not include l_s.
- An *index-loop* is when a node A has an index link l_i to another node B, and also another index path to B that does not include l_i.

Avoiding all of these topological features is sufficient to avoid the general property of redundancy in a network.

Theorem 1. *If a network has no one-cycles, forks or loops, then it has no redundancy.*

Proof. We show that redundancy implies at least one of the named features. A redundant network has a link l from A to B that can be removed without reducing coverage for any node. There are two cases: either l is an index link, or l is a search link. If l is an index link, then B was able to search A. When l is removed, B must still be able to search A. Node B can search A *directly* via a search path from B to A, which with l would have formed a one-index-cycle. Or B can search A *indirectly*:

- With a search path to a node C such that there is an index path from A to C. The search path from B to C, the index path from A to C and the link l formed an index-fork.
- With an index path from A to B other than l. This index path plus l formed an index-loop.

The second case is that l is a search link. This means that A could search B directly, and must still be able to search B after l is removed. This can be done *directly*, with a search path to B, which with l would have formed a search-loop. Or, *indirectly*:

- A may have a search path to D, such that there is an index path from B to D. The search path from A to D, plus l, plus the index path from B to D, would have formed a search-fork.

- B may have an index path to A. This index path, plus l, would have formed a one-search-cycle.

□

3 Network archetypes

We can now identify some archetypical network organizations described by the SIL model. Each archetype is a family of topologies that share a common general architecture. We restrict our attention to somewhat idealized networks, that is, non-redundant networks with full coverage, in order to understand the inherent advantages and disadvantages of various architectures. We do not claim to examine the entire design space of peer-to-peer topologies. Instead, by looking at some representative archetypes of a particular design point, that is, non-redundant full-coverage networks, we can both understand that design point clearly and also illustrate the value of SIL as a design tool.

We consider only the static topologies described by the SIL architectural model, in order to determine which topologies have efficiency or fault tolerance benefits and are worth examining further. If a particular archetype is selected for a given application, there are then operational decisions that must be made. For example, if a supernode archetype (described fully below) is chosen as desirable, there must be a way to form peers into a supernode topology as they join the system. One way to form such a network is to use a central coordinator that selects which nodes are supernodes and assigns them responsibility for non-supernodes. Alternatively, nodes could decide on their own whether to be supernodes or not, and then advertise their supernode status to connect to other, non-supernode peers. This dynamic process of forming a specific topology is outside the scope of this paper, as we wish to focus for now on which topology archetype is most desirable under various circumstances. For a discussion on how a topology can be constructed dynamically, see [6, 24].

Also, we focus on networks with no search/index or index/index redundancy. The impact of search/search and update/update redundancies is mitigated by the fact that a node processes only one copy of a duplicate search or update message and discards the rest (see Section 2). In contrast, search/index and index/index redundancies involve unnecessary work being done at two different peers, and it is difficult for those peers to coordinate and discover that their work is redundant. Therefore, in order to reduce load it is important to design networks that do not have search/index and index/index redundancies. To do this, we consider networks that do not have one-cycles or forks.

First, note that there are two basic network archetypes that can trivially meet the conditions of no search/index or index/index redundancy while providing full coverage:

- *Pure search networks*: strongly connected networks with only search links.
- *Pure index networks*: strongly connected networks with only index links.

In graph theory, a *strongly connected directed graph* is one in which there is a directed path from every node to every other node. Recall from Section 2 that in our SIL model, a path is either a sequence of forwarding links or a single non-forwarding link. When we say "strongly connected" in the definitions above (and below), we mean "strongly connected" using this definition of search and index paths.

In these basic topologies, there cannot be search/index or index/index redundancies since index links and search links do not co-exist in the same network. However, these networks are not "efficient" in the sense that nodes are lightly loaded. In a pure search network, every node processes every search, while in a pure index network, every node processes every index update. These topologies may be useful in extreme cases; for example, a pure search network is not too cumbersome if there are very few searches. A well known example of a pure search network is the Gnutella network.

Other archetypes combine search links and index links to reduce the load on nodes. We have studied four topology archetypes that are described by the SIL model, have full coverage and no search/index or index/index redundancy:

- Supernode networks
- Global index networks
- Parallel search cluster networks
- Parallel index cluster networks

As we discuss in more detail below, each different topology is useful for different situations. Some of these topologies are not new, and exist in networked systems today. Supernode networks are typified by the FastTrack network of Kazaa, while the global index network is similar to the organization of Netnews with a central indexing cluster (like DejaNews). However, the parallel search and index clusters have not been previously examined. While these four archetypes are just a sample of the topologies that can be described by SIL, they illustrate how SIL can be used to model a variety of networks with different characteristics.

A *supernode network* is a network where some nodes are designated as "supernodes," and the other nodes ("normal nodes") send both their indexes and searches to supernodes. The supernodes are linked by a strongly connected pure search network. A supernode network can be represented in our SIL model by having normal nodes point to supernodes with one FSL and one NIL, while supernodes point to each other using FSLs. An example is shown in Figure 5a. Each supernode therefore has the copies of several normal nodes' indexes. Supernodes process searches before forwarding them to other supernodes. Normal nodes only have to process searches that they themselves generate. Thus, supernodes networks result in much less load on an average peer than a pure search network. A disadvantage is that as the network grows, the search load on supernodes grows as well, and ultimately scalability is limited by the processing capacity of supernodes. This disadvantage exists even though there is unused processing capacity in the network at the normal nodes. These normal nodes cannot contribute this spare capacity to reduce the search load on supernodes, because even if a normal node is promoted to a supernode, every supernode must still process all the queries in the network. Supernode networks are most useful when search load is low and when there are nodes in the network powerful enough to serve as supernodes.

