Can Heterogeneity Make Gnutella Scalable?

Qin Lv[1], Sylvia Ratnasamy[2,3], and Scott Shenker[3]

[1] Princeton University, NJ, USA
[2] University of California at Berkeley, CA, USA
[3] International Computer Science Institute, Berkeley, CA, USA

1 Introduction

Even though recent research has identified many different uses for peer-to-peer (P2P) architectures, file sharing remains the dominant (by far) P2P application on the Internet. Despite various legal problems, the number of users participating in these file-sharing systems, and number of files transferred, continues to grow at a remarkable pace. File-sharing applications are thus becoming an increasingly important feature of the Internet landscape and, as such, the scalability of these P2P systems is of paramount concern. While the peer-to-peer nature of data storage and data transfer in these systems is inherently scalable, the scalability of file location and query resolution is much more problematic.

The earliest P2P file-sharing systems (*e.g.*, Napster, Scour) relied on a *centralized* directory to locate files. While this was sufficient for the early days of P2P, it is clearly not a scalable architecture. These centralized-directory systems were followed by a number of fully *decentralized* systems such as Gnutella and Kazaa. These systems form an overlay network in which each P2P node "connects" to several other nodes. These P2P systems are *unstructured* in that the overlay topology is *ad hoc* and the placement of data is completely unrelated to the overlay topology. Searching on such networks essentially amounts to random search, in which various nodes are probed and asked if they have any files matching the query; one can't do better on such unstructured systems because there is no information about which nodes are likely to have the relevant files. P2P systems differ in how they construct the overlay topology and how they distribute queries. Gnutella, for example, floods all queries and uses a TTL to restrict the scope of the flood. The advantage of such unstructured systems is that they can easily accommodate a highly transient node population. The disadvantage is that it is hard to find the desired files without distributing queries widely.

It seemed clear, at least in the academic research community, that such random search methods were inherently unscalable. As a result, a number of research groups have proposed designs for what we call "highly structured" P2P systems [9, 13, 10, 15]. In these structured systems the overlay topology is tightly controlled and files (or pointers to files) are placed at precisely specified locations.[1] These highly structured systems provide a mapping between the file identifier and location, so that queries can

[1] The Freenet system is what we would call a "loosely structured" system; file placement is affected by routing hints that are based on the object's identifiers; these locations are not precisely specified and so not all searches succeed.

be efficiently routed to the node with the desired file (or, again, the pointer to the desired file). These systems thus offer a very scalable solution for "exact-match" queries.[2]

However, structured designs are likely to be less resilient in the face of a very transient user population, precisely because it is hard to maintain the structure (such as neighbor lists, etc.) required for routing to function efficiently when nodes are joining and leaving at a high rate. Moreover, it has yet to be demonstrated that these structured systems, while well-suited for exact-match queries, can scalably implement a full range of partial query techniques, such as keyword searching, which are common on current file-sharing systems.[3]

Despite these issues, we firmly believe in the value of these structured P2P systems, and have actively participated in their design. However, the (sometimes unspoken) motivating assumption behind much of the work on structured P2P systems is that unstructured P2P systems such as Gnutella are inherently not scalable, and therefore should be abandoned. Our goal in this work is to revisit that motivating assumption and ask if Gnutella (and systems like it) could be made more scalable. We do so because, if scalability concerns were removed, these unstructured P2P systems might be the preferred choice for file-sharing and other applications with the following characteristics:

– keyword searching is the common operation,
– most content is typically replicated at a fair fraction of participating sites, and
– the node population is highly transient

The first condition is that one often doesn't know the exact file identifier, or is looking for a set of files all matching a given attribute (*e.g.*, all by the same artist). The second is that one isn't typically searching for extremely *rare* files that are only stored at a few nodes. This would apply to the music-sharing that dominates today's file-sharing systems. The third condition seems to apply to currently popular P2P systems, although it may not apply to smaller community-based systems.

File-sharing is one (widely deployed) example that fits these criteria, distributed search engines [4] might well be another. Unstructured P2P systems may be a suitable choice for these applications because of their overall simplicity and their ability to tolerate transient user populations and comfortably support keyword searches.[4] This all depends, of course, on whether or not such systems can be made scalable, and that is the question we address in this short paper.

