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 [5], 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 [14, 11], where queries are to locate a specific document with a specific identifier (see Section 5 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.

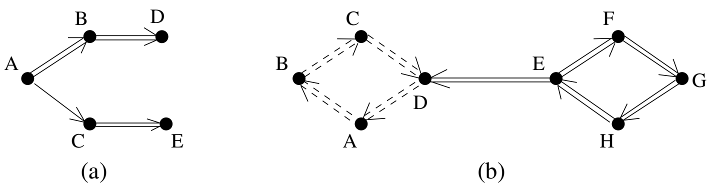

Fig. 1. Networks: (a) with search links, (b) with search and index links.

- 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 1a 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 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 1b shows a network that contains both search and index links. Index links are represented as dashed lines, single (- - ->) for non-forwarding index links and double (= = =>) for forwarding index links. (Note that Figure 1b 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 1b 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.
- *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

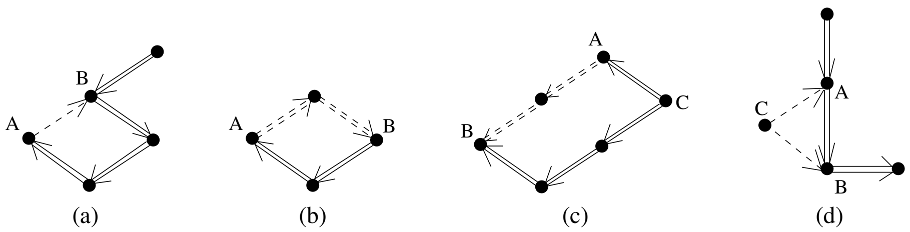

Fig. 2. Networks: a. with search/index redundancy, b. no search/index redundancy, c. search-fork, and d. index-fork

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 2a. 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 2b. 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 2c. Again, A will 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 2d.

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. The proof is straightforward: a redundant edge implies one of the named features. The full proof is available in the extended version of this paper [4]. □

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 [5, 16].

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 3a. 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

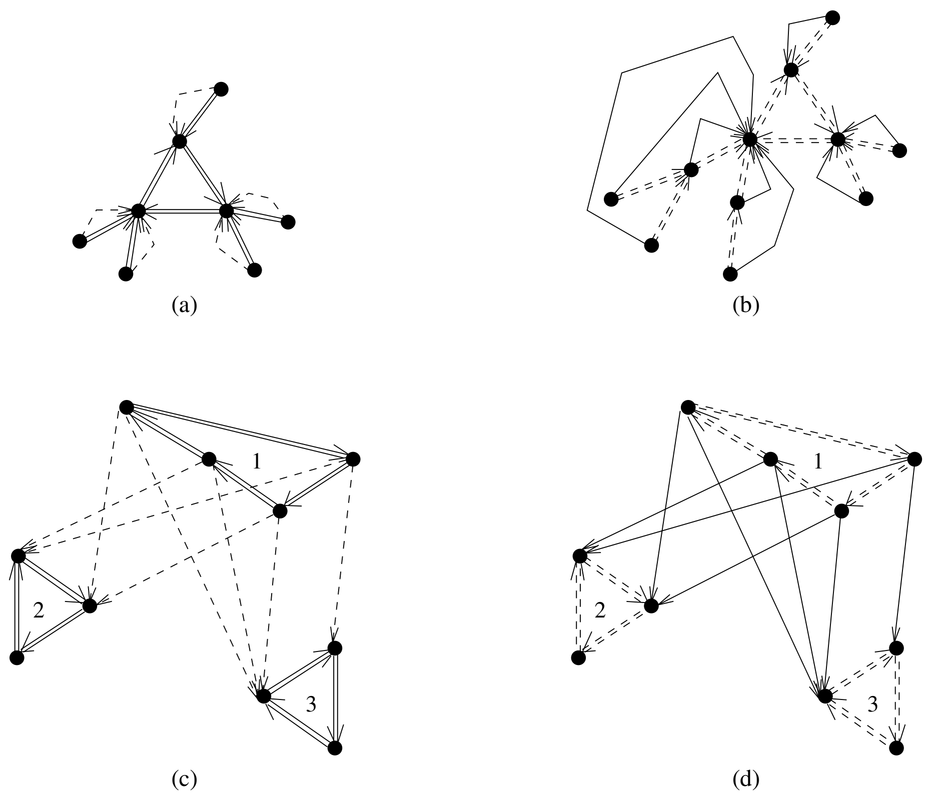

Fig. 3. Topology archetypes: a. Supernodes, b. Global index, c. Parallel search clusters, and d. Parallel index clusters. Some inter-cluster links are omitted in networks c and d for clarity.