An organization similar to supernodes is a *global index network*, as illustrated in Figure 5b. In this organization, some nodes are designated as global indexing nodes, and all index updates in the system flow to these nodes. A normal node sends its queries to

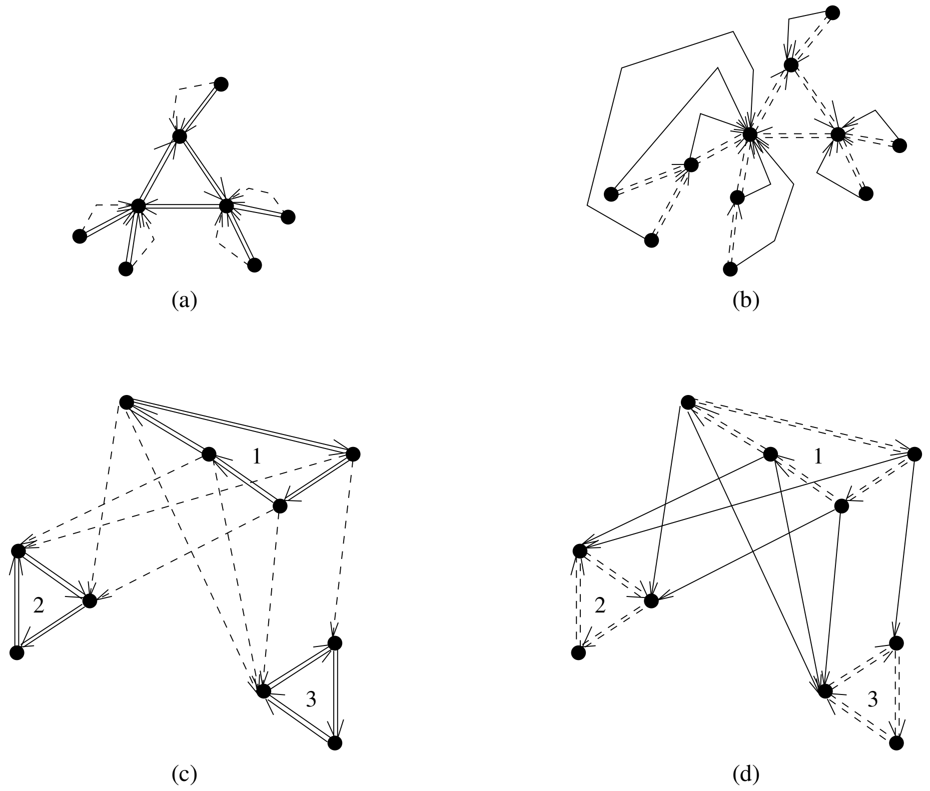


Fig. 5. Topology archetypes: a. Supernodes, b. Global index, c. Parallel search clusters, and d. Parallel index clusters. Some inter-cluster links are omitted in networks c and d for clarity.

one of these global indexing nodes. The global indexing nodes themselves are connected by a strongly connected pure index network. Under our model, normal nodes have a FIL to another normal node or to a global index node, and normal nodes also have NSLs to global index nodes. In this example, the normal nodes form a tree of index paths rooted at a global index node. Index updates flow from the normal nodes to form a complete set of global indexes at each of the global index nodes. Note that a similar tree-like structure could be constructed in the supernode network, where normal nodes would form a tree of search paths rooted at a supernode, while each normal node would have an index link directly to a supernode.

The advantages of global index networks are similar to those of supernode networks. Most nodes process only index updates and their own searches, while leaving the processing of all other searches to the global index nodes. Moreover, there are multiple nodes that have a complete set of indexes, so the network can recover from the failure of one node. However, the load on the global index nodes is high; each global index peer must process all the index updates in the system and a significant fraction of the searches. Global index networks are most useful when update load is low and when there are nodes in the network powerful enough to serve as index nodes.

A third organization is called *parallel search clusters*. In this network, nodes are organized into clusters of strongly connected pure search networks (consisting of FSLs), and clusters are connected to other clusters by NIL index links. An example is shown in Figure 5c. In this figure, the cluster “1” has outgoing NILs to the other clusters “2” and “3”. Clusters “2” and “3” would also have outgoing NILs to the other clusters, but we have omitted them in this figure for clarity. The nodes in each cluster collectively have a copy of the indexes of every node outside the cluster, so full coverage is achieved even though queries are only forwarded within a cluster. Unlike in a supernode topology, there are no nodes that must handle all of the queries in the network. Nodes only handle queries that are generated within their own cluster. Moreover, all of the search processing resources in the system are utilized, since every node processes some queries. A disadvantage of this topology is that nodes must ship their index updates to every other cluster in the network. If the update rate is high, this will generate a large amount of update load. In Section 4.2, we discuss how to tune the cluster network to minimize the update load. Parallel search clusters are most useful when the network is relatively homogeneous (in terms of node capabilities), and when the update rate is low.

Finally, the fourth organization is *parallel index clusters*. In this organization, clusters of strongly connected pure FIL index networks are connected by NSL search links. As a result, nodes in one cluster send their searches to one node of each of the other clusters. An example is shown in Figure 5d. (Again, some inter-cluster links are omitted in this figure.) Parallel index clusters have advantages and disadvantages similar to parallel search cluster networks: no node handles all index updates or all searches, and all resources in the system are utilized, while inter-cluster links may be cumbersome to maintain and may generate a large amount of load. Index cluster networks are useful for relatively homogeneous networks where the search rate is low.

These topology archetypes can be varied or combined in various ways. For example, a variation of the supernode topology is to allow a normal node to have an FSL pointing to one supernode and an NIL pointing to another. Another example is to vary parallel cluster search networks by allowing the search clusters to be constructed as mini-supernode networks instead of (or in addition to) clusters that are pure search networks. These and other variations are useful in certain cases. Allowing a mini-supernode network as a search cluster in a parallel search cluster network is useful if the nodes in that cluster are heterogeneous, and some nodes have much higher capacities than the others. Moreover, pure index and pure search networks are special cases of our four topology archetypes. For example, a supernode network where all nodes are supernodes and a parallel search cluster network where there is only one cluster are both pure search networks.

Note that our restriction of no redundancy can be relaxed to improve the fault tolerance or search latency of the system at the cost of higher load. For example, in a supernode network, a normal node could have an NIL/FSL pair to two different supernodes. This introduces, at the very least, an index/index redundancy, but ensures that the normal node is still fully connected to the network if one of its supernodes fails. Similarly, the goal of full coverage could be relaxed to reduce load. For instance, in many real networks, messages are given a time-to-live so that they do not reach every node. This results both in lower coverage and lower load.

4 Evaluation of network topologies

We quantify the strengths and weaknesses of the various topology archetypes in three steps. First, we define metrics that are computable over SIL graphs. Then, we use these metrics to evaluate analytically the strengths and weaknesses of different idealized architectures. Finally, we run simulations to validate our analytical results for less idealized networks. In this section, we perform the analytical steps, while Section 5 discusses our simulation results.

