Self-Organizing Parallel Search Clusters
Rocky Dunlap
rocky@cc.gatech.edu

Introduction and Motivation

A number of methods have been proposed to increase the efficiency and effectiveness of peer-to-peer searches. By controlling the topology of the peers, that is, the way in which nodes are organized in the network, we can offer some guarantees of high quality search performance. There are two schools of thought regarding how to control the topology of peer-to-peer networks. Using the structured approach, nodes are organized in the network in a strict, deterministic manner. When a new node enters the network, it is placed into a very specific location based on the structuring algorithm. For example, the Chord protocol (see [2]) places nodes in a highly defined ring structure. A new peer entering the network is assigned a specific location, and a number of update messages must be sent through the network to guarantee the integrity of the ring.

The other school of thought offers an unstructured approach to topology maintenance (see [3] for a listing of current implementations). In this approach peers come together in a less rigid manner. Although there are fewer guarantees as to the actual organization of the nodes, the unstructured approach carries the advantage of not having to maintain a significant amount of state information. Also, unstructured protocols do not require a centralized server to mediate the formation of the network. These networks are formed in a more ad-hoc manner in which the nodes decide among themselves how connections to peers will be made.

A simple model for describing peer-to-peer networks known as Search/Index Links (SIL) has been proposed in [1]. This model provides for two distinct types of links: search links and index links.

- A search link carries queries from one peer to another. Search links may further be divided into non-forwarding and forwarding search links. Non-forwarding search links (NSL) only carry a message one hop in the overlay, while forwarding search links (FSL) carry a query to a node, and then pass the query on to further nodes in the network.

- Index links carry updates from peer to peer. A non-forwarding index link (NIL) will propagate index updates one hop, while a node that receives a forwarding index link (FIL) will process the update, and then pass it on to neighboring nodes.

Figure 1. Parallel Search Clusters
Using the SIL model, a number of topologies are presented in [1]. Of particular interest is a network archetype known as Parallel Search Clusters. Nodes in this type of network are grouped into clusters where peers are strongly connected via forwarding search links. By *strongly connected* we mean that a query issued by one peer in the cluster will eventually reach every other peer in the cluster. Clusters are connected to each other via non-forwarding index links. Since all clusters are connected via index links, a search issued in one cluster will essentially reach the entire network (either by direct searching, or indirectly through another node’s index). Figure 1 depicts a peer-to-peer network in Parallel Search Cluster formation. Note that FSLs are shown using a solid line and NILs are shown using a single broken line. Some of the NILs have been left off for clarity.

Although the concept of Parallel Search Clusters sounds promising, it is not clear exactly how these clusters will form in an unstructured peer-to-peer network. Ideally, the clusters would form in a decentralized manner so that no single node has to maintain information about the entire cluster. In this paper we propose an algorithm by which nodes in a peer-to-peer network may self-organize into a parallel search cluster topology.

**Splitting Parallel Search Clusters**

We propose an algorithm for self-organizing parallel search clusters in which existing clusters divide into new clusters of fewer nodes. The intuition is that if nodes in a cluster are having to process too many queries, splitting the cluster into two segments will reduce the overall number of queries that each node has to process. Our goal is that the splitting algorithm functions in a decentralized manner such that nodes do not rely on a central server to coordinate the division of clusters. Additionally, our algorithm allows for “churn” in the network, that is, it takes into account that nodes will frequently be joining and leaving the peer to peer network. The splitting algorithm works in three phases as described below.

1. In phase one, nodes in a cluster are colored either green or red. At the end of splitting process, the green and red groups will each become new clusters.

2. During phase two, indexes are replicated between the green and red groups so that the new clusters will have the ability to search all content in the peer-to-peer network.

3. In the final phase, the two colored groups split into distinct search clusters. An important part of this phase in making sure that nodes in the each group are interconnected with search links. After splitting there will be no search links (only index links) between the two new clusters.

A primary question we must ask is how to determine when the appropriate time for a split has come? In order to answer this, we will assume that nodes in the peer-to-peer network are able to monitor their own query saturation level. We will also assume that if at least one of the nodes in a cluster is experiencing very high search demand, then all nodes in the cluster will benefit from a split and the saturated node should initiate the cluster splitting algorithm.

We address the problem of origin by noting that if a peer-to-peer network does not yet contain “clusters” in the strict sense, then the entire network may be considered one large cluster. After undergoing a series of split operations, the nodes in the network would then exist in a parallel search cluster topology.

