Search and Replication in Unstructured Peer-to-Peer Networks

Ubiquitous Computing, Autumn 2010-2011

Achin Agarwal 07CS1040
Anshul Gupta 07CS3020
Vighnesh Avadhani 07CS3032

November 10, 2010

Instructor: Prof. Niloy Ganguly

Department of Computer Science & Engineering

Indian Institute of Technology, Kharagpur

1 P2P Network Architectures

P2P network architectures can be classified into 3 kinds:-

1. **Centralized**: Napster and other similar systems have a constantly-updated directory hosted at central locations. Nodes in the P2P network issue queries to the central directory server to find which other nodes hold the desired files. Such centralized approaches do not scale well and have single points of failure.

2. **Decentralized but Structured**: These systems have no central directory server, but they have a significant amount of structure, i.e. the P2P overlay topology (i.e. the set of connections between P2P members) is tightly controlled and that files are placed not at random nodes but at specified locations that will make subsequent queries easier to satisfy.

3. **Decentralized and Unstructured**: These are systems in which there is neither a centralized directory nor any precise control over the network topology or file placement. Gnutella is an example of such designs. To find a file, a node queries its neighbors. The most typical query method is flooding, where the query is propagated to all neighbors within a certain radius. These unstructured designs are extremely resilient to nodes entering and leaving the system. However, the current search mechanisms are extremely unscalable, generating large loads on the network participants.

The discussion below is for Gnutella-like decentralized, unstructured P2P systems. The goal is to study more-scalable alternatives to existing Gnutella algorithms, focusing on the search and replication aspects.

2 Methodology

2.1 Abstractions

We assume that there are m objects of interest. Let q_i be the relative popularity, in terms of the number of queries issued for it, of the i'th object. The values are normalized, i.e $\sum_{i=1}^{m} q_i = 1$.

We assume each object i is replicated on r_i nodes, and the total number of objects stored in the network is R, i.e. $\sum_{i=1}^{m} r_i = R$.

We look at three aspects of a P2P system:-

1. **P2P network topology**: By network topology, we mean the graph formed by the P2P overlay network; each P2P member has a certain number of neighbors and the set of neighbor connections forms the P2P overlay network. Examples of network topologies are Power-Law Random Graph, Normal Random Graph, Gnutella graph, 2-dimensional grid etc.

2. **Query distribution**: By query distribution, we mean the distribution of query frequencies for individual files. We investigate the following distribution:

   - **Uniform Distribution**: All objects are equally popular, i.e. $q_i = 1/m$

- **Zipf-like Distribution**: Object popularity follows a Zipf-like distribution, i.e. $q_i \propto 1/i^\alpha$. Studies have shown that Napster, Gnutella and Web queries follow Zipf-like distributions.

3. **Replication**: By replication, we mean the number of nodes that have a particular file. We will use replication ratio to mean the percentage of nodes having the file.We consider several static replication distributions:-

   - **Uniform**: All objects are replicated at the same number of nodes, i.e. $r_i = R/m$.
   - **Proportional**: The replication of an object i is proportional to the query probability of the object, i.e. $r_i \propto q_i$. If only nodes requesting an object store the object, then the replication distribution is usually proportional to query distribution.
   - **Square-root**: The replication of an object i is proportional to the square root of its query probability q_i, i.e. $r_i \propto \sqrt{q_i}$.

There are only three relevant combinations of query distribution and replication distribution: uni- form/uniform, Zipf-like/proportional, and Zipf-like/square-root.

2.2 Metrics

We focus on efficiency aspects solely, and use the following simple metrics in our abstract P2P networks. These metrics, though simple, reflect the fundamental properties of the algorithms.

1. **User Aspects**:

   - **Pr(Success)**: The probability of finding the queried object before the search terminates. Different algorithms have different criteria for terminating a search, leading to different probabilities of success under various replication distributions.
   - **#hops**: Delay in finding an object as measured in number of hops. We do not model the actual network latency here, but rather just measure the abstract number of hops that a successful search message travels before it replies to the originator.

2. **Load Aspects**:

   - **#msgs per node**: Overhead of an algorithm as measured in average number of search messages each node in the P2P network has to process. The motivation for this metric is that in P2P systems, the most notable overhead tends to be the processing load that the network imposes on each participant. The load, usually interrupt processing or message processing, is directly proportional to the number of messages that the node has to process.
   - **#nodes visited**: The number of P2P network participants that a query's search messages travel through. This is an indirect measure of the impact that a query generates on the whole network.
   - **Percentage of message duplication**: Calculated as (total #msgs - #nodes visited)/total #msgs.
   - **peak #msgs**: To identify hot spots in the network, we calculate the number of messages that the busiest node has to process for a set of queries.