For the sake of brevity, we do not compare all of the possible archetypes described by the SIL model. Instead, we choose two archetypes: supernode networks, which represent a popular and widely deployed existing architecture, and parallel search clusters, which is a promising new architecture that we discovered from analysis of SIL. Our results show that search clusters are more efficient than supernode networks in some important scenarios, and this illustrates the value of the SIL model as a tool for discovering new architectures with desirable properties.

4.1 Metrics

First, we can define metrics that measure the amount of load on individual peers:

Definition 5. Search load *is the load on peers from processing searches. This is measured as the number of search messages that reach peers per unit time.*

Definition 6. Update load *is the load on peers from processing index updates sent via index links from other peers. This is measured as the number of index messages that reach peers.*

Definition 7. Total load *is the sum of the search load and update load.*

These metrics can be computed by analyzing SIL graphs. For example, the number of search messages that reach a peer A can be computed by finding all the peers that have a search path to A, and taking the sum of the search message rates for those peers. Notice that we are using a very general definition for "load" which encapsulates both the load on the network links and the load on the peers themselves. This is because we wish to define a general architectural model that is independent of the physical characteristics of networks or machine architectures. Moreover, we are mainly concerned with the relative measure of these metrics (e.g., network X has less load than network Y) and not their absolute values. Therefore, we define load in terms of "numbers of messages," and make the simplifying assumption that all messages are equally costly to process. This assumption may not be true, for example, if it is twice as costly to process a search as an index update. However, this situation is equivalent in our model to one where there are twice as many search messages as index messages. In fact, we study situations where there are more search messages than update messages generated in the system, more update messages than search messages, and an equivalent number of search and update messages. This allows us to model both the situation where one kind of message is produced more frequently and the case where one kind of message is more expensive to process.

A robust network must also be able to survive node failures. We can measure the resistance to failures by defining the *fault susceptibility* of a network:

Definition 8. Fault susceptibility *is the maximum decrease in network coverage caused by the failure of any one node.*

This metric can be calculated by determining which node, when removed from a SIL graph, causes the coverage in that graph to decrease the most. We assume failures are fail-stop, so a node that fails ceases to process and forward messages. When this happens, the network coverage, measured as the fraction of the nodes in the network that are searchable, may decrease. For example, in a network with full coverage (e.g., *coverage* $= 1$) a failure may cause the network to partition into three subnetworks, each unreachable from the other. If the subnetworks are of equal size, the fault susceptibility is 0.66, since each node can only search one third of the network and the coverage thus drops to 0.33. We have called this metric "fault susceptibility" instead of the more traditional "fault tolerance" because we are concerned with the effect of a single node failure rather than with the number of failures the network can tolerate. Implemented networks should certainly have a fail-over mechanism for recovering from the failure. However, recovery is best treated as part of the operational model.

Finally, we note that user satisfaction is increased if queries return results quickly. To measure this effect, we can define the *search latency* metric:

Definition 9. *The* search latency for a peer p *is the longest search path that a query generated by* p *must travel.*

Intuitively, this metric represents the time that a user must wait before all of the search results are guaranteed to return. When a user submits a query to node p, it will be forwarded to every peer r_i such that there is a search path from p to r_i. The user may quickly get some results, but will only be assured of getting all results when that query reaches every node r_i. Since messages can travel separate paths in parallel, the maximum time for the query to reach every node r_i (and for search results to return) is proportional to the number of hops from p to the r_i that is farthest away.

Definition 10. *The* search latency for a network N *is the average search latency over all peers* $p_1, p_2 \ldots p_n$.

4.2 Load analysis

Let us now analyze search network topologies using our evaluation framework. We focus on load, both because the problem of making networks efficient and scalable is a significant challenge in building peer-to-peer search networks, and because load turns out to be particularly amenable to an analytical approach. We consider two of the archetypes from Section 3: supernodes, because they represent the most popular deployed search networks (for example, Kazaa), and our novel parallel search cluster networks, which use the same SIL primitives as supernode networks (that is, FSLs and NILs).

First, we can compute the optimum cluster size for an ideal cluster network if we know the search and update load patterns in the network. Let N be the number of nodes in the network, and C be the number of nodes in a cluster (assuming for simplicity that all clusters are the same size), $1 \leq C \leq N$. We call the average search load generated by a node S_L, and the average index update load generated by a node U_L; these loads can be measured in messages per unit time. Each node must process the search load generated by every node in its cluster (including itself), and thus must process a total search load of $C \times S_L$ messages per unit time. Assume that the nodes in the cluster divide the indexes of nodes outside the cluster equally among themselves, so that each node must store N/C indexes and handle $(N/C) \times U_L$ messages per unit time update load. Then the total load on an average node in a cluster network is

$$L_{cl} = C \times S_L + (N/C) \times U_L \tag{1}$$

If we differentiate this expression with respect to C, and set the result to zero, we find that L_{cl} is minimized by

$$C = \sqrt{N \times U_L / S_L} \tag{2}$$

This expression shows that the cluster size can be tuned depending on the load in the network: when U_L is high, C should be high (large clusters), while a high S_L means that C should be low (lots of small clusters). Substituting (2) into (1) gives us

$$L_{cl} = 2 \times \sqrt{N \times S_L \times U_L} \tag{3}$$

as the load on an average node in an optimized cluster network.

Note that if clusters are small, each node may have to connect to a large number of nodes in other clusters. Even if bandwidth is plentiful, a node may prefer not to have a large number of open connections or to have to know the identities of a large number of nodes. This issue can be solved by using a few forwarding index links to allow a node to send its index entries to some nodes, who then forward those updates to multiple other nodes.