**Phase One: Coloring**

The purpose of phase one is to determine the node membership of the new clusters. Ideally each of the new clusters will contain approximately half the number of nodes as the original cluster. One can image the negative impacts of splitting into clusters of uneven sizes. The bigger of the two clusters may need to split again soon since the overall search load would not be reduced significantly. At the same time, peers in the smaller cluster may have to handle an overly high volume of index updates since there are...
fewer nodes in the cluster to accept index links. Splitting clusters approximately in half will likely reduce the number of splits necessary as the peer-to-peer network grows. Note that we are currently assuming the each peer in the network contributes equally to the overall search volume in the cluster. In reality, some nodes may generate a higher number of queries than other nodes. Further research must be done to determine how the number of searches generated by a peer should affect cluster assignments.

Phase one is initiated when one node in the cluster is experiencing high search volume such that it can no longer efficiently or completely process its incoming queries. The node will then send out a color \((r, h)\) message along all of its outgoing search links with coloring radius \(r\) and hop count \(h\). Peers use the coloring radius and hop count parameters to determine which color they will choose. Essentially, those nodes in the cluster within radius \(r\) will choose to be green and those nodes outside of radius \(r\) will choose to be red. The phase one algorithm works as follows:

1. A query-saturated peer \(P\) sends out a color message along all of its outgoing search links. The color message has radius \(r\) and hop count zero.
2. If a node receives a color message and has already chosen a color, it keeps its current color and does not forward the message. If the node has not yet chosen a color, it adds one to the hop count \((h)\) and compares \(h\) to \(r\).
3. If \(h\) is less than or equal to \(r\), then the node chooses to be green, otherwise the node chooses to be red.
4. After deciding on a color, the node forwards the color message along all of its outgoing search links.

Since cycles will likely exist in the network overlay, nodes may receive the color message more than once. In this case, the node simply ignores the message and keeps its current color. This will prevent the message from looping infinitely in the cluster.

When the coloring message is broadcast, it will contain a unique identifier generated by the coloring node known as the **split id**. This identifier will be used in later phases to differentiate among split operations.

**Choosing the Radius**

One difficult problem is selecting an appropriate value for the coloring radius. We have already stated that we would like to divide the cluster into two segments of approximately the same size. First, it would be beneficial to know the **cluster size**, that is, the number of peers currently in the cluster. One way of doing this is to broadcast a ping message into the cluster and simply count of the number of responses. Assuming the cluster is adequately connected, this would produce an accurate count of peers in the cluster (not taking into account any nodes that may join or leave after the ping has been broadcast through the cluster). A peer may also estimate its cluster’s size by monitoring queries on incoming search links. The peer may simply keep a distinct list of nodes that have sent queries through the cluster. The advantage of the latter method is that it does not require a message to be broadcast through the entire cluster. However, it may not produce results with the same accuracy as the ping method since the peer’s list of nodes may quickly become out of date as nodes drop out of the cluster. Also, since a peer does not know if it will become query-saturated and need to initiate a split, all nodes must maintain a node list.

Even if we are able determine a good approximation of the cluster size, this information alone is not enough to decide on a coloring radius. We also need to know something about the **cluster search density**, that is, the density of search links between peers in the cluster. Messages broadcast in a densely connected cluster will reach a large number of nodes quickly (i.e. with few hops) since each node has a relatively high number of outgoing search links. On the other hand, a message broadcast in a cluster with low density will require more hops to reach the same number of nodes. In general, we observe that coloring radius should be smaller as the cluster density increases.

How can we measure cluster search density? One method is to find the average number of outgoing search links maintained by each peer in the cluster. We refer to the number of outgoing search
links as the search degree of the node. Since no one peer has a map of the cluster overlay, we cannot efficiently generate such an average. One way around this problem is to approximate the average search degree. For example, a node could simply count its outgoing search links and use that as the average for the entire cluster. This would serve as a good approximation if the search links are evenly distributed throughout the cluster. A peer may also ask several neighbors for their search degrees and then calculate an average.

We now offer a formula for calculating the coloring radius \( r \) using the average search degree \( d \) and cluster size \( s \):

\[
r = \log_d \left( \frac{s}{2} \right)
\]

Since coloring radius must be an integer, we will round \( r \) to the nearest integer value. So, if there are 50 nodes in the cluster, and the average search degree is 3 then we will choose a coloring radius of 3 because \( 3^3 = 27 \) which is close to 25. If each node had a search degree of exactly three and none of the outgoing links within the first three hops created a cycle, then exactly 27 nodes would receive the color green and the other nodes would receive the color red. Of course, we are assuming ideal conditions for this example, but our inability to know the exact cluster topology requires such an approximation.