Our approach to improving the scalability of these systems is based on recent work which shows the prevalence of heterogeneity in deployed P2P systems and on work improving the efficiency of search in unstructured networks. After reviewing this relevant

[2] By an exact-match query, we mean that the complete identifier of the requested data object is known.

[3] There are standard ways to use the exact-match facility of hash tables as a substrate for keyword searches, substring searches, and more fuzzy matches as well [14]. However, it isn't clear how scalable these techniques will be in *distributed* hash tables.

[4] We should add that unstructured P2P systems, whether scalable or not, will never be able match the exact-matching performance of highly structured P2P systems. We expect that these highly structured P2P systems will eventually find many uses; we just aren't convinced that popular-music file-sharing will be one of them.

background in Section 2, we describe, in Section 3, a simple flow control algorithm that takes advantage of heterogeneity and evaluate its performance in Section 4. We offer this algorithm not as a polished design but as a proof-of-concept that heterogeneity can be used to improve the scalability of unstructured P2P systems. Similarly, one should take this position paper not as a “proof” that Gnutella can be made scalable, but as a conjecture in search of more evidence.

2 Background

Our design is based on two ongoing research thrusts: attempts to improve the search facilities in Gnutella [8, 3, 1], and a measurement study [11] that revealed significant heterogeneity in P2P systems.

Improvements to Gnutella: Gnutella uses TTL-limited flooding to distribute its queries. Lv *et al.* [8] investigate several alternative query distribution methods on a range of overlay topologies: random graphs, power-law random graphs,[5] grids and a measured Gnutella topology. Based on their results, they propose the use of multiple parallel random walks instead of flooding. To search for an object using a random walk, a node chooses a neighbor at random and send the query to it. Each neighbor in turn repeats this process until the object is found. Random walks avoid the problem of the exponentially increasing number of query messages that arise in flooding.

Cohen *et al.* [3] study proactive replication algorithms.[6] They find that the optimal replication scheme for random search is to replicate objects proportional to the square-root of their query rate; this replication scheme can be implemented in a distributed fashion by replicating objects a number of times proportional to the length of the search.

These combined approaches, proactive replication plus parallel random walking, was tested via simulation in [8] and was found to significantly improve – by two orders of magnitude in some cases – the performance of flooding and passive replication, as measured by the query resolution time (in terms of number of overlay hops), the per-node query load and the message traffic generated by individual queries. In the case of power-law random graphs it was observed that high degree nodes experienced correspondingly high query loads. For this reason, the authors conclude that P2P network construction algorithms should avoid generating very high degree nodes.

Adamic *et al.*[1] approach the problem from a very different viewpoint. They take power-law random graphs as a given and instead ask how to best search on them. They, too, suggest the use of random walks, but that these random walks should be biased to seek out high-degree nodes. They show how this leads to superior scaling behavior in the limit of large systems. However, their work does not take into account the query load on individual nodes; as shown by [8] the high-degree nodes carry an extremely large share of the query traffic and are likely to be overloaded.

[5] Power-law random graphs are graphs where the degree distribution is a power-law and the nodes are connected randomly consistent with the degree distribution. A node’s “degree” is its number of neighbors.

[6] With proactive replication, an object may be replicated at a node even though that node has not requested the object. Passive replication, where nodes hold copies only if they’ve requested the object, is the standard approach in Gnutella-like P2P systems.

Heterogeneity: Saroiu *et al.*, in [11], report on the results of a measurement study of Napster and Gnutella. Their study reveals (amongst other things) a significant amount of heterogeneity in bandwidth across peers. The authors of [11] argue that architectures for P2P systems should take into account the characteristics of participating hosts, including their heterogeneity. However, since the measurement study was the main focus of [11], they did not propose any design strategies.

These two lines of work, the observed heterogeneity and the improvements to Gnutella-like P2P systems, lead us to the following observation. Using random-walk searches on power-law random graphs is an efficient search technique, but results in a very skewed usage pattern with some nodes carrying too large a share of the load. Measurement studies reveal, however, that node capabilities (particularly in terms of bandwidth, but presumably in terms of other resources such as CPU, memory and disk) are also skewed. If we could align these two skews – that is, have the highest capacity nodes carry the heaviest burdens – we might end up with a scalable P2P design.