We can perform a similar load analysis on supernode networks. Let P be the number of supernodes, $1 \leq P \leq N$. If we assume that normal nodes are divided equally among supernodes, then the load on a supernode is

$$L_{sn} = (N/P) \times U_L + N \times S_L \tag{4}$$

that is, the update load received from normal nodes connected to the supernode plus the search load from the whole network, since for full coverage each supernodes handle all searches in the network. The load on a normal (non-super) node is only the node's own load, or $L_{nn} = S_L + U_L$. In a network with P supernodes and $N - P$ normal nodes, the expected total load on a node is

$$L_{avg} = 2 \times U_L + S_L + P \times (S_L - U_L/N - S_L/N) \tag{5}$$

This equation is linear in P, and is minimized by minimizing P, e.g. $P = 1$. However, supernode networks are not usually constructed with a single supernode, as the load on

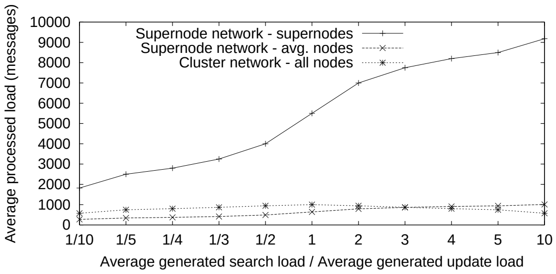


Fig. 6. Load of optimized networks.

this one central indexing node is too high. Usually, some small fraction of the network nodes, say one tenth, is chosen to serve as supernodes.

Using our expressions for L_{cl}, L_{sn} and L_{avg}, we can compare optimized cluster and supernode networks. Figure 6 shows a comparison of cluster networks and supernode networks for a situation where $P = N/10$. The horizontal axis in this figure shows different values of the ratio S_L/U_L, and the vertical axis shows the number of messages processed by nodes, assuming $S_L + U_L = 100$. As the figure shows, nodes in a cluster network are always less loaded than supernodes, by up to a factor of 16 (when $S_L >> U_L$). Nodes in a cluster network are only about twice as loaded as the average node in a supernode network, and sometimes (in the case of $S_L >> U_L$) cluster network nodes are less loaded than the average node in a supernode network. We can draw two conclusions from these results. First, if the search load is heavy and the update load is light, search clusters are more efficient than supernode networks for the average node in the network. Second, in all cases, cluster networks spread the work around more evenly than supernode networks, which is important when there are no "super-capacity" nodes that can take on most of the burden in the network. To confirm these analytical conclusions, we next examine simulations of networks.

5 Experimental results

Our analytical examination of architectures (Section 4) has shown that a new architecture, parallel search clusters, offers efficiency benefits over existing supernode architectures. To confirm this analysis, especially for less idealized and more realistic or "messy" topologies, we have run simulations. Our simulations also allow us to study other properties of networks, such as search latency and fault susceptibility.

5.1 Simulation setup

We have constructed a simulator to generate networks with a given topology, and to evaluate the metrics of load, fault susceptibility and search latency. Our simulator takes several input parameters, which are summarized in Table 1. Since we are focusing here on the operational model, we calculate metrics over static topologies and do not deal with nodes joining or leaving.

Parameter	*Description*	*Base value*
n	Number of nodes	100
P_A	Average links per node for PLOD	5
P_M	Maximum links per node for PLOD	10
NS	Number of supernodes	$1 \ldots n$
NC	Number of clusters	$1 \ldots n$
SL	Avg. search load generated by a peer	$10 \ldots 100$
UL	Avg. update load generated by a peer	$10 \ldots 100$

Table 1. Simulation parameters

For our experiments, we generated pure search, supernode and parallel search cluster networks. The details of generating topologies are described in Section 5.5. For each scenario, we generated 50 instances of each topology, and the results we report represent averaging our metrics over all of these instances. Each network instance had n nodes. Although all the results we report used the same number for n (see Table 1) we ran experiments with different values of n and observed comparable results. In each case, our results have 95 percent confidence intervals of $\pm$ 3 percent or less of the value reported, unless otherwise noted. We assume for simplicity that each node has equivalent bandwidth and processing capacity. We allowed search/search redundancy, since this redundancy reduces fault susceptibility and search latency without increasing load, and because real networks such as Gnutella and Kazaa also have search/search redundancy.

For all types of networks, we assigned two parameters to each peer p_i: SL_i, the amount of search load created by the peer, and UL_i, the amount of index update load created by the peer. The mean SL_i (denoted *SL*) and the mean UL_i (denoted *UL*) were specified per experiment, and the individual SL_i's and UL_i's were normally distributed with the given mean. The sum $SL + UL$ was constant across all experiments. By examining a topology and determining for each peer p which peers can send search and update traffic to p, we can calculate the total load for p (see Definition 7).

Definition 11. *The* load for a peer p *is the sum of:*

- *The* SL_i *for each peer* r_i *such that there is a search path from* r_i *to* p*, and*
- *The* UL_i *for each peer* r_i *such that there is an index path from* r_i *to* p.

Definition 12. *The* average load for a network N *is the average over all peers* $p_1, p_2 \ldots p_n$ *in* N *of the load for each peer* p_i.

Definition 13. *The* maximum load for a network N *is the load at the most heavily loaded peer* p *in* N.

The search latency for a peer P was calculated using two steps. First, we determined the shortest path s_i to each peer P_i where P_i had a search path to P. Then, we calculated search latency as the length of the longest s_i. The search latency for the network was the average search latency of each peer.

Recall that the fault susceptibility of the network is the maximum change in coverage that could be caused by removing one peer from the network. This was measured in

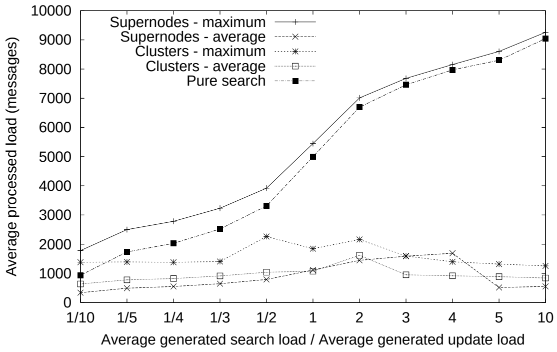


Fig. 7. Load of different topologies.

a straightforward way, by removing each peer, measuring the change in coverage, and then restoring the peer and its connections to the network before removing the next peer.

5.2 Load experiments

First, we examined the load characteristics of supernodes and parallel clusters versus each other and a baseline of a pure search network. To do this, we varied the ratio between the average SL and UL from $SL = UL \times 10$ to $SL = UL/10$. For each point in this interval, we chose network parameters (such as the number of supernodes or clusters) to tune the networks for efficiency. This process is discussed in Section 5.6. We also constructed a Gnutella-like pure search network as a baseline comparison.

