

Giorgos Saloustros HY558

Making Gnutella-like P2P Systems Scalable

Gia is a file sharing peer-to-peer system. Its design is focusing mainly in the increase of the total capacity of search requests that the system can serve in the unit of time, in addition to previous file sharing p2p systems:

- Napster was the first file sharing system that appeared in the 1999. File search requests were served from a central directory server, and the files exchange were taking place between the peers.
- Gnutella is a completely p2p file sharing system. Gnutella implements the file searching utility between the peers using flooding with a preconfigured hops number.
- KaZaa introduces the notion of super nodes. File searching is now taking place again with flooding but only between the super nodes.

In order for Gia to increase the number of the search requests that system can serve, its design follows the following characteristics:

1. A **dynamic topology adaptation** protocol that puts most nodes within short reach of high capacity nodes. The adaptation protocol ensures that the well-connected (*i.e.*, high-degree) nodes, which receive a large proportion of the queries, actually have the capacity to handle those queries.
2. Tokens to nodes based on available capacity. **One-hop replication** of pointers to content. All nodes maintain pointers to the content offered by their immediate neighbors. Since the topology adaptation algorithm ensures congruence between high capacity nodes and high degree nodes, the one-hop replication guarantees that high capacity nodes.
3. A **search protocol** based on biased random walks that direct queries towards high-capacity nodes, which are typically best able to answer the queries.
4. An **active flow control** scheme to avoid overloaded hot-spots. The flow control protocol explicitly acknowledges the existence of heterogeneity and adapts to it by assigning flow-control tokens to nodes based on available capacity.

Dynamic Topology Adaptation

Every Gia node uses a satisfaction metric called S . S values lay in the space $[0,1]$. When the node X receives a request for new connection from node Y it decides the acceptance or the rejection of the request based on the following algorithm:

```

Let  $C_i$  represent capacity of node  $i$ 
if  $num\_nbrs_X + 1 \leq max\_nbrs$  then {we have room}
    ACCEPT  $Y$ ; return

    {we need to drop a neighbor}
     $subset \leftarrow i \ \forall i \in nbrs_X$  such that  $C_i \leq C_Y$ 
if no such neighbors exist then
    REJECT  $Y$ ; return
    candidate  $Z \leftarrow$  highest-degree neighbor from  $subset$ 

if ( $C_Y > max(C_i \ \forall i \in nbrs_X)$ ) { $Y$  has higher capacity}
or ( $num\_nbrs_Z > num\_nbrs_Y + H$ ) { $Y$  has fewer nbrs}
then
    DROP  $Z$ ; ACCEPT  $Y$ 
else
    REJECT  $Y$ 

```

In the current study, the variable Capacity C_i models the network bandwidth of the node. Variable C_i could be expanded in order to include other parameters such as disk latency, CPU etc. The parameter max_nbrs represents the maximum number of neighbor peers. The node executes the the topology adaptation algorithm every $I = (TxK)^{-(1-S)}$. As we can see from the equation the more S reaches 1 the time gap between two consecutive runs of topology adaptation increases.

The satisfaction metric is calculated as shown in the following figure.

```

Let  $C_i$  represent capacity of node  $i$ 
if  $num\_nbrs_X < min\_nbrs$  then
    return 0.0
     $total \leftarrow 0.0$ 
for all  $N \in neighbors(X)$  do
     $total \leftarrow total + \frac{C_N}{num\_nbrs_N}$ 
 $S \leftarrow \frac{total}{C_X}$ 
if  $S > 1.0$  or  $num\_nbrs_X \geq max\_nbrs$  then
     $S \leftarrow 1.0$ 
    return  $S$ 

```

Flow Control

To avoid creating hot-spots or overloading any one node, Gia uses an active flow control scheme in which a sender is allowed to direct queries to a neighbor only if that neighbor has notified the

sender that it is willing to accept queries from the sender. This feature is very important for Gia because it uses biased random walks instead of flooding. That means that if a node becomes overloaded it will start to drop search requests. This is not a problem for systems that use flooding because there are many copies of the same search request.

To provide better flow control, each Gia client periodically assigns flow-control tokens to its neighbors. Each token represents a single query that the node is willing to accept. Thus, a node can send a query to a neighbor only if it has received a token from that neighbor, thus avoiding overloaded neighbors. In the aggregate, a node allocates tokens at the rate at which it can process queries. If it receives queries faster than it can forward them (either because it is overloaded or because it has not received enough tokens from its neighbors), then it starts to queue up the excess queries. If this queue gets too long, it tries to reduce the inflow of queries by lowering its token allocation rate. To provide an incentive for high-capacity nodes to advertise their true capacity, Gia clients assign tokens in proportion to the neighbors' capacities, rather than distributing them evenly between all neighbors. Thus, a node that advertises high capacity to handle incoming queries is in turn assigned more tokens for its own outgoing queries. We use a token assignment algorithm based on Start-time Fair Queuing (SFQ). Each neighbor is assigned a fair-queuing weight equal to its capacity. Neighbors that are not using any of their assigned tokens are marked as inactive and the left-over capacity is automatically redistributed proportionally between the remaining neighbors. As neighbors join and leave, the SFQ algorithm reconfigures its token allocation accordingly.⁷ Token assignment notifications can be sent to neighbors either as separate control message by piggy-backing on other messages.

One hop replication

To improve the efficiency of the search process, each Gia node actively maintains an index of the content of each of its neighbors. These indices are exchanged when neighbors establish connections to each other, and periodically updated with any incremental changes. Thus, when a node receives a query, it can respond not only with matches from its own content, but also provide matches from the content offered by all of its neighbors. When a neighbor is lost, either because it leaves the system, or due to topology adaptation, the index information for that neighbor gets flushed. This ensures that all index information remains mostly up-to-date and consistent throughout the lifetime of the node.

Search Protocol

Gia uses biased random walks. That means that rather forwarding a search query to a randomly selected neighbor, it forwards the query to the highest capacity neighbor from which it has been assigned flow tokens. The reason for this is that the highest capacity neighbor is more likely to have answers for the incoming query. Each query has a MAX RESPONSES parameter, the maximum number of matching answers that the query should search for. In addition to the TTL, query duration is bounded by MAX RESPONSES. Every time a node finds a matching response for a query, it decrements the MAX RESPONSES in the query. Once MAX RESPONSES hits zero, the query is discarded. Query responses are forwarded back to the originator along the

reverse-path associated with the query.