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 3b. 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 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 3c. 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 [4], 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 3d. (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. We focus our evaluation on 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. Due to space limitations, we will only highlight here some of the results of our evaluation. For a complete discussion of our analytical and simulation results, see the extended version of this paper [4].

First, we must define metrics to evaluate both the efficiency and the robustness of networks. One measure of efficiency is *load*, which represents the amount of work that is done by each node in the network. In particular, we define load as the number of messages processed by nodes per unit time. There are two types of messages in the network: search messages and index update messages. We treat both types of messages as equally costly to process. A situation where one type of message is more costly to process can be represented in our framework as a scenario where there is more of one type of message than the other. Another measure of efficiency is *search latency*, or the response time for searches. We calculate this metric by examining the longest path length that queries must take in the network, because the path length is related to the amount of time a user must wait before the query returns all results. Finally, we also examine *fault susceptibility*, which represents the amount that service is degraded after a node fails. We calculate fault susceptibility by finding the node whose failure would cause the largest decrease in coverage, and measuring that decrease.

Next, we can obtain analytical results. Specifically, we examine the load of networks, both because this is most important to ensuring scalability, and also because it is particularly amenable to analytical evaluation. The load on a node X is the result of the search load and update load generated by nodes that send messages to X, and thus the network-wide load depends on the average number of search and update messages generated by nodes in the network. In [4], we define equations for the load on nodes as a function of the rate of search and update messages and the topology of the network. We can use those equations to compare different network archetypes. Imagine a network with 100 nodes, where each node generates 100 messages per unit time, divided between search and index messages. Figure 4 shows the load in cluster and supernode networks as a function of the ratio between the average number of generated search and update messages. As the graph shows, supernode networks have lightly loaded nodes on average but heavily loaded supernodes, especially as the search load in the network

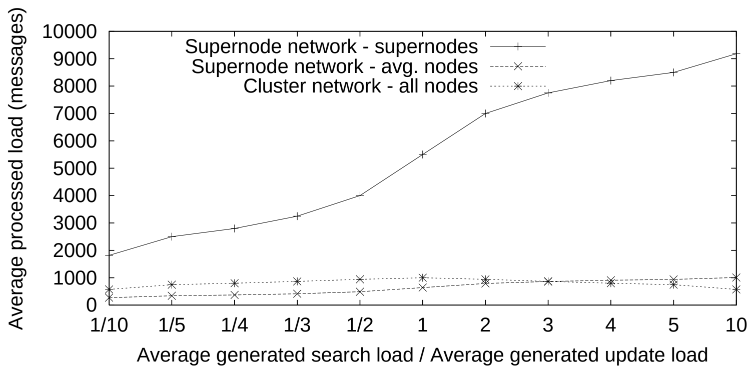

Fig. 4. Load of optimized networks.

increases. This is because supernodes do all of the search processing in the network. In contrast, nodes in a cluster network are much less loaded. In a cluster network, the search processing is shared among all of the nodes, and thus there are no nodes that are overloaded. Moreover, the cluster size can be adjusted based on the load in the network; for example, smaller clusters are better as the search load increases. The result is that a cluster network allows nodes to be lightly loaded, as if they were normal nodes in a supernode network, without needing to have any overloaded supernodes.

Our analytical result assumes idealized networks where all clusters are the same size and all supernodes are responsible for the same number of normal nodes. In contrast, real networks are likely to be more "messy" and we wanted to see if the load advantages of cluster networks applied to such messy networks. To do this, we used a simulator to generate several cluster networks, where each network had clusters of varying sizes. Similarly, we generated supernode networks, where the number of normal nodes varied from supernode to supernode. In each network, we chose the number of clusters or supernodes to minimize network-wide load. Table 1 lists the parameters we used to generate networks and calculate load.

Parameter	*Description*	*Base value*
n	Number of nodes	100
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

Our results indicate that the load in our simulated networks matches closely with the load predicted by Figure 4. As in our analytical results, our simulation results (not shown)

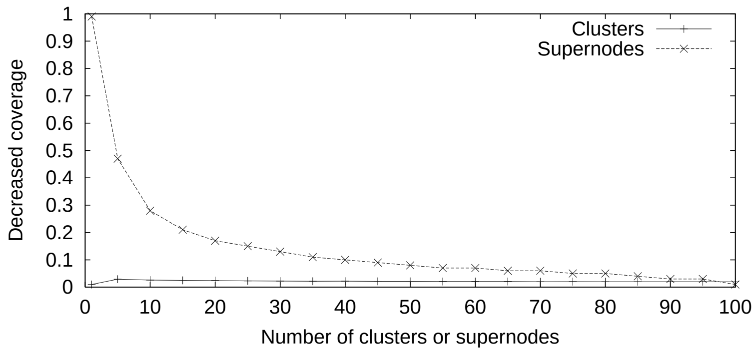

Fig. 5. Fault susceptibility.