This is what we attempt to do in the next section.

3 Design

Our design for an unstructured P2P system uses a distributed flow control and topology construction algorithm that (1) restricts the flow of queries into each node so they don't become overloaded and (2) dynamically evolves the overlay topology so that queries flow towards the nodes that have sufficient capacity to handle them.[7] Over this capacity-driven topology we use a biased random walk quite similar to that employed in [1]. The combination of these algorithms results in a system where queries are directed to high capacity nodes, and thus are more likely to find the desired files. Our design results in a quasi-hierarchical organization of nodes with high-capacity nodes at the higher levels in the hierarchy. This "hierarchy" is not, however, explicit; instead it is achieved through a distributed, adaptive and lightweight self-organization algorithm.

Flow control and Topology Adaption: In what follows, we represent the capacity of node i by c_i. For the purpose of this paper, we assume that c_i denotes the maximum number of messages node i is willing/able to process over a given time interval T. A node i is connected, at the application-level, to a set of *neighbor* nodes, denoted $nbr(i)$. For each of $j \in nbr(i)$ a node i maintains the following information:

– $in[j,i]$: the number of incoming messages from node j to i in the last time interval T. Every node i reports its total incoming rate ($in[*,i] = \sum_{j \in nbr(i)} in[j,i]$) to all its neighbors.
– $out[i,j]$: the number of outgoing messages from node i to j in the last time interval T

[7] We still consider this an *unstructured* design, even though the topology has become more structured, because the topology adjustment, while it improves scalability, is not required for correctness of operation.

– $outMax[i,j]$: the maximum number of messages node i can send to node j per time interval T. At all times, $out[i,j] <= outMax[i,j]$ and $outMax[i,j] <= out[i,j] + (c_j - in[*,j])$.[8]

When a new node, say i, joins the network, it is connected to a random set of neighbor nodes. For each neighbor node j, i initializes:

– $outMax[i,j] = c_i/d_i$
– $in[j,i] = 0$ and
– $out[i,j] = 0$

Every node i counts the number of messages it receives from and transmits to each neighbor node j ($in[j,i]$ and $out[i,j]$ respectively). Periodically, node i checks whether it is overloaded; *i.e.* whether its total incoming query rate exceeds its capacity. If overloaded, node i attempts to adapt the topology so as to reduce its incoming query load – node i selects a neighbor (say p) with a high incoming query rate (*i.e.* high $in[p,i]$) and redirects it to another neighbor (say q) with high spare capacity. Intuitively, the above topology adaptation rule sets up a direct "link" between a node with high query rate (node p) and one with high capacity (node q) and gets the overloaded node (node i) out of the way. If the overloaded node i cannot find an appropriate neighbor q (as would be the case when all of i's neighbors are running close to capacity), it requests node p to throttle the number of queries it is forwarding to node i (*i.e.* reduce outMax[p,i]). The detailed pseudocode for the above high-level description is shown in Figure 1.

Note that all flow control/topology adjustment decisions made by a node are based on local information (i.e. about the node itself and its neighbors). Further, if a node's capacity was to undergo a sudden change or if the node leaves the system, the topology would automatically re-adapt to accommodate the change.

Our simulations in Section 4 use the algorithm from Figure 1 with parameter values of $\delta = 10$, $\gamma = 1.25$, and $T = 100s$.[9] We defer an exploration of the parameter space to later work. While our initial simulation results appear promising, we expect to incorporate extensions and modifications to the above rules as we continue to study the behavior of our algorithm. For example, a useful measure to factor into the topology adjustment process would be the amount of time a node is expected to remain in the system. We also plan to add a *proactive* component to the above algorithm whereby a node whose capacity is not being utilized can explicitly discover and connect to nodes that are experiencing high query loads.

Search: We use a random-walk based search that combines features from the algorithms in [8, 1]. As in [8] we use TTLs to terminate walks and "state-keeping" to accelerate walks.[10] Similar to [1], a node forwards a query to its neighbor with the

[8] $c_j - in[*,j]$ represents node j's current spare capacity. With the above bounds on $out[i,j]$ and $outMax[i,j]$, a node i can at most increase its current outgoing rate to j by an amount equal to node j's current spare capacity.