The results are shown in Figure 7. This figure shows both average and maximum load for the supernode and parallel cluster topologies, as well as the load for the pure search network (where the average and maximum are the same.) On the extreme left of Figure 7, searches are rare (for each search issued 10 index updates are generated), while the extreme right represents a mostly search scenario. In this result, the 95 percent confidence interval of the maximum load in a parallel cluster topology is as large as $\pm\ 10$ percent (in the case of $SL = UL/10$). First, note that the maximum load in the supernode network is close to, though higher than, the load in the pure search network. This is because supernodes must handle both searches and updates, while pure search network peers only handle searches. Note also that the average load for supernode networks is significantly lower than that for pure search networks, which is to be expected, since most of the nodes in a supernode network are only handling their own local searches and updates.

A striking result in Figure 7 is that both the maximum and average load of parallel cluster networks are relatively low, roughly comparable to the average load in a supernode network. In fact, the average load in a parallel cluster network is always (in these experiments) within a factor of two of the average load in a supernode network, and the maximum load in a cluster network is only as much as a factor of four larger than the average load in a supernode network. Moreover, sometimes the average and

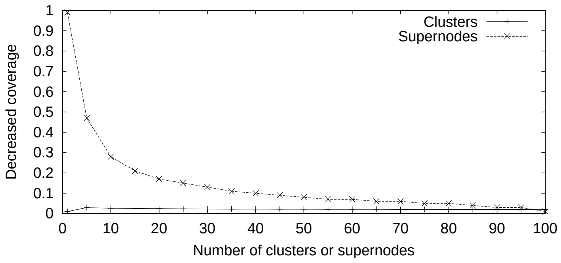


Fig. 8. Fault susceptibility.

maximum load in a cluster network is less than the average load in a supernode network. This indicates that nodes in a parallel cluster network are effectively sharing the load, contributing their resources to reduce the overall load on all nodes in the network. In contrast, in a supernode network, some nodes are lightly loaded while the supernodes are heavily loaded, and the most heavily loaded supernode can be up to seven times more loaded than the most heavily loaded node in a parallel cluster network. Note that our simulation results are consistent with our analytical comparison from Section 4.2.

We can draw the following conclusions from these results:

- A parallel cluster network ensures that no nodes are overloaded (e.g., more than twice as loaded as the average node), while only increasing the average load on nodes by up to a factor of two over a supernode network. This is beneficial:
  - Under the assumptions of our simulation (e.g., all nodes have roughly equal capability), and
  - When a primary goal is to reduce both the maximum and average load on nodes in the system.
- A supernode network ensures that most nodes in the system are lightly loaded, at the cost of placing heavy load on supernodes. This is beneficial
  - When some nodes have higher capacities than others, so that these high capacity nodes can serve as supernodes, and
  - When a primary goal is to reduce the average load on nodes in the system.

5.3 Search latency and fault susceptibility experiments

The next set of experiments we ran was to measure the search latency and fault susceptibility of the supernode and parallel cluster topologies. These metrics do not depend on the search or update rate; instead, the connectivity and topology of the network determines how quickly searches will be returned, and how vulnerable the network is to node failures. In this section, we report search latency and fault susceptibility as the number of clusters or supernodes vary.

First, Figure 8 shows the results for fault susceptibility. The horizontal axis in this figure shows the number of clusters or supernodes (depending on the type of network), and the vertical axis shows fault susceptibility, measured as decreased coverage after

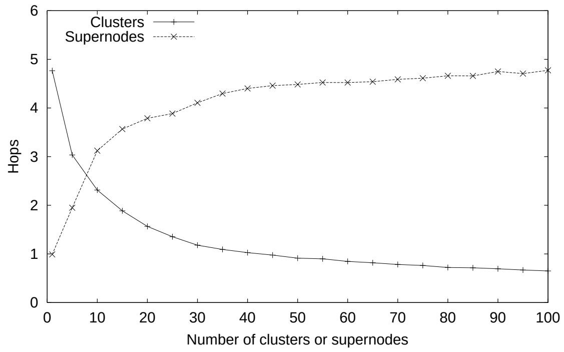


Fig. 9. Search latency.

a failure. For supernodes, the fault susceptibility drops as the number of supernodes increases. When there is only one supernode, the failure of that supernode is catastrophic, while multiple supernodes means that only some of the network nodes are disconnected if one supernode fails. In contrast, for clusters, fault susceptibility initially increases, and then slowly decreases. When there is one cluster, there are no indexing links, and failure only affects the failed node. When there is more than one cluster, a failure prevents other nodes in a cluster from searching the indexes stored by the failed node. However, as the number of clusters increase (and their size decreases), fewer nodes depend on any one node, and the fault susceptibility drops. Overall, cluster networks have a much lower fault susceptibility than supernode networks, since the failure of a cluster node causes a slight decrease in coverage, while the failure of a supernode completely disconnects some nodes.

Next, Figure 9 shows the results for search latency. In a supernode network, the search latency is low when there is only one supernode, since all searches travel only one hop to that supernode. Increasing the number of supernodes means that multiple hops are necessary to reach all supernodes. In a cluster network, as the number of clusters increases, clusters become smaller, and search latency decreases. Figure 9 shows that neither topology archetype is clearly superior in terms of search latency, as the latency for both archetypes varies between one and five hops.

5.4 Discussion

Supernode networks are used as an alternative to pure search networks like Gnutella because they reduce the load on normal nodes and thus increase the scalability of the system. However, our results confirm our qualitative observation that the scalability of the system is limited by the capacity of supernodes. In a network with full coverage, the supernodes handle all of the searches, and therefore must have very high capacity. If we are interested in constructing a network that bounds the maximum load, parallel search clusters is a more attractive option. This is useful if there are not many super-high-capacity nodes that can acts as supernodes, because we can better utilize the aggregate resources of the system.

Even if a network is heterogeneous and some nodes are high capacity, we still may choose a parallel cluster network. Consider a situation where we want the system to be robust in the face of failures. We may specify that we want a failure to result in a small decrease in coverage, say, no more than five percent of the nodes become unreachable. This is barely feasible in a supernode network; as shown in Figure 8, we would need more than 80 percent of the nodes to be supernodes to achieve such a low fault susceptibility. Then, the network would resemble a pure search network and would lose the scalability advantages of the supernode network. The other alternative, having normal nodes connect to more than one supernode, results in index/index redundancy with an attendant increase in load on the supernodes. On the other hand, a parallel cluster network easily achieves low fault susceptibility without increasing load. As shown in Figure 8, even the most fault susceptible network of five clusters experiences no more than three percent unavailable nodes after a failure.