indicate that the average cluster network node is far less loaded than supernodes, and roughly as heavily loaded as a normal node in a supernode network. The main difference between the simulation and analytical results is that in the simulation, some cluster nodes are up to a factor of two more loaded than the average cluster node. This is because when clusters are different sizes, nodes in large clusters must process more searches than nodes in small clusters. Nonetheless, a supernode is still far more loaded, by up to a factor of seven, than the most heavily loaded node in a cluster network. The problem of overloading is inherent to supernode networks; adding more supernodes does not decrease the search load on existing supernodes, and adding more normal nodes adds more processing capacity but that capacity goes unused. In contrast, a cluster network uses all the processing capacity and is thus inherently more efficient.

In addition to calculating the load in our simulated networks, we also calculated search latency and fault susceptibility. For a supernode network, search latency depends on the number of supernodes. A supernode network with one supernode has a search latency of one hop, since all searches travel only to the supernode. When there is more than one supernode, search latency is at least two hops, as a search must travel from a normal node to a supernode, and then across another hop to another supernode. Search latency may be more than two hops if the supernodes are not fully connected and searches have to travel multiple hops to reach all supernodes. In a cluster network, the search latency depends on the topology of the search network in each cluster. However, in our simulated cluster networks search latency was fairly low; when there was more than one cluster latency was three hops or less. Thus, we can conclude that the search latencies in supernode and cluster networks are comparable.

In contrast, the fault susceptibility of supernode and cluster networks were quite different. Recall that we are studying networks where $coverage = 1$; that is, the whole network is searchable by all nodes. Figure 5 shows the decrease in coverage as function of the number of clusters or supernodes. As the figure shows, a failure in a supernode network can cause a large decrease in coverage. This is because each supernode is responsible for a significant fraction of the network, and if a supernode fails its normal nodes are effectively partitioned until they can reconnect. In contrast, no node in a cluster network is as vital, because all nodes are sharing the search processing work. If one node fails then nodes in the same cluster as the failed node will experience a decrease in coverage, but the decrease will be small and nodes in other clusters will not notice any decrease. As a result, a cluster network offers much more resilience to node failures than a supernode network.

In summary, cluster networks ensure that no node is overloaded, without significantly increasing load on an average node. This load sharing is vital when there are no nodes in the network that are powerful enough to act as supernodes. Moreover, as a supernode network grows, even powerful supernodes can become overloaded, and thus for pure scalability reasons a cluster network may be preferred. At the same time, if robustness in the face of faults is important, a cluster network may also be preferred, since service in a cluster network will degrade much less after a failure than in a supernode network. Thus, cluster networks, suggested by our SIL model, are a useful alternative to supernode networks for peer-to-peer search.

5 Related work

Several researchers have examined special algorithms for performing efficient search in peer-to-peer search networks, including parallel random walk searches [8, 1], flow control and topology adaptation [9], iterative deepening search [15], routing indices [6] and local indices [15]. Others have suggested that the content can be proactively placed in the network to ensure efficiency [3, 9], or that the network be structured with low diameter [10]. Each of these approaches could be used to further optimize the general archetypes described by the SIL model.

Moreover, a large amount of attention recently has been given to distributed hash tables (DHTs) such as CHORD [14] and CAN [11]. DHTs focus on efficient routing of queries for objects whose names are known, but often rely on a separate mechanism for information discovery [11]. Information discovery is the focus of our work. Moreover, the huge popularity, wide deployment and clear usefulness of Gnutella/Kazaa-style networks mean that optimizing such networks is an important research challenge.

Some investigators have proposed mechanisms for using peer-to-peer networks to answer structured queries. Examples include DHT-based SQL queries [7] and the Local Relational Model [2]. 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.

Others have performed measurements of the peer-to-peer systems Gnutella and Napster [13, 12]. However, we know of no studies of deployed supernode networks, which are more widely used than were either Napster or Gnutella at their peak.

6 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. E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. SIGCOMM*, August 2002.
4. 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.
5. B.F. Cooper and H. Garcia-Molina. Ad hoc, self-supervising peer-to-peer search networks. Technical Report, Stanford University Database Group, 2003.
6. 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.
7. 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.
8. 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.
9. 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.
10. G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. IEEE Symposium on Foundations of Computer Science*, 2001.
11. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM*, Aug. 2001.
12. 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.
13. 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.
14. 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.
15. 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.
16. B. Yang and H. Garcia-Molina. Desigining a super-peer network. In *Proc. ICDE*, 2003.