[9] In practice, δ and γ need not be fixed values. A node can quite easily communicate more precise increase and decrease parameters to its neighboring nodes.

[10] With state-keeping, a search is given a unique identifier. A node then remembers the neighbors to which it has already forwarded queries for a given identifier, and if a query with the same

```
_checkLoad(node $i$)
  // periodic flow control and topology adaptation at $i$
  If ($\sum_{j \in nbr(i)} in[j,i] > c_i$) // $i$ overloaded?
    pick $p \in nbr(i)$ with max $in[j,i]$ over all $j \in nbr(i)$
    search for node $q (\neq p) \in nbr(i)$ such that:
      ($outMax[i,q] - out[i,q] \geq in[p,i]$) && ($p \notin nbr(q)$)

  If ($q$ exists) // adapt topology
    redirect $p$ to $q$; // i.e. $p$ disconnects from
                         // $i$ and connects to $q$
    $outMax[i,q] = outMax[i,q] - in[p,i]$;
    $outMax[p,q] = in[p,i]$;
    $outMax[q,p] = outMax[i,p]$;

  else // no satisfactory $q$ exists
    node $i$ send $p$ a slowdown message
  forall $j \in nbr(i)$: $in[j,i] = 0$, $out[i,j] = 0$
  forall $j \in nbr(i)$: $outMax[i,j] = outMax[i,j] + \delta$

_recvSlowDown(node $j$, node $i$)
  // node $j$ receives a slowdown message from $i$
  $outMax[j,i] = outMax[j,i]/\gamma$
```

Fig. 1. *Pseudo-code for flow control and topology adaptation. Although not explicitly shown in the pseudocode, the upper bound on outMax is enforced in performing the increase operations on outMax.*

maximum value of $outMax$. This causes searches to quickly gravitate towards high capacity nodes. As in Adamic's work, the high degree nodes in our topology do face higher query load. However, unlike Adamic's work, high degree nodes in our system are also high capacity nodes and hence better equipped to handle this load.

In summary, a node i that receives a query, forwards it to its neighbor j with the highest value of $outMax[i,j]$ over all $j \in nbr(i)$ provided (1) $out[i,j] < outMax[i,j]$ and (2) i has not previously forwarded the same query to j.

While we do not explore proactive replication strategies in this paper, any of the solutions described in [8] could be used to further improve the performance of our search algorithm. In its stead, we assume that the number of copies of each object is proportional to the rate at which it is being queried (which would occur under passive replication). Moreover, we assume that these copies are randomly placed proportional to the capacity of nodes.

identifier arrives back at the node, it is forwarded to a different neighbor. Using state-keeping thus reduces the likelihood that a random walk will traverse the same path twice. State can be discarded after a short time by a node, so it does not become burdensome.

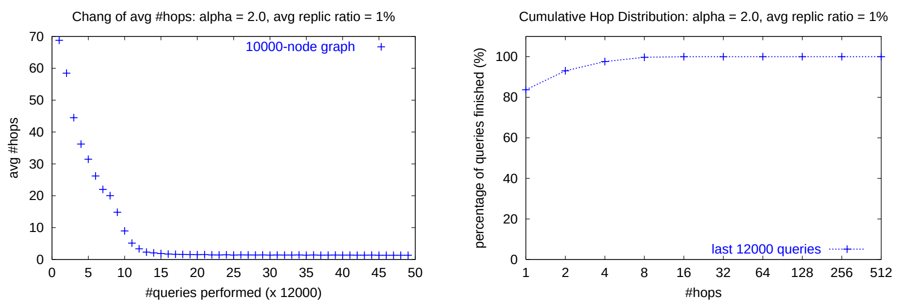

Fig. 2. *Average query resolution time and the distribution of query resolution times for a 10,000 node simulation using a Zipf-like capacity distribution*

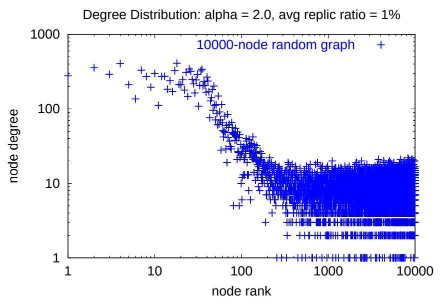

Fig. 3. *Degree distribution for a 10,000 node simulation using a Zipf-like capacity distribution*

4 Evaluation

In this section, we present initial simulation results on the performance of our design outlined in Section 3. We stress that these results are not intended to provide a comprehensive picture of the behavior of our algorithm but merely to serve as a first-order proof of concept.