5.5 Generating network topologies

We constructed pure search networks using the Gnutella model, where we connect two nodes using a pair of FSLs going in opposite directions. Since other investigators have noted that Gnutella networks tend towards a power law distribution [10, 19], we have constructed pure search networks using the PLOD algorithm [16]. The PLOD algorithm takes two parameters, the average number of links per node and the maximum number of links per node, and generates a power-law network. The values we chose for these parameters (see Table 1) were determined experimentally to ensure full coverage, reduce fault-susceptibility, and reduce search latency. (For a discussion of this process, see [5].)

We built supernode networks by designating some nodes as supernodes; the number of supernodes *NS* is specified as an input parameter to the simulation. Each normal node was assigned to a supernode randomly using a generator that produced normally distributed random values (via the polar Box-Muller algorithm [3]). The number of normal nodes assigned to a given supernode is normally distributed with a mean of n/NS and the standard deviation of one quarter of the mean. Normal nodes were connected to their supernode with one outgoing FSL and one outgoing NIL. Supernodes were connected using a Gnutella-like network of FSLs similar to the pure search network.

We made parallel search cluster networks by assigning nodes to clusters of FSLs, and connecting separate clusters with NILs. The number of clusters *NC* is specified as an input parameter to the simulation. The number of nodes in a given cluster is normally distributed with a mean of n/NC and a standard deviation of one quarter of the mean. Note that this means that the number of nodes in a cluster is similar to the number of normal nodes assigned to a supernode. Within a cluster, we built a power law search network of FSLs, similar to the pure search network above. After we had constructed clusters, every node in the network was given one NIL to one randomly chosen member of each other cluster.

5.6 Tuning networks

The archetypes of Section 3 are really families of widely varying topologies (as mentioned earlier). For example, different supernode networks can differ in the number of supernodes, the distribution of normal nodes to supernodes, the topology of the pure search network between supernodes, and so on. Given this diversity, which supernode variant do we compare to which parallel clusters variant? Our approach to this dilemma is as follows. First, we select a scenario with a given number of nodes and a given query and update load. Then, we search for parameters that lead to "tuned" supernode networks, that is, networks that minimize load for this scenario while retaining full coverage. We similarly find "tuned" parallel cluster networks. Then, we compare tuned instances of supernode networks to tuned instances of parallel cluster networks. We iterate over different scenarios, comparing different supernode and parallel cluster instances in each case. The process of finding good networks is described in Section 6.

In summary, for supernodes, we used:

- 5 supernodes for the interval $SL = 10 \times UL$ to $SL = 5 \times UL$, and
- 20 supernodes for the interval $SL = 4 \times UL$ to $SL = UL/10$.

For parallel clusters, we used:

- 15 clusters for the interval $SL = 10 \times UL$ to $SL = 3 \times UL$, and
- 10 clusters for the interval $SL = 2 \times UL$ to $SL = UL/2$, and
- 5 clusters for the interval $SL = UL/3$ to $SL = UL/10$.

6 Simulating "good" networks

In order to compare network archetypes, we must first find good values for the parameters that describe a network. For example, the load, fault susceptibility and search latency of a supernode network varies depending on the number of supernodes. In this section we describe the process of tuning the parameters of pure-search, supernode, and parallel search cluster networks. The "tuned" networks that result are the ones used in the experiments described in Section 5.

6.1 Power-law pure search networks

First, we studied the construction of power law pure search networks. All three of the topology types have subnetworks that are pure search networks, and thus it is necessary to decide how to construct power law graphs before we conduct the rest of our experiments. The PLOD algorithm for constructing power-law graphs allowed us to specify two parameters: the maximum number of connections and the average number of connections. Our approach was to find suitable values for these parameters for a Gnutella-like pure search network. Then, we used these same parameters to construct the search networks within clusters and between supernodes. This allows us to perform an "apples-to-apples" comparison between the different topology types as much as possible by constraining the search network topology to have similar connectivity. Note that as long as there are no partitions in the search network, the average and maximum connectivity only impacts the search latency and fault susceptibility of the network. The load of a full coverage search network does not vary depending on how we construct the power law graph, since every node still handles every query.

We selected 10 as the maximum number of connections because it seems reasonable that a node should not have to know about more than ten percent of the peers in the network. In fact, it might be reasonable to set a small fixed number (say, 10) as the maximum number of neighbors that a peer must have, even if the network grew beyond 100. This would allow nodes to participate in a large search network without needing to maintain a large number of network connections.

We studied the effect of the second parameter, the average number of connections, by varying the average between 1 and 10. The results are shown in Figure 10. For the

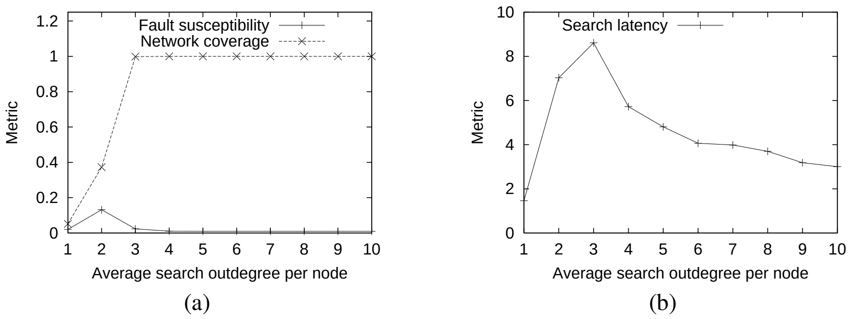


Fig. 10. Pure search networks: (a) coverage and fault susceptibility, and (b) search latency.

graphs in this figure, the average number of connections is on the horizontal axis. First, let us examine network coverage in Figure 10a. The network does not have full coverage (e.g., the network coverage is less than one) unless there are at least four connections on average per peer. (The data point for 3 connections is 0.998 coverage.) If there are less than four connections, there simply is not enough connectivity to ensure that there are no partitions. Next, consider the fault susceptibility metric (also Figure 10a), which represents the decrease in coverage when one node fails. If the average connectivity is at least 5, then fault susceptibility is only 0.01. In other words, when a node fails, that node is no longer searchable, but all other nodes are. These results indicate that it is reasonable to use 5 as the average connectivity, because it provides maximum coverage and minimum fault susceptibility.