**Phase Two: Replicating Index Links**

In order to maintain high coverage in the peer-to-peer network, green nodes and red nodes must exchange indexes before splitting into distinct clusters. After the split, the cluster composed of green nodes will no longer be able to search the red nodes directly. Instead, nodes in the green cluster must maintain indexes of nodes in the red cluster. Red nodes may also have index links from other clusters that need to be replicated to green nodes and vice versa. During phase two, green nodes in the cluster will create index packages addressed to red nodes, while red nodes will create similar index packages addressed to green nodes. In line with our decentralized approach, we do not emphasize that a particular green node should establish an index link with a particular red node. Instead, we only require that when a green node sends out its index package, some red node picks it up. As long as any red node receives and installs the green index package (i.e. actually creates the needed index links) then all red nodes will be able to take advantage of that index.

A nodes index package includes an outgoing index link for its own data, as well as any other incoming index links. Therefore, every node will have at least one index link to add to its package (its own index). If a node also has index links from outside clusters, it should include those in the package so that nodes of the other color are able to search that index. Here are the steps that occur during this phase.

1. Nodes generate an index package containing:
   a. An outgoing index link for its own content
   b. A list its incoming index links

2. Index packages are always addressed to the opposite color, red to green and green to red.

3. Index packages are sent out on a random walk through the cluster.

4. If a node receives an index package addressed to the other color, it simply forwards the package along one of its outgoing search links. If the node already knows a neighbor of the other color, it will choose to forward the index package on to that neighbor. Otherwise it will send the package along a random search link.

5. If a node receives an index package addressed to its own color, then it will install the package by setting up the index links in the package. The package is not forwarded on to other nodes.

**Phase Two Initiation**
Without a centralized server, it is not obvious how to determine if all of the nodes have been colored. Despite this fact, nodes may begin phase two as soon as they receive the `color()` message. After choosing a color, a node will prepare its index package and also attach the coloring message to the index package. As the index package propagates throughout the cluster, it could potentially arrive at a node that has not yet received the coloring message. In this case, the node will use the coloring message attached to the index package to choose a color, and then determine how to handle the index package (by installing or forwarding the package).

**Border Nodes**

We are under the assumption that index update rates are low when compared to the number of search requests. For this reason, we force the first node of appropriate color that receives an index package to install that package. We assume that the node that receives the index package will be able to handle the index links. However, it may be the case that certain border nodes (i.e. nodes who have neighbors of the other color) may become overloaded with index links since they will be the first to receive the index packages. It should be noted that clusters with a low search density are more likely to suffer from overloaded border nodes since there may only be a few border nodes in the entire cluster.

One method of avoiding overloaded border nodes is to allow those nodes to move some of their incoming index links to neighboring nodes of the same color. We propose a `donate(i, n)` operation where \( i \) is the index link to donate and \( n \) is the current number of incoming index links maintained by the donating node. If a node receives a donate message it will decide whether or not to install the index based on the \( n \) parameter. If the receiving node already has \( n \) or more incoming index links it will choose not to accept the index and forward it on to a neighboring node of the same color. If the receiving node has less than \( n \) incoming index links, it will choose to accept the donated link. After a node accepts the link, it will send a message back to donating node indicating the link has been accepted and the donating node can safely drop the link.

**Phase Three: Splitting**

In the final phase, the original cluster is split into two distinct clusters. After the split, there will be no search links between the two clusters and all nodes in each of the new clusters will have index links pointing to nodes in the other new cluster. Two issues must be resolved before the colored cluster can execute a split. First, we need a mechanism by which each node in the original cluster knows that the time to split has arrived. Secondly, we need to guarantee that the nodes in each cluster are completely connected with search links.

**Initiating Phase Three**

After forwarding its index package, a node will ask all of its neighbors if they have chosen a color. When all neighbors have chosen a color, then the node may begin phase three. Search links with neighbors of the same color are maintained without any changes. If the node is a border node, it chooses one neighbor of the opposite color and establishes a special outgoing index link called a bridge and drops all other search links with nodes of the opposite color. The bridge link serves two purposes. First of all, it acts as a regular index link between the two new clusters. Secondly, it provides a method for border nodes to “remember” a node of the opposite color in case it needs to forward index packages on to nodes of the opposite color. When a bridge node is created, it will be annotated with the `split id` of the current split operation. We also require that the same identifier be attached to outgoing index packages. A border node receiving an index package addressed to the opposite color will then forward it along the bridge so it will reach a node of the correct color.