Simulation Methodology: We use, as our main evaluation metric, the average query resolution time (as measured in terms of the number of application-level hops required to find a particular object). We measure this in simulations in which the object popularity, the rate at which queries are issued for it, follows a Zipf-like distribution where the popularity of the i'th most popular object is proportional to $i^{-\alpha}$. In these simulations we used $\alpha = 1.2$ (based on the Gnutella measurements in [12]). Individual nodes generate queries using a Poisson process with an average query rate of 1.2 queries/minute. Recall, as described above, that we use a proportional replication strategy so that objects are replicated proportional to their query rate. The average degree of replication is 1%[11] – that is, on average objects are replicated on 1% of the nodes, but the more

[11] We also experimented with a replication degree of 0.1% and obtained similar results

popular objects are replicated more than the less popular ones. Recall also that objects are assigned randomly to nodes, proportional to their capacity c_i.

To investigate the effect of heterogeneous node capabilities, we use two different capacity distributions. First, we use a Zipf-like distribution where c_i is proportional to $i^{-\beta}$ where $\beta = 2$. Second, we used a distribution based (loosely) on the measured bandwidth distributions of [11]. Here, we assume 5 capacity levels separated by an order of magnitude. The node population is then divided amongst these levels as shown in Table 1. The distribution reflects the observation that a fair fraction of Gnutella clients have dial-up connections to the Internet, the majority are connected via cable-modem or DSL and a small number of participants, via high speed lines.

Table 1. *Gnutella-like node capacity distributions*

Capacity level	Percentage of nodes
x	20%
10x	45%
100x	30%
1000x	4.9%
10000x	0.1%

For each simulation, we start with nodes connected in a uniform random graph topology; for the Zipf-like capacity distribution we use a 10,000 node graph with average degree 9.9, for the Gnutella-like distribution capacity distribution we use a a 5,000 node graph with average degree 7.5. We then assign the capacities and objects to the nodes as described above.

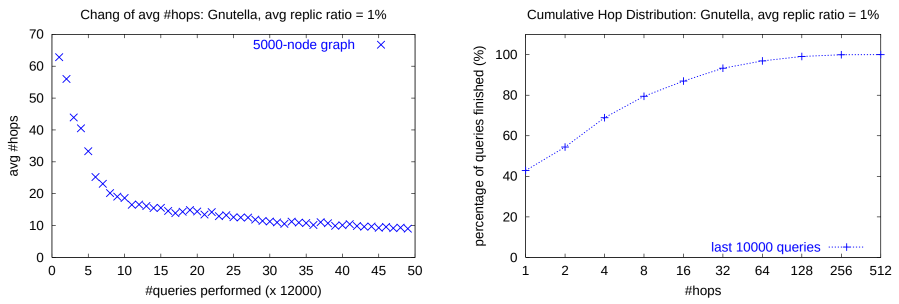

Fig. 4. *Average query resolution time and the distribution of query resolution times for a 5,000 node simulation using a Gnutella-like capacity distribution*

Results: Figures 3 and 5 plot the results of simulations using, respectively, the Zipf and Gnutella-like capacity distribution. The first plots show how, as the topology adjusts,

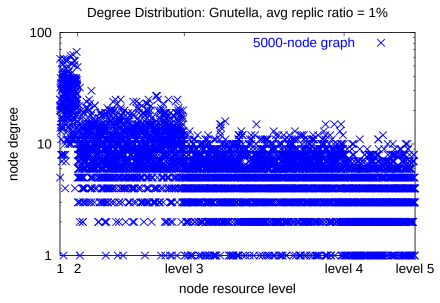

Fig. 5. *Degree distribution for a 5,000 node simulation using a Gnutella-like capacity distribution*

the average query response time rapidly decreases from 70 to ~ 2 in the Zipf case and 9 in the Gnutella case. The middle plots show the distribution of the number of hops for individual queries towards the end of the simulation run. In both cases all queries were successfully resolved. The last plots show the degree distributions of the resulting topology at the end of the simulation run. As one can see, the degree distributions evolve to match the capacity distributions – higher capacity nodes have significantly higher degrees – causing the improved performance.