Finally, the search latency decreases as connectivity increases (at least once full coverage is assured) as shown in Figure 10b. Intuitively, this makes sense; more connections means that there is more likelihood of a shorter path between nodes. Certainly, we could increase connectivity until search latency was one, and at this point every node would have an FSL directly to every other node. However, such a network would be hard to maintain, and as argued above, it is better if each node only has to know about a few neighbors. A pure search network could certainly achieve less latency by increasing connectivity without going to extreme of having every node connected to every other. Nonetheless, it is reasonable for our purposes to use 5 as the average connectivity, since search latency is not too bad (4.8 hops on average).

6.2 "Good" supernode and parallel search cluster networks

Next, we constructed supernode and parallel cluster networks so that we could measure the network load for each topology. We examine each topology individually, to determine the effect of the construction parameter (number of clusters or number of supernodes) on load.

Supernode networks We constructed supernode networks where the number of supernodes varied between 1 and 100. If there is one supernode, the network resembles a Napster network, where there is a centralized server that handles all of the indexing and searching. At the other extreme, 100 supernodes, the network is a pure search network, with no index links.

We examine the situation where the average search load SL was ten times the average update load UL, equal to the average UL, and one tenth of the average UL. The results for $UL = SL$ are shown in Figure 11a. This graph shows two metrics, the

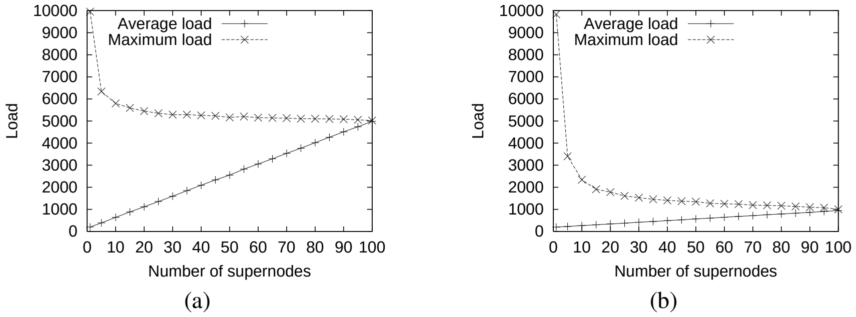


Fig. 11. Load versus number of supernodes: (a) $UL = SL$, (b) $UL >> SL$.

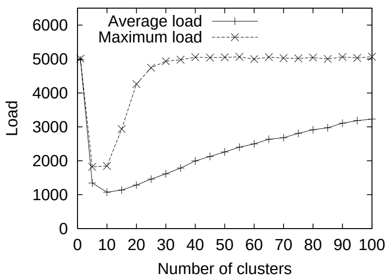
average load over all peers, and the maximum load on any one peer, versus the number of supernodes. As the figure shows, there can be a large discrepancy between the average and the maximum. Normal nodes only handle the load that they themselves generate. In contrast, a supernode handles all of the updates for its assigned normal nodes, as well as all of the search messages generated by any node in the network. The result is that supernodes can become heavily loaded. Note also that increasing the number of supernodes lightens the maximum load (e.g., the load on supernodes) but after about 20 supernodes adding another supernode does not significantly decrease the maximum load. This is because adding a supernode decreases the update load on other supernodes (by reassigning some normal nodes and their updates to the new supernode) but does not decrease search load at all. After 20 supernodes, the decrease in maximum update load from adding another peer is negligible.

For the situation where $UL >> SL$, adding a supernode is much more beneficial because the update load is the main load in the system and extra supernodes can share that update load. This effect is illustrated in Figure 11b. Nonetheless, even in this case adding more than 20 supernodes offers diminishing returns. Finally, in the case where $UL << SL$ (not shown), the benefit of adding supernodes beyond five is slight, and even going from one to five supernodes only decreases the maximum load slightly (from 10,022 load units to 9,260 load units).

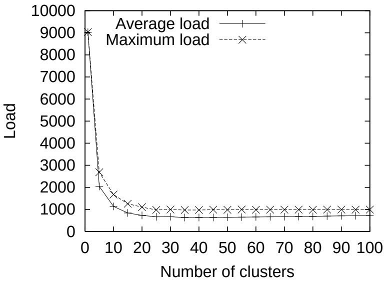
We can conclude that the load on supernodes can be very high, much higher than that on normal nodes in some cases. It does not make sense to have a large number of supernodes, since that unnecessarily increases the chance that a node will have to serve as a supernode and bear the high load. It is best to find the "knee" in the load versus supernode count curve, and use that in determining the number of supernodes. Our experiments (including those shown above and others not shown) indicate that if the average SL is five or more times the average UL, then the curve flattens at about 5 supernodes, and for SL less than five times the average UL the curve flattens at about 20 supernodes.

Parallel search cluster networks To study parallel search cluster topologies, we studied networks where the number of clusters varied between 1 and 100. The extreme of one cluster represents a pure search network, while the extreme of 100 clusters represents a pure indexing network. Again, we examined cases where UL was much larger than, equal to, and much smaller than SL.

The results for $UL = SL$ are shown in Figure 12a. This figure shows both the



(a)



(b)

Fig. 12. Load versus number of clusters: (a) $UL = SL$, (b) $UL << SL$.

maximum load on any node and the average load over all nodes. As the figure indicates, both the total and maximum load is high for one cluster, decreasing to a minimum around 5 or 10 clusters, and then increasing again. The shape of this curve is due to two competing effects: the increased update load on peers as the number of clusters increases, and the decreased search load on peers as clusters become smaller. When there is only one cluster, the search load dominates, since there are no updates but every peer handles every search. Creating even a few clusters dramatically reduces the search load, and the update load is not yet significantly expensive. Adding more clusters than 10 offers diminishing returns for decreasing search load, while the update load continues to increase.

We can also conclude from these results that when there are 5 or 10 clusters, most nodes share the load relatively evenly, since the gap between the average and maximum load is small. However, as the number of clusters increases, the gap increases as well. This is because with a larger number of clusters of normally distributed size, there is a greater chance that some clusters will have only a few nodes. Each node in these small clusters must take on a higher index update load, and these nodes carry the maximum load in the system. The extreme of this effect is observed when there are more than 40 clusters, where there is a high probability that there is a cluster of only one node, and that one node must handle updates from all of the other peers in the system.