**Connecting the Potential Clusters using Border Nodes**

Self-Organizing Parallel Search Clusters

5
Because the coloring message is broadcast outward from one node, those nodes within the coloring radius (the green potential cluster) will already form a connected graph. The red potential cluster, on the other hand, will likely suffer from fragmentation. In other words, the red nodes will typically lie on the outer edges of the original cluster and will not necessarily form a connected graph. During phase three, the red fragments must be located and search links must be established so that the resulting red cluster is a connected graph. We argue that in order to connect the red fragments, we must simply establish search links among all the red border nodes. Recall that border nodes are those nodes with at least one neighbor of the opposite color. As the coloring message of phase one propagates outward, red fragments will begin to form at the nodes one hop beyond the coloring radius. At greater radii, the nodes will choose to be red and will continue to form a connected graph with its border node(s). Therefore, by connecting the border nodes we ensure that all red fragments are connected.

It is easy for a node to determine if it is a border node. When a node receives the \( \text{color}(r, h) \) message it will simply check to see if it is one hop away from the radius, and if so, record locally that it is a border node. When phase three begins, all red border nodes will send a message along outgoing search links connected to green neighbors. Green nodes that receive the message will forward the message along all outgoing search links. If another red border node receives the message, it will establish a search link with the node that originated the message. After all search links have been established, the red potential cluster will be connected.

In the final phase of the split, any node that has a search link with a neighbor of the opposite color will simply drop that search link, thus splitting the original cluster into two distinct clusters.

**Atomicity of a Split**

Although the split algorithm has three logical phases, it may be thought of as an essentially atomic operation in the following manner. When a node receives its initial coloring message it chooses a color and immediately begins preparing its index package. The index package will then be shipped off (including the attached coloring message) along outgoing search links on a random walk. At this point the neighboring nodes will likely have chosen their respective colors so that phase three can be executed. Since all three actions take place in quick succession without requiring significant input from other nodes, the entire split operation may be thought of as an atomic action. The advantage of this atomicity is that the cluster will not be in a transient or inconsistent state for long periods of time leaving most of the network’s resources for processing queries and index updates.

**Dealing with Churn**

In a typical peer-to-peer network, nodes are unreliable and may join or leave the network at any time and without prior notice. Therefore, it is essential that the splitting algorithm function properly with respect to the unreliable nodes. In this paper we will only deal with “graceful” exits. This means that before leaving the cluster a node will execute an exit procedure to ensure that it does not leave the cluster in an inconsistent state.

**Joining a Cluster**

A node that wishes to join a peer to peer network organized into parallel search clusters will first ask the Hostcatcher for the address of some nodes in the network. The new node will then contact a node and attempt to join that node’s cluster. If the cluster is stable (i.e. not currently in the process of splitting), then the new node will simply establish some search links with several nodes in the cluster by requesting the address of several neighbors. Additionally, the new node must establish index links to all of the other clusters in the network. To facilitate this process we rely on the fact that all existing nodes in the cluster will already have outgoing search links to all other clusters. Therefore, the new node can simply ask a neighboring node for the addresses of all of its outgoing index links, and the establish index links with the same addresses.
If the node wishing to join the cluster discovers that its neighbors are in the process of a split operation, then it will choose to be the same color as its neighbors. If it has neighbors of different colors, the node randomly selects a color and drops any search links to nodes of the other color. At this point the node can continue in the split operation as usual. As noted above, the entire split algorithm may be considered an atomic operation, at least at the node level. Therefore a node that is in the split process may simply wait for it to complete before allowing a new node to contact it.

Leaving a Cluster

When a node decides to leave a cluster we must guarantee that the following two properties hold. First of all, the cluster should remain connected after the departing node drops out of the overlay. For clusters with higher search densities this will not likely be a concern. However, we may require that a node leaving a loosely connected cluster contact neighboring nodes so that those nodes may establish search links with each other. In this way we guarantee that the cluster will remain connected.

An exiting node must also donate all of its incoming index links to other nodes in the cluster. In this case the node chooses an extremely high value of \( n \) so that other nodes in the cluster are forced to accept the donated index link.

During a split, we must also handle the special case when a border node decides to exit the cluster. Since border nodes maintain the bridge links between potential clusters, we must guarantee that another node accepts responsibility of the bridge so that a path exists between the two clusters. We will require that an exiting border node simply donate its bridge index to a neighbor of the same color. We also extend the donate operation so that donated index links will retain their original split ids. After donating the index, the neighboring node that accepts it essentially becomes a new border node for the cluster.

Summary

We have presented a three phase approach to self-organizing parallel search clusters. In the first phase, nodes are colored red or green indicating the membership of the two potential new clusters. In the second phase, index links are replicated between the red and green groups so that each new cluster can still access content in the other cluster. In the third phase the clusters physically split as search links are severed between nodes of different colors. At the node level our algorithm may be thought of as an atomic operation requiring minimal input from outside nodes thus reducing the number of messages that must propagate through the cluster and allowing the network to withstand “churn.” The split is executed in a decentralized manner so that no single point of failure would leave the network overlay in an inconsistent state.
Bibliography