5 Other Related Work

The Kazaa [5] network appears similar to our work in many respects. Kazaa is a self-organizing, multi-layered network where powerful hosts act as search hubs (called SuperNodes). Details of the Kazaa protocols and code are not publicly available making it hard to draw comparisons to our work.

In [7] the authors propose a cluster-based architecture for P2P systems (CAP). CAP organizes nodes into a two-level hierarchy using a centralized clustering server [6]. Each cluster has a delegate node that acts as directory server for objects stored by nodes within the cluster. Delegate nodes perform intra-cluster node membership registration while a central server tracks existing clusters and their delegates. To some extent, our algorithms achieve the same effect as CAP, with high capacity nodes behaving like delegate nodes. There are however significant differences: unlike CAP our algorithms are completely decentralized and we do not build an *explicit* hierarchy.

File sharing in FreeNet [2] uses hints about the placement of files to improve search scalability. FreeNet does not address node heterogeneity.

6 Discussion

We started this paper with the basic question of whether one could make unstructured P2P systems scalable. Building on the work in [8, 3, 1, 11] we proposed a design that appears to achieve significant scalability. This design is extremely preliminary, and and our evaluation leaves many questions unanswered. We offer it however, merely as support for the conjecture that unstructured P2P systems can be significantly improved, perhaps to the point where their scalability is no longer a barrier.

Our design also raised the more philosophical question of how to deal with heterogeneity. Most of the highly structured P2P designs start with the assumption of a homogeneous node population and then alter their designs to *accommodate* heterogeneity. The design we present here actively *exploits* heterogeneity. One can ask whether the highly structured designs could also be modified to exploit heterogeneity.

References

[1] ADAMIC, L., HUBERMAN, B., LUKOSE, R., AND PUNIYANI, A. Search in power law networks. *Physical Review. E 64* (2001), 46135–46143.

[2] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. Freenet: A Distributed Anonymous Information Storage and Retrieval System. ICSI Workshop on Design Issues in Anonymity and Unobservability, July 2000.

[3] COHEN, E., AND SHENKER, S. Optimal replication in random search networks. preprint, 2001.

[4] INFRASEARCH. http://www.infrasearch.com.

[5] KAZAA. http://www.kazaa.com.

[6] KRISHNAMURTHY, B., AND WANG, J. On network-aware clustering of web clients. In *Proceedings of SIGCOMM '00* (Stockholm, Sweden, Aug. 2000).

[7] KRISHNAMURTHY, B., WANG, J., AND XIE, Y. Early measurements of a cluster-based architecture for p2p systems. In *ACM SIGCOMM Internet Measurement Workshop* (San Francisco, Nov. 2001).

[8] LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. Search and replication in unstructured peer-to-peer networks. preprint, 2001.

[9] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A Scalable Content-Addressable Network. In *Proceedings of SIGCOMM 2001* (Aug. 2001).

[10] ROWSTRON, A., AND DRUSCHEL, P. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighteenth SOSP* (2001), ACM.

[11] SAROIU, S., GUMMADI, K., AND GRIBBLE, S. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Conferencing and Networking* (San Jose, Jan. 2002).

[12] SRIPANIDKULCHAI, K. The popularity of gnutella queries and its implications on scalability. In *O'Reilly's www.openp2p.com* (Feb. 2001).

[13] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001* (Aug. 2001).

[14] WITTEN, I. H., MOFFAT, A., AND BELL, T. C. *Managing Gigabytes: Compressing and Indexing Documents and Images*, second ed. Morgan Kaufmann, 1999.

[15] ZHAO, B., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. UCB Technical Report, 2001.