For the case where $UL = 10 \times SL$ (not shown), the shape of the graph is roughly similar to Figure 12a, although the larger update rate means that there is a relatively larger gap between the average and maximum traffic. However, a different result is observed for $UL = SL/10$, as shown in Figure 12b. In this case, the minimum load is observed when there are 100 clusters; in other words, when the network is a pure index network. Because of the extremely high rate of searches, the search load always dominates and increasing the number of clusters decreases both the search load and the total load.

These results and experiments for other values of UL and SL suggest the following conclusions:

– For the case where $UL << SL$, increasing the number of clusters decreases total load, and a pure search network provides optimal load.
– For cases where UL is greater than, equal to or only somewhat smaller than SL, the minimum load can be attained with a small number (about 5-20) clusters.

7 Related work

Several researchers have examined special algorithms for performing efficient search in peer-to-peer search networks. For pure search networks, these techniques include parallel random walk searches [14, 1], flow control and topology adaptation [15], and iterative deepening search [23]. For networks with indexing, techniques include routing indices [7] and local indices [23]. Each of these approaches is useful for “fixing-up” an existing, inefficient network. Because these techniques can be used to improve the efficiency of the networks described by the SIL model once the networks are built, they are complementary to our own work. Moreover, while an operational model can describe techniques such as random walk searches, the the space of broadcast-based techniques in the architectural SIL model is rich enough to merit study. Some research has begun into constructing efficient networks a priori; see for example [24, 17].

Others have suggested that the content can be placed in the network to ensure efficiency [4, 15] or that the network topology be reorganized based on the location of the content [8]. Content-based techniques are useful if the content in the network can be appropriately analyzed. Our SIL model is content-agnostic, which is useful both in the architectural model phase, and in general when the network content is not easily analyzed. Moreover, proactive replication may require nodes to store content they are not interested in. Our techniques do not require proactive replication in order for search to be effective.

Some work has also focused on constructing P2P networks for end goals other than efficiency. FreeHaven [9] is a peer-to-peer search network that seeks to provide anonymity for content authors, while SOS constructs a P2P overlay to avoid denial-of-service attacks [13].

Moreover, a large amount of attention recently has been given to distributed hash tables (DHTs) such as CHORD [21] and CAN [18]. DHTs focus on efficient routing of queries for objects whose names are known, but often rely on a separate mechanism for information discovery (as pointed out in [18]). The emphasis on efficient routing of a location query, as opposed to efficient broadcast of a content-discovery query, means that a DHT necessarily has different requirements and topologies than the networks we study here. Moreover, there are still interesting questions in broadcast-based networks, and we feel that the full potential of such networks has yet to be realized. Finally, the huge popularity, wide deployment and clear usefulness of broadcast networks mean that optimizing such networks is an important research challenge. Although we could conceivably extend our model to describe DHTs, the very simplicity of the current model makes it powerful and useful for our needs.

Several researchers have proposed mechanisms for using peer-to-peer networks to answer structured queries. Examples include DHT-based SQL queries [11], the Local Relational Model [2], or distributed caching for OLAP queries [12]. It may be interesting to extend our model for more structured queries. However, there are many research issues in content-based queries, and we have focused on those as a starting point.

Other studies have performed measurements of deployed peer-to-peer systems. The nature of Gnutella and Napster peers and the connections between them were characterized in [20], and the nature of the Gnutella topology was studied in [19]. In addition, a large amount of work has been done to measure other network topologies, especially the Internet topology as a whole. For example, Tangmunarunkit et al examined the structure of the Internet and idealized topologies that accurately model this structure [22].

8 Conclusion

We have introduced a Search/Index Link model of P2P search networks that allows us to study networks that reduce the load on peers while retaining effective searching and other benefits of P2P architectures. With only four basic link types, our SIL model can represent a wide range of search and indexing structures. This simple yet powerful model also allows us to generate new and interesting variations. In particular, in addition to the supernode and pure search topologies, our SIL model describes topologies such as *parallel search clusters* and *parallel index clusters*. Analytical results, as well as experimental results from our topology simulator, indicate that a parallel search cluster network reduces overloading by allowing peers to fairly share the burden of answering queries, rather than placing the burden entirely on supernodes. This topology makes better use of the aggregate resource of the system, and is useful in situations where placing an extremely high load on any one peer is infeasible. Moreover, our results show that other considerations, such as fault susceptibility, may also point to parallel search clusters as an attractive topology.

References

1. L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:46135–46143, 2001.
2. P.A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. Workshop on the Web and Databases (WebDB)*, 2002.
3. G.E.P. Box and M.E. Muller. A note on the generation of random normal deviates. *Annals Math. Stat.*, 29:610–611, 1958.
4. E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. SIGCOMM*, August 2002.
5. B. F. Cooper and H. Garcia-Molina. SIL: Modeling and measuring scalable peer-to-peer search networks (extended version). http://www-db.stanford.edu/˜cooperb/pubs/-searchnetsext.pdf, 2003.
6. B.F. Cooper and H. Garcia-Molina. Ad hoc, self-supervising peer-to-peer search networks. Technical Report, Stanford University Database Group, 2003.
7. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.
8. A. Crespo and H. Garcia-Molina. Semantic overlay networks, 2002. Technical Report.
9. R. Dingledine, M.J. Freedman, and D. Molnar. The FreeHaven Project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, July 2000.
10. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. SIGCOMM*, 1999.
11. 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 *Proc. 1st Int'l Workshop on Peer-to-Peer Computing (IPTPS)*, 2002.
12. P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K.L. Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. In *Proc. SIGMOD*, 2002.
13. A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *Proc. SIGCOMM*, Aug. 2002.
14. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ACM International Conference on Supercomputing (ICS'02)*, June 2002.
15. Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.
16. C. Palmer and J. Steffan. Generating network topologies that obey power laws. In *Proc. of GLOBECOM 2000*, Nov. 2000.
17. G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. IEEE Symposium on Foundations of Computer Science*, 2001.
18. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM*, Aug. 2001.

19. M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.
20. S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Conferencing and Networking*, Jan. 2002.
21. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, Aug. 2001.
22. H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proc. SIGCOMM*, Aug. 2002.
23. B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.
24. B. Yang and H. Garcia-Molina. Desigining a super-peer network. In *Proc. ICDE*, 2003.