3 Analysis of Search Methodologies

3.1 Flooding

One of the major load issues in P2P networks is the load on individual network participants. Unfortunately, the flooding search algorithm used in Gnutella exacerbates this problem. Gnutella uses TTL(Time-To-Live) to control the number of hops that a query can be propagated. However, choosing the appropriate TTL is not easy. If the TTL is too high, the node unnecessarily burdens the network. If the TTL is too low, the node might not find the object even though a copy exists somewhere. Another problem with flooding is that, there are many duplicate messages introduced by flooding, particularly in high connectivity graphs. These limitations mean that flooding incurs considerable message processing overhead for each query, increasing the load on each node as the network expands and the query rate increases, to the point that a node can be so loaded that it has to leave the network.

3.2 Expanding Ring

We can use successive floods with increasing TTLs. A node starts a flood with small TTL - if the search is not successful, the node increases the TTL and starts another flood. The process repeats until the object is found. We expect this method to perform particularly well when hot objects are replicated more widely than cold objects, which is likely the case in practice. This method is called "expanding ring". Expanding ring reduces message overhead signifcantly compared with regular flooding with a fixed TTL. Though expanding ring solves the TTL selection problem, it does not address the message duplication issue inherent in flooding.

3.3 Random Walk

Random walk is a well-known technique, which forwards a query message to a randomly chosen neighbor at each step until the object is found. We call this message a "walker". The standard random walk (which uses only one walker) can cut down the message overhead by an order of magnitude compared to expanding ring across the network topologies. However, there is also an order of magnitude increase in user-perceived delay. To reduce the delay we increase the number of walkers, i.e. instead of just sending out one query message, a requesting node sends k query messages, and each query message takes its own random walk. The expectation is that k walkers after T steps should reach roughly the same number of nodes as 1 walker after kT steps, and indeed simulations confrm that. Therefore, by using k walkers, we can expect to cut the delay down by a factor of k. The k-walker random walk is a much more scalable search method than flooding.

3.4 Principles of Scalable Search in Unstructured Networks

The following conclusions can be reached from the analysis of the search techniques above:-

- *Adaptive termination is very important.* TTL-based mechanism does not work. Any adaptive/dynamic termination mechanism must avoid message implosion at the requester node. The checking method described above is a good example of adaptive termination.

- *Message duplication should be minimized.* Preferably, each query should visit a node just once. More visits are wasteful in terms of the message overhead.
- *Granularity of the coverage should be small.* Each additional step in the search should not significantly increase the number of nodes visited. This is perhaps the fundamental difference between flooding and multiple-walker random walk. In flooding, an additional step could exponentially increase the number of nodes visited; in random walk, it increases only by a constant.

4 Replication

In certain P2P systems such as Gnutella, only nodes that request an object make copies of the object. Other P2P systems such as Freenet allow for more proactive replications of objects, where an object may be replicated at a node even though the node has not requested the object. For such systems, we must answer the question: how many copies of each object should there be so that the search overhead for the object is minimized, assuming that the total amount of storage for objects in the network is fixed?

4.1 Problem Formulation

There are m objects and n sites(nodes). Each object i is replicated at r_i random (distinct) sites(recall that $R = \sum_i r_i$), and that object i is requested with relative rates q_i, where we normalize this by setting $\sum_i q_i = 1$. For convenience, we assume that query and replication strategies are such that $1 \ll r_i \leq n$ and that searches go on until a copy is found. Search consists of randomly probing sites until the desired object is found. Thus, the probability $Pr(k)$ that the object is found on the k'th probe is given by: $Pr_i(k) = \frac{r_i}{n}(1-\frac{r_i}{n})^{k-1}$. An object's average search size A_i is merely the inverse of the fraction of sites which have replicas of the object: $A_i = \frac{n}{r_i}$. We are interested in the average search size of all the objects: $A = \sum_i q_i A_i = n \sum_i \frac{q_i}{r_i}$.This metric essentially captures the message overhead of efficient searches. We assume that the average number of these replicas per site, $\rho = \frac{R}{n}$. The question is how to allocate these R replicas among the m objects.

4.2 Replication Strategies

1. **Uniform Replication**: The simplest replication strategy is to create the same number of replicas of each object: $r_i = \frac{R}{m}$. We call this uniform replication strategy. In this case the average search size $A_{uniform}$ is given by: $A_{uniform} = \sum_i q_i \frac{m}{\rho} = \frac{m}{\rho}$ which is independent of the query distribution.
2. **Proportional Replication**: It is clear that uniformly replicating all objects, even those that are not frequently queried, is inefficient. A more natural policy is to replicate proportional to the querying rate: $r_i = Rq_i$. This should reduce the search sizes for the more popular objects. However, a quick calculation reveals that the average remains the same: $A_{proportional} = n \sum_i \frac{q_i}{Rq_i} = \frac{m}{\rho} = A_{uniform}$
3. **Square Root Replication**: Given that Uniform and Propportional have the same average search size, a natural question is what is the optimal way to allocate the replicas so that the average search size is minimized? A simple calculation reveals that Square-Root

replication is optimal; that is, A is minimized when $r_i = \lambda\sqrt{q_i}$ where $\lambda = \frac{R}{\sum_i \sqrt{q_i}}$. The average search size is $A_{optimal} = \frac{1}{\rho}(\sum_i \sqrt{q_i})^2$.